



**ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL**

**Facultad de Ingeniería en Electricidad y Computación**

**“DISEÑO DE UN SISTEMA EMBEBIDO BASADO EN LA  
TARJETA DE DESARROLLO PYBOARD PARA EL CONTROL  
DE UNA BANDA TRANSPORTADORA.”**

**INFORME DE MATERIA INTEGRADORA**

Previo a la obtención del Título de:

**INGENIERO EN ELECTRICIDAD, ELECTRÓNICA Y  
AUTOMATIZACIÓN INDUSTRIAL**

**NICKY EDUARDO CÓRDOVA PLAZA**

**ITALO ADRIÁN ESPARZA VILLACÍS**

**GUAYAQUIL – ECUADOR**

**AÑO: 2017 - 2018**

## AGRADECIMIENTO

Mis más sinceros agradecimientos al PhD. Douglas Plaza al darnos la confianza de desarrollar este proyecto y recibir su apoyo a las varias dudas que teníamos respecto al tema, al apoyo financiero al proporcionarnos las dos tarjetas Pyboard y el módulo Wi-Fi.

Además, agradezco al Ing. Franklin Kuonqui por sus tutorías y paciencia al asesorar el presente documento y en muchos consejos que nos proporcionó para tener un mejor proyecto.

Agradezco también a la Ing. Sianna Puente por la ayuda prestada en el tema de la creación de la página web y por la paciencia al escuchar muchas de mis peticiones e ideas relacionadas con el proyecto, y discutir muchos temas interesantes ligados a este.

Gracias a mis compañeros Dan Cabrera y José Santana por su apoyo en algunos temas y materiales prestados en este proyecto y también los respectivos agradecimientos al Ing. Livintong Miranda por la ayuda y sugerencias que nos dio en el Laboratorio de Control de Procesos Industriales.

*Italo Adrián Esparza Villacís.*

En primera instancia mi agradecimiento es a Dios, por ser quien guía el destino de mi vida, y ser pieza fundamental para alcanzar este nuevo logro.

A mis padres Carlos Córdova y Martha Plaza, por su excelente educación que a través de sus enseñanzas me han permitido ser una persona de bien y de valores, además, de su apoyo incondicional durante esta etapa de mi vida.

Al PhD. Douglas Plaza por ser un excelente tutor, quien siempre estuvo predispuesto de brindarme información relevante para la elaboración del proyecto, gracias por confiar en nosotros. A la ingeniera Sianna Puente por apoyo y consejos para la resolución de ciertos problemas que se nos presentaron en el proyecto.

*Nicky Eduardo Córdova Plaza.*

## DEDICATORIA

El presente proyecto lo dedico a mis padres por ser ese pilar en mi vida, darme gran ejemplo a seguir y apoyo en las decisiones que he tomado. También dedico este trabajo a mis hermanas, Betzabeth y Samantha, por darme consejos y compartir su tiempo conmigo. A mi hermano pequeño por ser un amigo en tiempos de ocio y darme fuerzas.

Además, dedico este proyecto a mi compañera de vida y a quien ha estado a mi lado apoyándome sin dudar en todas mis decisiones, ella es Sianna Puente, e inclusive me proporciono sus conocimientos en el presente proyecto de materia integradora.

*Italo Adrián Esparza Villacís.*

Primeramente, el proyecto se lo dedico a Dios, quien me dio la oportunidad de superarme y ser mi fortaleza en cada meta que me propongo.

A mis padres Carlos Córdova y Martha Plaza, personas que amo y admiro mucho, por ser mis modelos a seguir en la vida, que siempre me han brindado su confianza, su apoyo para la obtención de este nuevo logro, por sus consejos que me han sido de gran ayuda para la vida diaria.

*Nicky Eduardo Córdova Plaza.*

# TRIBUNAL DE EVALUACIÓN

.....  
**PhD. Douglas Antonio Plaza Guingla**

PROFESOR COLABORADOR

.....  
**Ing. Franklin Ilch Kuonqui Gainza**

PROFESOR TUTOR

## DECLARACIÓN EXPRESA

"La responsabilidad y la autoría del contenido de este Trabajo de Titulación, nos corresponde exclusivamente; y damos nuestro consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

.....  
Nicky Eduardo Córdova Plaza

.....  
Italo Adrián Esparza Villacís

## RESUMEN

El presente proyecto consiste en la programación de la tarjeta de desarrollo Pyboard, cuya finalidad es diseñar un sistema automatizado para el control de una banda transportadora usando una tarjeta de desarrollo Pyboard programada con Micropython que sirva de modelo para la generación de soluciones de bajo costo para la pequeña industria nacional. Por otro lado, debido a que los PLC requieren un alto costo de inversión para su implementación, estas empresas deciden no implementar sistemas de control en sus líneas de producción, por esta razón, se presenta la tarjeta de desarrollo Pyboard programable en Micropython como una alternativa al uso de controladores lógicos programables, lo cual reduciría considerablemente la inversión en el desarrollo de estos sistemas, y además, se incentivaría a que la pequeña industria automatice sus procesos industriales.

La metodología de trabajo del proyecto se lo estructuró en tres etapas, en diseño electrónico, programación de las tarjetas Pyboards, y diseño de la página web. En el diseño electrónico para llevar a cabo el control del subsistema IMS10, se realizó una serie de diseños, que permitieron una interacción entre los actuadores y sensores hacia la tarjeta, en la programación de las tarjetas se implementó códigos responsables del control y automatización del proceso, y por último el diseño de la página web que representa el scada donde se puede controlar y monitorear el proceso.

Se obtuvo un sistema embebido conformado de tres placas impresas hechas en fibra de vidrio, dos tarjetas Pyboard y dos módulos wifi, protegidos por una caja de plástico industrial, con borneras para la alimentación, y adquisición/envío de señales, además, de un SCADA diseñado en una página web.

Se logró controlar eficientemente el subsistema IMS10 y la banda transportadora, usando dos controladores, debido a que el procesador de la Pyboard carece de la funcionalidad de multihilos, también, soluciona un problema económico, mediante el uso de la Pyboard y sus periféricos, que presentan una alternativa de automatización para procesos industriales, siendo más baratos, de código abierto, y flexible.

Palabras claves: MicroPython – Multihilos – Controlador – Adquisición – Página web.

## **ABSTRACT**

*The present project consists in the programming of the Pyboard, whose purpose is to design an automated system for the control of a conveyor belt using programmed Pyboard with Micropython that serves as a model for the generation of low cost solutions for the small national industry. On the other hand, PLCs require a high investment cost for their implementation, these companies decide not to implement control systems in their production lines, for this reason, the Pyboard is presented as an alternative to the use of programmable logic controllers, which it would reduce considerably the investment in the development of these systems, and in addition, it would encourage small industry to automate its industrial processes.*

*The working methodology of the project was carried out in three stages, electronic design, programming of the Pyboards cards, and design of the website. First, in the electronic design to carry out the control of the IMS10 subsystem, a series of designs were made, which allowed an interaction between the actuators and sensors towards the card. Then, in the programming of the cards, codes responsible for the control and automation of the process were implemented. Finally, the design of the website that represents the scada where you can control and monitor the process.*

*An embedded system was obtained consisting of three printed plates made of fiberglass, two Pyboard cards and two wifi modules, protected by an industrial plastic box, with terminals for the power supply, and acquisition / sending of signals, in addition, of a SCADA designed on a web page.*

*It was possible to efficiently control the IMS10 subsystem and the conveyor belt, using two controllers, because the Pyboard processor lacks the functionality of multithreading, also solves an economic problem, by using the Pyboard and its peripherals, which present an automation alternative for industrial processes, being cheaper, open source, and flexible.*

*Keywords: MicroPython - Multi-threads - Controller - Acquisition - Website.*

# ÍNDICE GENERAL

RESUMEN .....	I
ABSTRACT .....	II
ÍNDICE GENERAL .....	III
CAPÍTULO 1 .....	1
1. DELIMITACIÓN DEL PROBLEMA.....	1
1.1 Planteamiento del Problema.....	1
1.2 Objetivos.....	2
1.2.1 Objetivo General.....	2
1.2.2 Objetivos Específicos.....	2
1.3 Justificación.....	3
1.4 Alcance del Proyecto.....	4
CAPÍTULO 2 .....	6
2. METODOLOGÍA DE TRABAJO.....	6
2.1 Diseño electrónico.....	6
2.1.1 Circuito de aislamiento entre las etapas de control y potencia. ....	6
2.1.3 Circuito acondicionamiento de señales digitales del proceso. ....	7
2.1.4 Diseño PCB para placas de interfaz Pyboard – IMS10.....	11
2.1.5 Diseño PCB para placa motor y acondicionamiento de sensores SM1 y SM2.....	13
2.2 Programación de la tarjeta Pyboard. ....	15
2.2.1 Movimiento de banda transportadora con sensores. ....	15
2.2.2 Movimiento de banda transportadora sin sensores. ....	15
2.2.3 Movimiento de banda transportadora con sensores. ....	18
2.2.4 Control secuencial del subsistema IMS10 con Pyboard. ....	20
2.2.5 Proceso almacenar en el subsistema IMS10.....	22
2.2.6 Proceso liberar en el subsistema IMS10.....	27
2.3 Diseño de la página web. ....	31
2.3.1 Conexión a internet de Pyboard a sitio web.....	31
2.3.2 Arquitectura de la página web.....	35
CAPÍTULO 3 .....	36
3. RESULTADOS.....	36



3.1	Hardware.....	36
3.2	Software. ....	39
3.3	Página web.....	40

# CAPÍTULO 1

## 1. DELIMITACIÓN DEL PROBLEMA.

En este capítulo se fundamentará el planteamiento del problema, así como su justificación y objetivos, relacionado a la implementación de un sistema automatizado para el control de una banda transportadora usando la tarjeta de desarrollo Pyboard, con el fin que sirva de modelo para generar soluciones de bajo coste.

### 1.1 Planteamiento del Problema.

Las pequeñas empresas al empezar su producción generalmente no están en capacidad de automatizar sus procesos industriales debido a que los equipos necesarios requieren una mayor inversión en comparación a los utilizados en un proceso sin automatizar. Por esta razón, estas empresas deciden no implementar sistemas de control en sus líneas de producción para así reducir el gasto de inversión inicial.

Los controladores lógicos programables (PLC) representan un gasto considerable en el sistema de control, donde su costo de adquisición asciende aproximadamente a los 1500 dólares. Además, para el uso adecuado de un PLC es necesario comprar módulos adicionales como una Interfaz Humano-Máquina (HMI), módulos de alimentación, módulos de expansión, entradas/salidas digitales y analógicas, entre otros, debido a que estos no vienen incluidos lo cual incrementa el costo en cerca de 1000 dólares más. Otra complicación es que algunos de estos módulos solo se pueden usar exclusivamente en una versión específica de PLC y es imposible de utilizar en otra versión y ni mencionar el uso en alguna otra marca de controlador, en vista de que no existe compatibilidad entre estos equipos.

A pesar de que los PLC presentan grandes ventajas como la fácil integración con los equipos gracias al programa TIA PORTAL, robustez y confiabilidad en el sistema, fácil instalación gracias a sus dimensiones; se encuentran estancados en una mejora deficiente de hardware y software, a causa de

que la empresa desarrolla internamente las actualizaciones del de acuerdo con los criterios de cada fabricante. Más aun, estos controladores tienen un alto coste de software y licencias de operación; además, muchas plataformas solo funcionan en versiones específicas del sistema operativo Windows para la que fueron desarrolladas, tal es el caso de TIA PORTAL V13 el cual es diseñado íntegramente para Windows 7 y presenta errores si están instalados en otra versión.

Otro aspecto que considerar en las empresas que manejan controladores lógicos programables es que necesitan personal capacitado específicamente para el uso de estos equipos, ya que al cambiar o actualizar estos dispositivos se requiere de cursos adicionales para que se puedan implementar las nuevas herramientas, incrementando los gastos y el temor en las empresas a la migración a nuevas tecnologías.

## **1.2 Objetivos.**

### **1.2.1 Objetivo General.**

Implementar un sistema automatizado para el control de una banda transportadora usando una tarjeta de desarrollo Pyboard programada con Micropython que sirva de modelo para la generación de soluciones de bajo costo para la pequeña industria nacional.

### **1.2.2 Objetivos Específicos.**

- i. Desarrollar una etapa de acondicionamiento de señales digitales para su recepción y envío, entre la tarjeta Pyboard, la banda transportadora y el subsistema IMS10.
- ii. Programar el control secuencial en la tarjeta Pyboard, para la generación de señales PWM, comunicación UART con el módulo Wi-Fi ESP-8266 y control de las señales digitales E/S.
- iii. Elaborar una página web para la interacción entre el usuario y el proceso de la banda transportadora, de esta manera realizará la función de SCADA, permitirá la regulación de la velocidad, el paro

o marcha del proceso, el cambio del sentido de giro y el control del proceso del subsistema IMS10.

### **1.3 Justificación.**

En el país, las pequeñas empresas no pueden solventar gastos en implementación de sistemas de automatización para sus líneas de producción, por este motivo se presenta la tarjeta de desarrollo Pyboard programable en Micropython como una alternativa al uso de controladores lógicos programables, lo cual reduciría considerablemente la inversión en el desarrollo de estos sistemas a corto y largo plazo. De esta forma, se incentivaría a que la pequeña industria automatice sus procesos industriales.

Esta tarjeta de desarrollo al ser de software libre presenta las facilidades para obtener toda la información necesaria acerca de ella como: esquemáticos, actualizaciones de firmware, librerías para los nuevos módulos que aparecen en el mercado, que hayan sido diseñados para este o así mismo para los que sean compatibles y a los cuales la comunidad les ha diseñado alguna librería, entre otros y todo eso de forma gratuita. En consecuencia, cualquier desarrollador puede crear su propia versión de la tarjeta Pyboard de acuerdo con sus necesidades. Además, el lenguaje Python al tener una gran comunidad de programadores siempre va estar en permanente desarrollo y actualización de librerías y firmware, en busca de aprovechar el mayor rendimiento de la tarjeta, además de encontrar una gran variedad de información en la web, por lo tanto siendo esta la mayor ventaja en desarrollar sistemas embebidos de control basados en Micropython, debido a que el ahorro en licencias y hardware es considerable en contraste al uso de PLC como controlador principal.

Por otro lado, la tarjeta presenta una gran flexibilidad y compatibilidad con actuadores, sensores, pantallas LCD independientemente del fabricante, además de ordenadores que no necesariamente tiene que ser potentes, integrando estos dispositivos en un mismo sistema, sin problemas de compatibilidad, a diferencia de los PLC en donde si surgen estos inconvenientes.

Dentro de las fortalezas que motivan el uso de la tarjeta Pyboard es su lenguaje de programación Micropython, el cual es una versión de Python 3 orientado a microcontroladores. Por lo tanto, las empresas no van a tener inconvenientes en adquisición y renovación de licencia del software, adicionalmente es compatible con la mayoría sistema operativo del mercado.

Python actualmente se ha convertido en uno de los lenguajes de programación más básicos y de fácil aprendizaje entre programadores profesionales, intermedios o recién iniciados, de manera que esto incrementa el número de desarrolladores en diversas áreas. Inclusive el uso de la tarjeta se vuelve una herramienta interesante para aquellos estudiantes que cursan materias de fundamentos de programación, y no le encuentren una aplicación práctica a Python generando desinterés en los estudiantes. De esta forma al interactuar con la tarjeta y el sistema de control de la banda transportadora y observar un uso industrial a Python, despertará el interés y la curiosidad de seguir investigando y aprendiendo.

#### **1.4 Alcance del Proyecto.**

Para el control de la banda transportadora se utilizará control por modulación de ancho de pulso (PWM) y serán proporcionados por la tarjeta de desarrollo Pyboard, la cual estará aislada por una etapa de protección constituida por opto acopladores, de esta forma protegiendo el circuito de control de cualquier falla en la etapa de potencia. Para la recepción y envío de señales digitales provenientes de sensores y actuadores se diseñará una etapa de acondicionamiento de señales con amplificadores operacionales (OPAMP) LM324, de esta forma reducir y amplificar el nivel de voltaje.

En la tarjeta Pyboard se programará un control de tipo secuencial, utilizando el lenguaje de programación Micropython, teniendo en cuenta el estado de los sensores magnéticos localizados a la izquierda y derecha del subsistema IMS, y los requerimientos proporcionados por el usuario, interactuando con la interfaz humano-maquina desarrollada en la página web. Sin embargo, no se va a realizar un diagnóstico de sensores y actuadores al inicio del proceso, ni se mostrará la velocidad del motor.

Se diseñará un sitio web, donde se podrá controlar el sentido de giro del motor, se podrá enviar el ancho de pulso de las señales PWM y la frecuencia para regular la velocidad del motor. También se podrá observar cuando se active algún sensor magnético, se encontrarán los botones de marcha y paro del proceso. Adicionalmente para la comunicación Wi-Fi se utilizará el módulo ESP-07S y se enviará los datos al Pyboard por medio de comunicación UART, en este módulo Wi-Fi, utilizaremos comandos AT para la conexión a la red.

Se elaborará las respectivas placas de circuito impreso (PCB) para las etapas de control y acondicionamiento de señales, con sus respectivas protecciones.

Se presentará finalmente dentro de una estructura sólida que sirva de interfaz física para el usuario, con su respectivo botón de encendido/apagado, Reset, conector cable paralelo DB-9, borneras para conectar sensores, bornera de alimentación de 24 voltios de corriente directa (VDC), bornera de salida al motor. También esta estructura contendrá las placas de la etapa de acondicionamiento, control, modulo Wi-Fi y las tarjetas Pyboard, de esta manera tener una mejor presentación. Además, al sitio web se podrá acceder desde cualquier ordenador que tenga conexión a internet, para poder acceder al control del proceso.

## **CAPÍTULO 2**

### **2. METODOLOGÍA DE TRABAJO.**

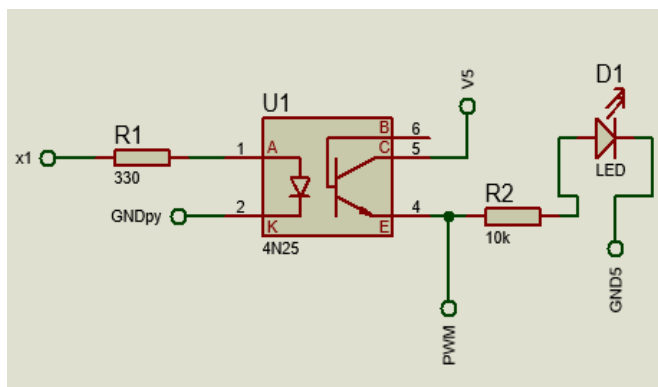
En este capítulo se detallan los métodos usados para el desarrollo del presente proyecto, además del proceso de programación de los códigos de control de la parte automática y manual de la banda transportadora, como también del proceso de diseño de las etapas de acondicionamiento de las señales digitales y página web.

#### **2.1 Diseño electrónico.**

Para llevar a cabo el control del subsistema IMS10, se realizó una serie de diseños electrónicos, que permitieron una interacción entre los diferentes tipos de actuadores y sensores hacia la tarjeta Pyboard, tales como, un circuito de aislamiento o de protección, un circuito de acondicionamiento de las señales para los sensores y actuadores, el circuito del módulo de potencia, así como, los diseños PCB para cada uno de los circuitos antes mencionados.

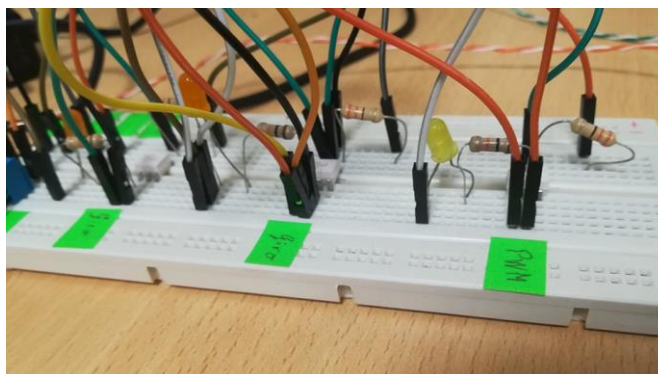
##### **2.1.1 Circuito de aislamiento entre las etapas de control y potencia.**

Para separar la etapa de potencia con la etapa de control (tarjeta Pyboard), se diseñó un circuito de aislamiento conformado por opto-acopladores. Se requirió un optoacoplador por cada señal que ingrese al puente H, en este caso son 3 señales, los pines X1, X2 y X3. Aparte de ello, el diseño de este circuito se lo realizó en PROTEUS, además, su funcionamiento fue respectivamente simulado y en la figura 2.1 se muestra el circuito con un opto-acoplador para aislar el pin X1 con la entrada PWM del puente H.



**Figura 2.1: Circuito para aislamiento de etapas para el pin X1 (PWM).**

Para realizar pruebas de funcionamiento del código programado en la tarjeta Pyboard en la planta IPA26, se implementó este circuito en protoboard, con el fin de que, si se presentaba algún fallo en el diseño, este pueda ser fácilmente corregido a tiempo, por esta razón en este circuito fue utilizado un opto-acoplador 4N25, resistencias de 1 kilohmios ( $k\Omega$ ) y un diodo LED, el cual indica que el PWM está activo o el cambio del sentido de giro, como se puede observar en la figura 2.2.



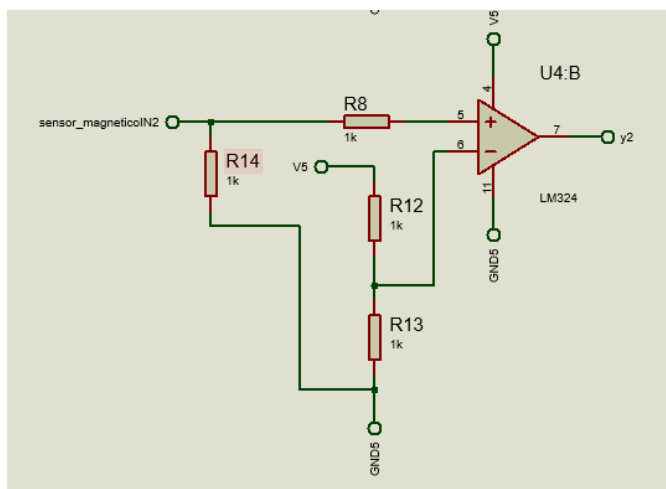
**Figura 2.2: Protoboard con el circuito de aislamiento.**

### 2.1.3 Circuito acondicionamiento de señales digitales del proceso.

Al utilizar la tarjeta Pyboard, la comunicación no se la realizó utilizando el esclavo PROFIBUS-DP y en esta tarjeta se encuentra conectado el sensor magnético B1, que se encuentran a izquierda del IMS10, y el sensor magnético B2, ubicado a la derecha del IMS10 (desde ahora serán llamados SM1 y SM2, respectivamente). En consecuencia se necesita de una etapa de acondicionamiento de señal, que baje el nivel de voltaje en alto de 24 VDC a 5 VDC, el cual es el voltaje tolerado por



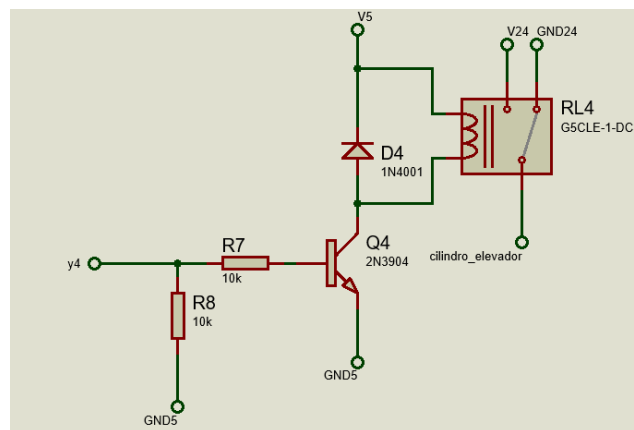
la tarjeta Pyboard, para cumplir con este cometido se diseñó un circuito comparador con un OPAMP LM324 alimentado con fuente dual de 5 VDC y 0 VDC, es decir que su salida será de 0 VDC cuando el valor del pin inversor sea menor al pin no inversor, en este diseño se ubicó en el pin inversor una red de división de voltaje alimentada por 3.3 VDC, con dos resistencias iguales de 1 k $\Omega$ , es decir el pin inversor tiene un voltaje de 1.65 VDC, al hacer esto se resolvió el problema que se tenía cuando el sensor estaba en estado lógico bajo y de esta forma se pueda detectar el cambio en el pin inversor, ya que de esta forma se puede comparar los 0 VDC con los 1.65 VDC, cosa que no se podría detectar si el pin inversor está conectado directamente a tierra ( GND), de esta manera la salida del OPAMP es de 0 VDC en ese caso. También, cuando el sensor este en alto, el pin no inversor tendrá 24 VDC, de esta manera el pin no inversor será mayor al pin inversor que tiene 1.65 VDC en su terminal, haciendo que la salida del OPAMP sea de 5 VDC. Por último, este circuito fue simulado en PROTEUS y se indica a continuación en la figura 2.3.



**Figura 2.3: Circuito para bajar el nivel de voltaje de 24 a 5 VDC.**

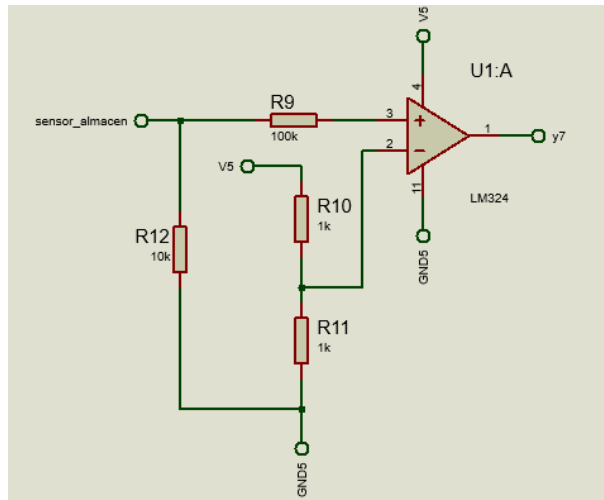
La estructura del subsistema IMS10 contiene varios actuadores que son controlados por electroválvulas de 24 VDC y un sensor fin de carrera (el cual será nombrado desde ahora como SFC), que detecta si en la estructura se encuentran alojados un número máximo de pallets, la señal de este sensor se encuentra dentro del cable DB-9 que sale del IMS10. Así

mismo, el SFC tiene un funcionamiento muy parecido a los sensores magnéticos SM1 y SM2, por lo que se puede acondicionar su señal de igual forma que estos con un OPAMP dentro del empaquetado LM324, caso contrario será con los actuadores, con estos no se puede utilizar OPAMPS ya que si bien entrega el nivel de voltaje necesario, este no entrega la corriente para que la válvula electroneumática pueda estar cien por ciento funcional, se realizó pruebas con el empaquetado LM324 y aunque se encendía el LED rojo indicando que la válvula del actuador se encendió, el pistón del cilindro nunca se activaba, razón por la cual se optó por trabajar con relés para que el nivel de voltaje (24 VDC) y corriente se suministrada por la fuente directamente, este circuito se detalla en la imagen de la figura 2.4, en donde se observa el circuito de fuerza para el actuador de cilindro elevador.



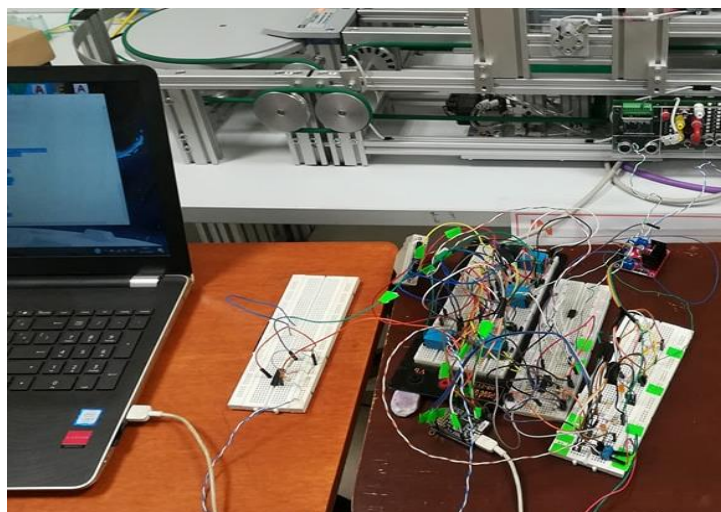
**Figura 2.4: Circuito para activar actuadores diseñados en PROTEUS.**

Todos los actuadores tendrán el mismo circuito para acondicionamiento de señal, pero hay que destacar que el sensor SFC tiene el circuito parecido que los sensores magnéticos SM1 y SM2, pero la resistencia R9 señalada en la figura 2.5, tiene que ser mayor que la utilizada en el circuito de SM1, ya que el sensor SFC tiene una corriente de salida más grande y esta resistencia debe absorber esa corriente, dado que en la tarjeta solo importa el nivel de voltaje y no la corriente, además que esta corriente debe ser lo más mínima posible para evitar fallos en la tarjeta Pyboard.



**Figura 2.5: Circuito acondicionador para sensor fin de carrera.**

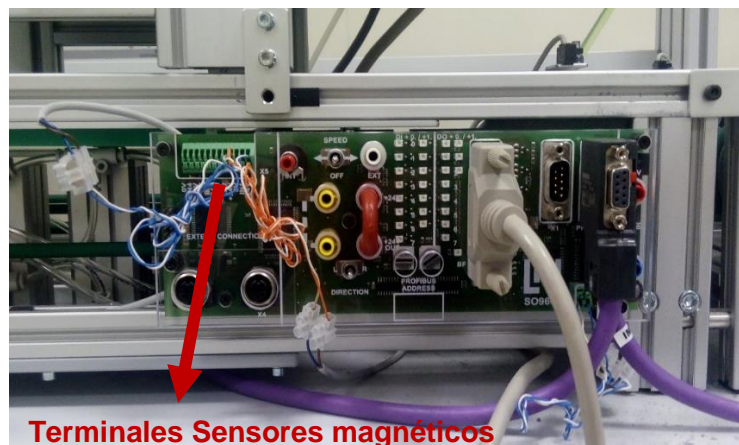
Se implementó en protoboard, cuatro de los circuitos con los relés para que active los correspondientes actuadores: cilindro de parada, cilindro separador, cilindro paralelo y cilindro elevador. Además, se utilizó el OPAMP LM324 para tener el comparador que reduzca la señal de voltaje proveniente del sensor SFC. También de los relés sale el cable que va hacia las válvulas electroneumáticas, los cuales están por default desactivados, pero con una señal de alto de 24 VDC estos se activan, aparte de ello el sensor SFC del pin 7 se conectó al circuito de acondicionamiento de señal. Los protoboard y las conexiones con el subsistema IMS10 son mostrados en la figura 2.6.



**Figura 2.6: Conexiones IMS10 - Protoboard - Pyboard.**

#### 2.1.4 Diseño PCB para placas de interfaz Pyboard – IMS10.

Con las simulaciones y pruebas en protoboard realizadas con éxito, se dispuso al diseño de las placas para posteriormente imprimirlas en fibra. Serán dos placas en total y se clasificarán según su uso y la disponibilidad de los sensores. La primera placa se encargará exclusivamente del acondicionamiento de las señales digitales provenientes del IMS10 y la segunda placa tendrá a cargo el control del Puente H y de acondicionar la señal digital de los dos sensores magnéticos SM1 y SM2, esto se debe a que estos sensores no se encuentran dentro del cable paralelo del subsistema, sino que se conectan de forma directa como se indica en la figura 2.7, al igual como se conecta el motor, directamente en al esclavo PROFIBUS-DP.



**Figura 2.7: Puerto de entrada para sensores externos en el esclavo PROFIBUS - DP.**

En esta placa se conecta la tarjeta Pyboard, específicamente los pines Y3, Y4, Y5, Y6, Y7 y GND, La tierra de la tarjeta está conectada con la tierra de la fuente de 5 VDC para tener igual referencia. Las señales de los pines de la Pyboard activan los 4 Relés que se encuentran sobre la tarjeta y habrá un pin digital de entrada a la tarjeta de control, el cual será Y7, siendo este el sensor SFC, por tal razón en la PCB se encuentra el circuito comparador diseñado con OPAMPS y alimentado con 5 VDC y GND. En esta placa se podrá conectar el DB-9 que proviene del subsistema IMS10, por esta razón se puede notar la presencia del CONN-D25F (Conector DB-25 Hembra) con sus respectivos pines.

También se observa las borneras de 24 VDC y 5 VDC. Este modelo se muestra en la figura 2.8 y su respectivo PCB se muestra a continuación en la figura 2.9.

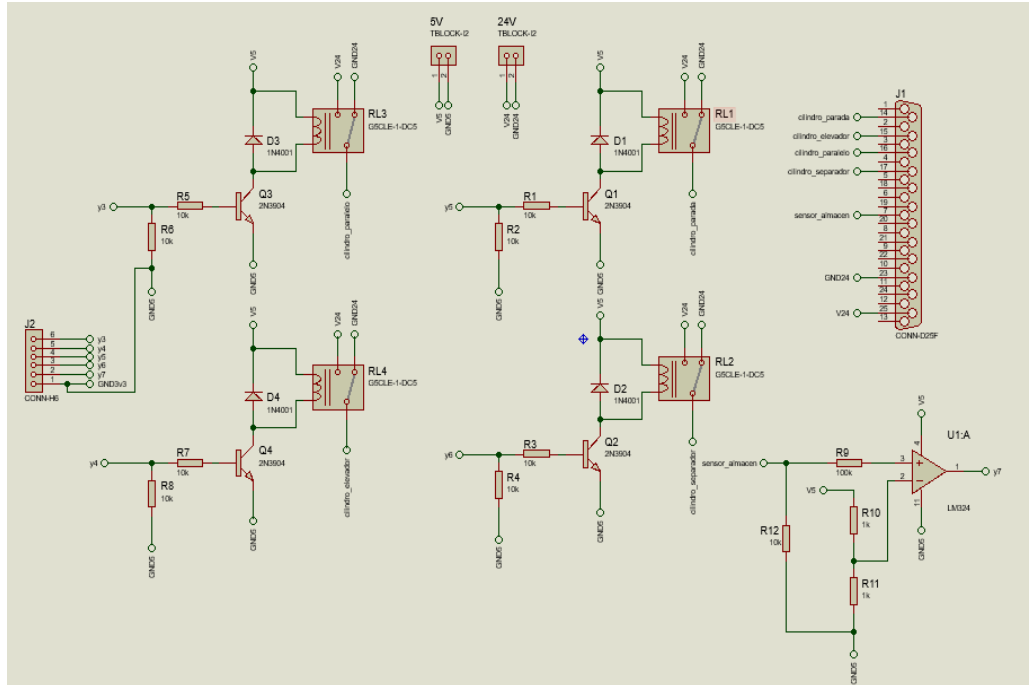


Figura 2.8: Diseño en Proteus de la placa para interfaz entre Pyboard - IMS10.

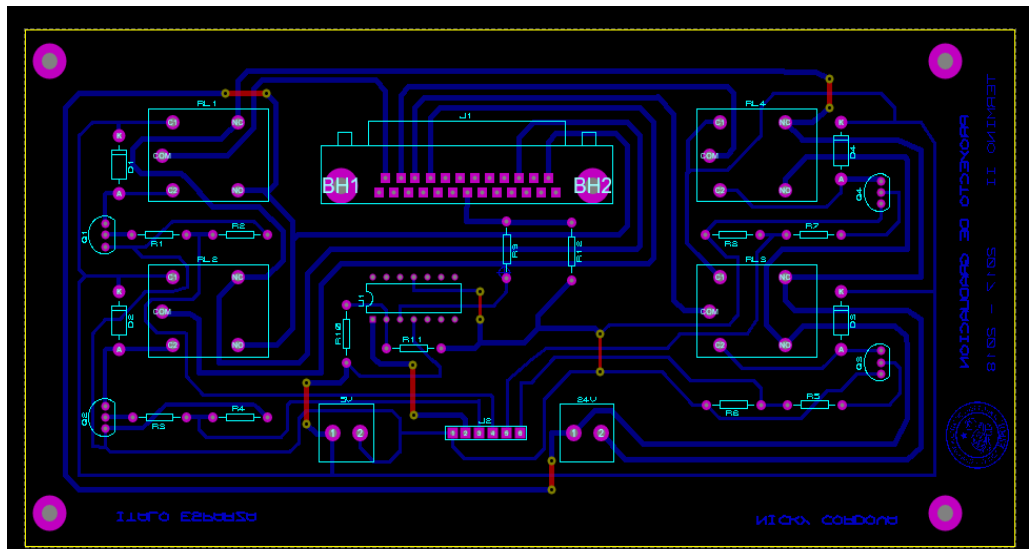


Figura 2.9: PCB Interfaz Pyboard - IMS10.

### **2.1.5 Diseño PCB para placa motor y acondicionamiento de sensores SM1 y SM2.**

Esta placa es la interfaz de conexión entre el puente H y la tarjeta Pyboard, además incluye los circuitos de acondicionamiento de señal de los dos sensores SM1 y SM2, los cuales se encuentran fuera del cable paralelo del subsistema IMS10, de esta manera la placa contiene la parte de potencia para el control de la banda transportadora, las protecciones en donde se utilizan los optoacopladores 4N25, este dispositivo aislará la tierra de la tarjeta (GNDpy) de la tierra de 5 VDC, la cual está conectada con la tierra de 24 VDC. De esta forma, se podrá observar 3 opto-apladores en la placa y del pin 4 se tienen sus correspondientes salidas: PMW, GIRO1 y GIRO2; correspondientes a las señales X1, X2, X3 provenientes de la tarjeta Pyboard. Además, de las protecciones se encuentra a los dos comparadores con OPAMPS que solo utiliza un empaquetado LM324, estos comparadores acondicionan la señal y bajan el nivel de voltaje de los sensores SM1 y SM2.

Las entradas y salidas de la placa serán de los pines de la Pyboard siendo estos X1, X2, X3, Y1, Y2 y GNDpy, también se debe alimentar con voltaje de 5 VDC, 24 VDC, los cuales tienen sus respectivas borneras de entrada. También, los sensores SM1 Y SM2 ingresan a la placa por medio de borneras ubicadas a un lado de las borneras de alimentación, adicionalmente se tiene las salidas de la placa hacia el puente H y el motor. De esta manera, el puente H recibe las señales PWM, GIRO1 y GIRO2, se alimenta con 24 VDC, 5 VDC y GND24 (Tierra de la fuente de 24 VDC). Todo lo indicado se muestra en la figura 2.10.

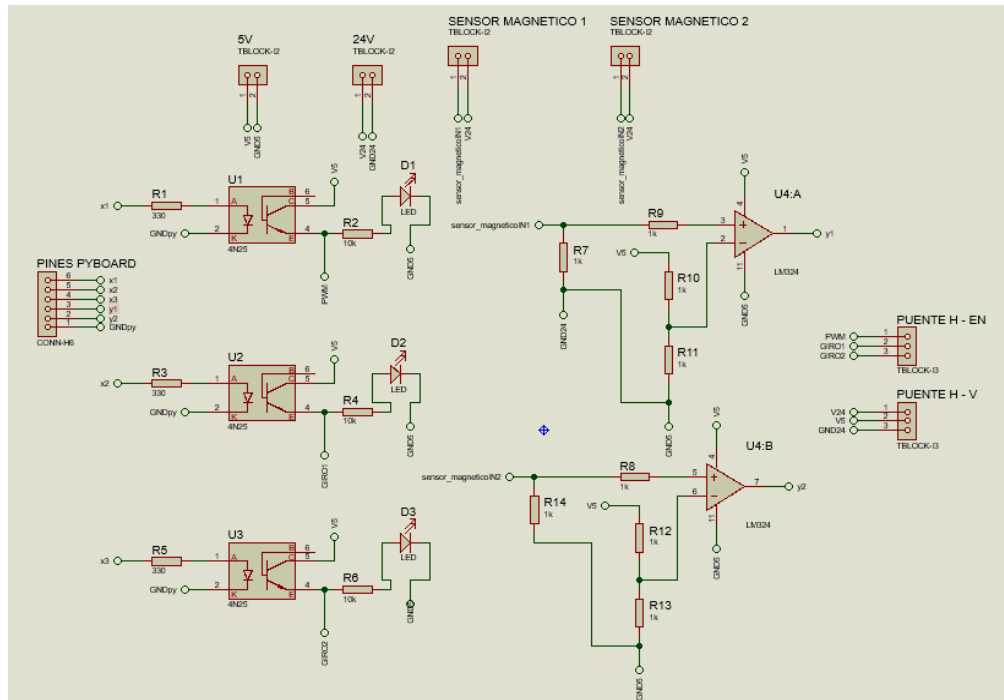


Figura 2.10: Diseño en Proteus para la interfaz Pyboard - Puente H, sensores SM1 y SM2.

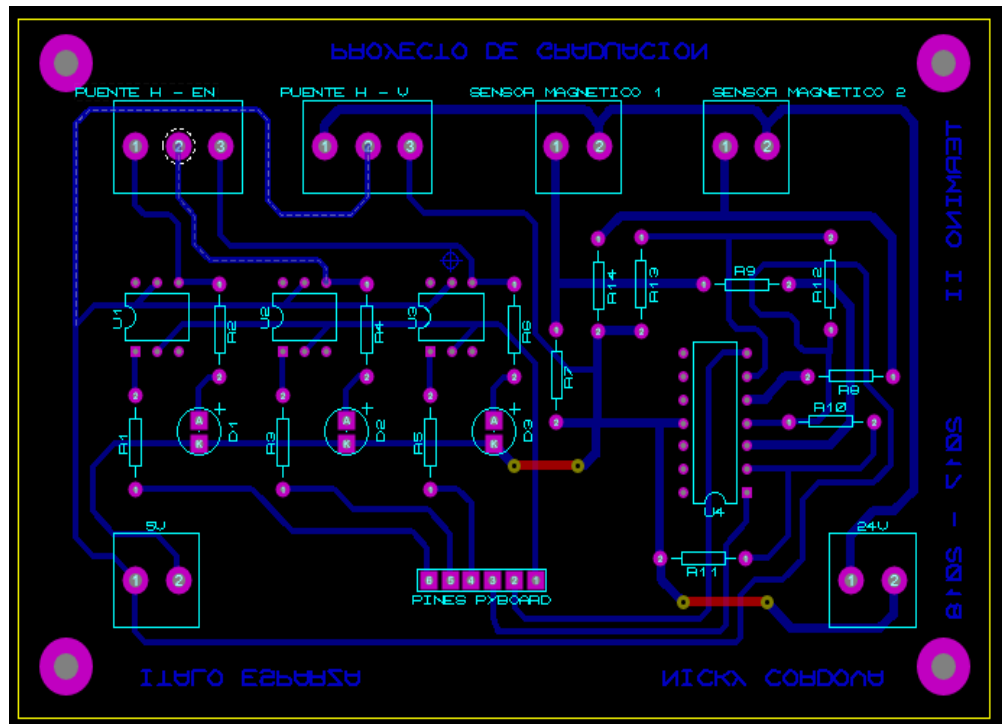


Figura 2.11: PCB Interfaz Pyboard - Puente H, sensores SM1 y SM2.

## **2.2 Programación de la tarjeta Pyboard.**

Para el control de la banda transportadora y su uso en el subsistema de almacenamiento intermedio IMS10, se utilizará la tarjeta de desarrollo Pyboard, la programación de este dispositivo se la realizó dentro de su memoria FLASH, en donde se pudo crear los archivos necesarios para la implementación del proceso automatizado de almacenamiento y liberación de pallets dentro del IPA26. Además, el programa debe estar dentro del archivo main.py, se realizaron algunas pruebas iniciales para entender el funcionamiento de los pines de la tarjeta y de las librerías exclusivas que esta posee dentro de Python versión 3 que esta simplificado para microcontroladores (Micropython), uno de estos es el “Programa intensidad de leds”, que se encuentra en Anexo 1, literal a.

### **2.2.1 Movimiento de banda transportadora con sensores.**

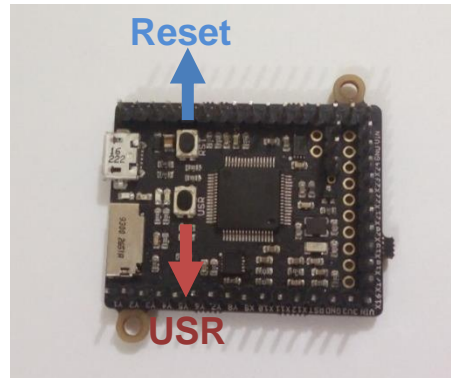
Para el desarrollo del código se profundizó explícitamente en las librerías que se utilizaron, como es en el caso de la implementación del PWM en el control de los pulsos y lo pines habilitadores del troceador de clase D (puente H L298), y en el código se puede observar la forma de cómo se utilizaron las salidas PWM asignadas a cualquier pin de la tarjeta. Para utilizar PWM se asignó un temporizador y de esta manera habilitar el canal (channel) con una determinada frecuencia y ancho de pulso, los cuales pueden ser posteriormente reemplazados con otros valores numéricos a fin de regular la velocidad. En esta serie de pruebas iniciales se observó que el motor solo acepta altas frecuencias, y deben ser mayores a 2000 Hz para un correcto funcionamiento y el rango de ancho pulso varía entre 0 y 100, el código utilizado en estas pruebas está en el Anexo 1, literal b.

### **2.2.2 Movimiento de banda transportadora sin sensores.**

Una vez que se tuvo una idea de cómo programar en la tarjeta Pyboard, se planteó realizar la programación de un código principal para el control de la banda transportadora; es decir, que regule la velocidad y



que cambie el sentido de giro cuando se presione el botón USR (botón Usuario); el botón se indica en la figura 2.12.

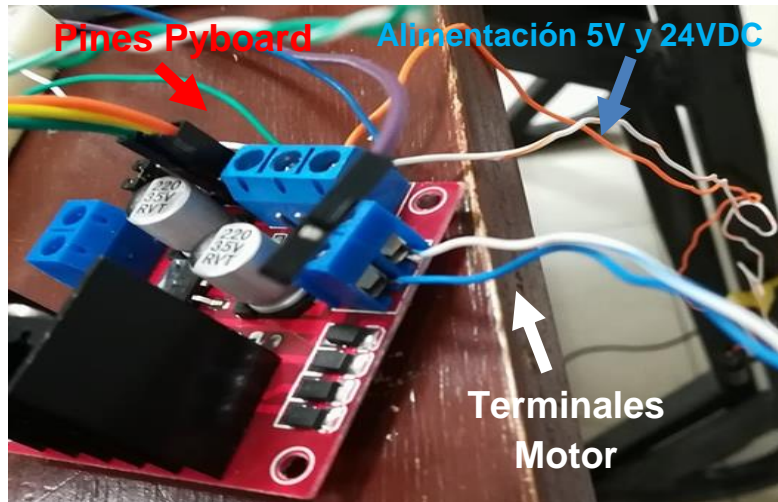


**Figura 2.12: Tarjeta Pyboard en la cual se indica botón USR y Reset.**

Para llegar a un código funcional, se programó el movimiento y regulación de velocidad de la banda transportadora sin el uso de sensores, además se realizaron pruebas en la parte posterior de la planta IPA26, en el esclavo de comunicación PROFIBUS-DP del subsistema IPA5 para no interferir en otras actividades dentro del laboratorio. En este esclavo se desconectaron los terminales del motor de corriente continua de 24 VDC y se conectó al puente H, como se muestra en la figura 2.13, y del terminal de pines del puente H se conectaron a los pines X1, X2 y X3 de la tarjeta Pyboard. Luego, en el código que se programó para el control de la banda, X1 es la salida del PWM, X2 y X3 son los habilitadores para el cambio de giro del puente H (EN1 y EN2), esto se muestra en la figura 2.14.



**Figura 2.13: Esclavo PROFIBUS - DP, con entrada DB) y terminales del motor.**



**Figura 2.14: Conexiones realizadas en el puente H.**

En el script main.py se tiene el programa en el que al inicio del código se deben importar las librerías, se inicializan los pines de salida X1, X2 y X3, el canal del PWM y el temporizador Timer 2, hay que tener en consideración que los Timer del 1 al 4 son los que están reservados para el de PWM, el Timer 3 está reservado como un temporizador interno de la tarjeta, y los Timer 5 y 6 están reservados para el uso con servomotores, se tiene que usar los temporizadores adecuados o podría provocar errores al ejecutar el código si se utiliza temporizadores en otros usos que no son a los que están destinados.

En el programa “Control banda IPA5”, detallado en Anexo 1 en el literal c, inicialmente se mueve la banda transportadora en un sentido a una velocidad media y si se presiona el botón USR, la banda se mueve en sentido contrario, en esta parte del código se reguló las frecuencias, y con esto se tendrá ciertos límites, es decir, una frecuencia inferior y una superior. De esta manera el límite superior fue de 4000 Hz, más allá de esto la velocidad no varía, pero pone en riesgo el motor; y a menos de 2000 Hz el motor empezó a emitir un sonido agudo, indicando que se está forzando a funcionar. Por lo tanto, estos dos límites deben ser restringidos al usuario para evitar el mal funcionamiento en el proceso.

### 2.2.3 Movimiento de banda transportadora con sensores.

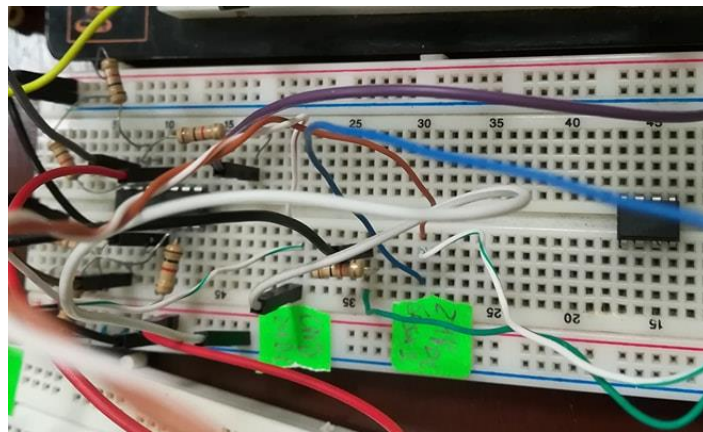
En el código mostrado en el Anexo 1, literal c, se controló el motor de la banda, pero sin el uso de sensores, así que en el programa que controla el movimiento del motor de la banda transportadora anterior se incluyó los sensores para que interactúen con el movimiento de la banda. Antes de agregar los sensores en el programa y conectarlos a la tarjeta, se procedió a estudiar el funcionamiento del sensor ya que este no es digital, los sensores magnéticos con los que cuentan toda la planta son de tipo relé, es decir que requiere estar conectado en el extremo de alimentación (cable café) a 24V y el extremo de salida del relé a un resistor para que cierre el circuito del relé interno del sensor magnético, y la señal que sale de entre el terminal IN y el resistor, será de 24V si el sensor detecta el imán del pallet o de 0V si no lo detecta, los sensores magnéticos se detallan a continuación en la figura 2.15, como se observa se encuentran al costado de la estructura de la banda transportadora.



**Figura 2.15: Sensor magnético B1, situado a la derecha del IMS10.**

Luego, se montó este circuito en protoboard para probarlo con los sensores en un programa sencillo, el programa es “Prueba sensores” y se detalla en Anexo 1 en el literal d, igualmente el circuito en protoboard se observa en la figura 2.16, el código está programado para que cuando se detecte el sensor magnético en alto se encienda un led de la tarjeta, o si no, se mantenga apagado. Se observa que el pin está configurado como un pin PULL\_DOWN, con esta configuración, el pin

internamente está conectado a tierra y solo cambiará de estado cuando el pin este en alto, el significado de “en alto” significará que el pin está a 3.3 VDC (VCC). El uso de la configuración PULL\_DOWN, fue un desacierto ya que, al mínimo del sensor, la tarjeta lo tomaba como un alto, y se decidió a tomar la configuración PULL\_UP, ya que está a pequeñas variaciones de voltaje en el sensor no lo tomaba como un alto, sino como un bajo.



**Figura 2.16: Circuito acondicionamiento de señales digitales en protoboard.**

Con la adición de las señales de los sensores, se pudo interactuar un poco más con el movimiento de la banda, ya que se pudo hacer que cuando se active un sensor, disminuya la velocidad de la banda y si detecta el otro sensor cambie el sentido de giro de la banda para volver al principio y volver otra vez a girar en el sentido inicial y a una velocidad alta. Para realizar esta prueba se conectó nuevamente el puente H en el motor DC de la banda transportadora del IPA5 y los pines X1, X2, X3 de la tarjeta en los pines habilitadores del puente H y los pines Y1 y Y2 a los pines 1 y 7 del OPAMP LM324, que son las salidas del circuito comparador, Y1 corresponde al SM11 y Y2 al SM2.

En el código “Control-secuencial Banda con Sensores”, indicado en Anexo 1 en el literal e, el movimiento inicial será el movimiento de la banda transportadora hacia la derecha con velocidad máxima, con una frecuencia de 2500 Hz y cuando detecte el sensor SM1 esta velocidad disminuye pero se sigue moviendo hacia la derecha, para este efecto

en el programa se disminuye el ancho de pulso a 45%, cuando se detecta el sensor SM2 este cambia su giro y la banda avanza hacia la izquierda por 15 segundos y luego regresa a su movimiento inicial de moverse con velocidad máxima y hacia la derecha.

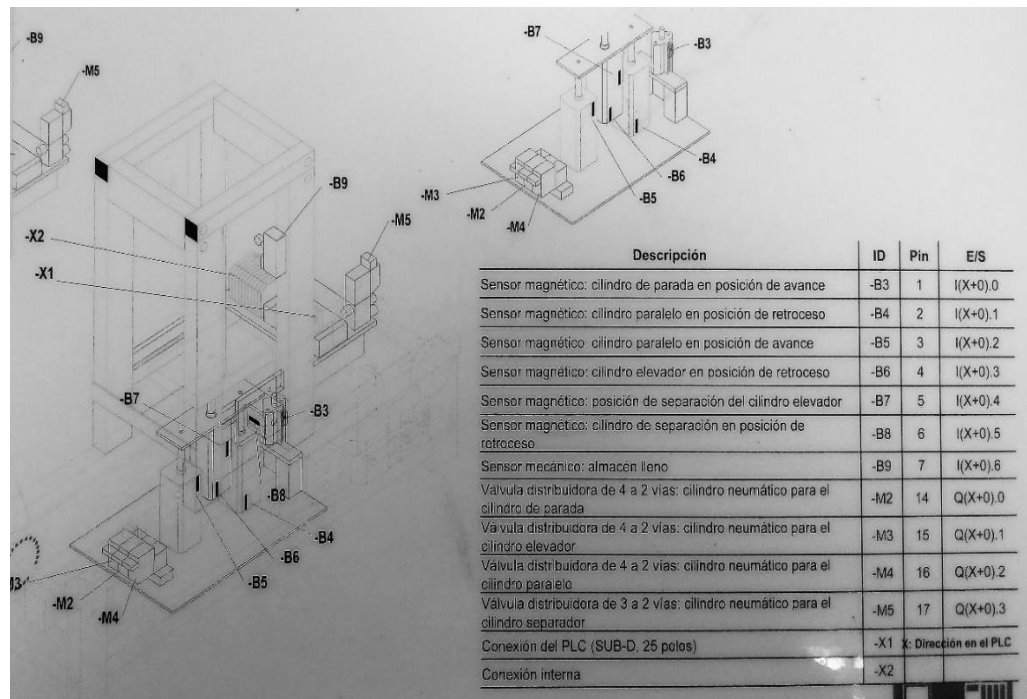
También se programó un código parecido al control secuencial mostrado en “Control-secuencial Banda con Sensores”, en que se tiene la secuencias anidadas y se observa claramente que una acción es consecuencia del orden de la activación de los sensores, es decir, primero se debe activar SM1 para que se pueda detectar SM2, ya que si no se activa SM1 la secuencia se quedará realizando lo que se encuentra antes y no después del IF que valida que se detectó el sensor SM1, se adjunta el programa “Control secuencial anidado” en Anexo 1, literal f. Además, este tipo de programación debe ser exacta y se programa de igual manera que en TIA PORTAL, en el PLC s7-300, como si se utilizará lenguaje Ladder o metodología GRAFCET.

#### **2.2.4 Control secuencial del subsistema IMS10 con Pyboard.**

Con el funcionamiento correcto del movimiento de la banda utilizando sensores se pudo utilizar la base del programa anterior para incluirlo en el control secuencial del subsistema IMS10, para realizar la programación del código se siguió el proceso desarrollado por este subsistema, el cual se programó de igual manera en TIA PORTAL y es cargado en el PLC, además se puede observar este código programado en bloques en el Anexo 2. El proceso de Almacenamiento intermedio del subsistema IMS10, se divide en dos, el proceso de almacenar pallets, en que se apilaran 4 pallets con sus respectivos porta sixpacks hasta que se active el sensor fin de carrera, y el proceso de liberar pallets desde la estructura metálica del IMS10.

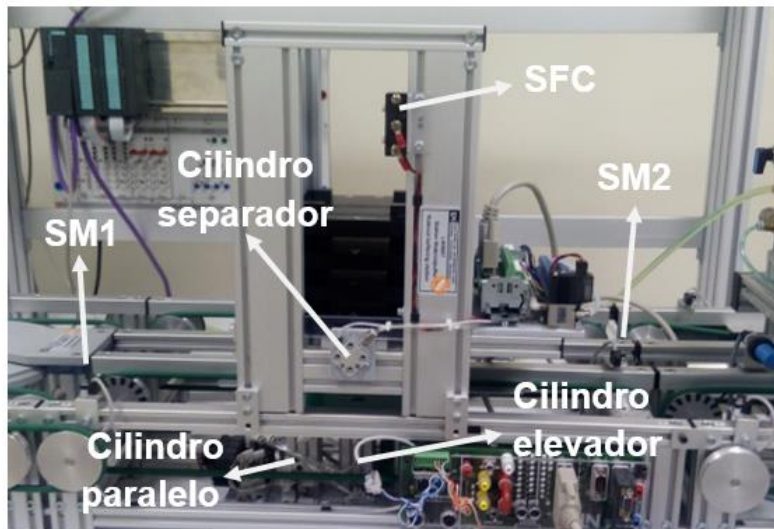
Para controlar ambos procesos se debe tener acceso a los sensores y actuadores de la planta, y ya que no se utilizará el esclavo PROFIBUS-DP que cuenta el IMS10, se desconectará el motor y los sensores SM1 y SM2 al igual como se lo hacía para en la sección 2.2.3 de tal manera

que para poder utilizar los sensores y actuadores mostrados en el Quickchart de la figura 2.17, se debió desconectar el cable paralelo con el conector DB-25 que se encuentra en el esclavo de comunicación, ya que este cable comunica con todo el subsistema IMS10, de esta forma y siguiendo lo indicado en el Quickchart se puede saber cuáles son los pines que el programa utilizará, además hay que destacar que este subsistema debe ser alimentado por 24 VDC y GND (Tierra), por medio del cable paralelo.



**Figura 2.17: Quickchart del subsistema IMS10.**

Los pines que se utilizaron del DB-25, fueron: el Pin 7, el cual es el sensor de fin de carrera (que se denominará SFC desde ahora); el Pin 14, el cual es el actuador de cilindro de parada; el Pin 15, el cual es el actuador de cilindro elevador; el Pin 16, el cual es el actuador de cilindro paralelo y el Pin 17, el cual es el actuador de cilindro separador. Estos sensores y actuadores se indican en la figura 3.18 en una imagen del subsistema IMS10.



**Figura 2.18: Detalle de los sensores y actuadores en IMS10.**

Se realizó una prueba previa para corroborar que los sensores y actuadores trabajan en perfectas condiciones, para el sensor utilizamos el programa de prueba anterior “Prueba sensores”, indicado en Anexo 1 y para los actuadores se utilizó el programa de prueba “Prueba actuadores IMS10” mostrado en Anexo 1, literal g, en este código solo se activan los actuadores secuencialmente, primero se debe presionar el botón de usuario USR de la tarjeta Pyboard y con esta acción se activan los cilindros paralelos, haciendo que los pistones de la parte de abajo del IMS10 se extiendan, luego de esta acción si se presiona de nuevo el botón USR, el cilindro elevador se extenderá, teniendo de esta manera los cilindros paralelos y elevador extendidos, de esta manera, simulando que se almacena un pallet dentro de la estructura metálica del IMS10, posteriormente si presionamos el botón RESET de la tarjeta, los cilindros se desactivarán y volverán a su estado inicial.

### **2.2.5 Proceso almacenar en el subsistema IMS10.**

Para el proceso de almacenar se utilizaron los sensores y actuadores del subsistema IMS10, la banda transportadora que moverá los pallets para posicionarlos bajo la estructura metálica del subsistema y para precaución se desconectó la tarjeta de comunicación PROFIBUS-DP, ya que no se utilizó el PLC, sino la tarjeta Pyboard como controlador del proceso.

Para la programación del código de la parte de almacenar se sigue la metodología Grafcet y se planteó todo el proceso en etapas y condiciones de transición, en las etapas se observa el cambio de estado de los actuadores y en las condiciones de transición se evalúan los estados de los sensores, el proceso será detallado en la tabla mostrada a continuación en la tabla 2.1.

<b>Etapas 1</b>	No se realiza acción
<b>Condición de transición 1 a 2</b>	Estado de Marcha_Almacenar en alto, en este caso este botón será el USB ubicado en la tarjeta Pyboard.
<b>Etapas 2</b>	Se mueve banda transportadora hacia la derecha a velocidad alta. X1 en alto (PWM), X2 en alto (En1) y X3 en bajo (En2).
<b>Condición de transición 2 a 3</b>	Sensor magnético SM1 (B1 en Quickchart) ubicado en la parte izquierda del IMS10, se debe activar para pasar a la siguiente etapa.
<b>Etapas 3</b>	Se reduce la velocidad de la banda transportadora y se activa cilindro de parada (M2 en Quickchart).
<b>Condición de transición 3 a 4</b>	Tiempo de espera de 4 segundos, para que el pallet pueda avanzar y el cilindro de parada lo detenga y se quede justo bajo la estructura del IMS10.
<b>Etapas 4</b>	Se activa el cilindro paralelo (M4), el cual subirá el pallet, se desactiva cilindro de parada (M2) y la banda transportadora se detendrá, es decir, X1 esta desactivado.
<b>Condición de transición 4 a 5</b>	Tiempo de espera de 5 segundos, hasta que los pistones del cilindro paralelo se extiendan correctamente.
<b>Etapas 5</b>	Se activa el cilindro elevador (M3), de esta forma el pallet empujará los cilindros separadores y podrá subir y quedarse

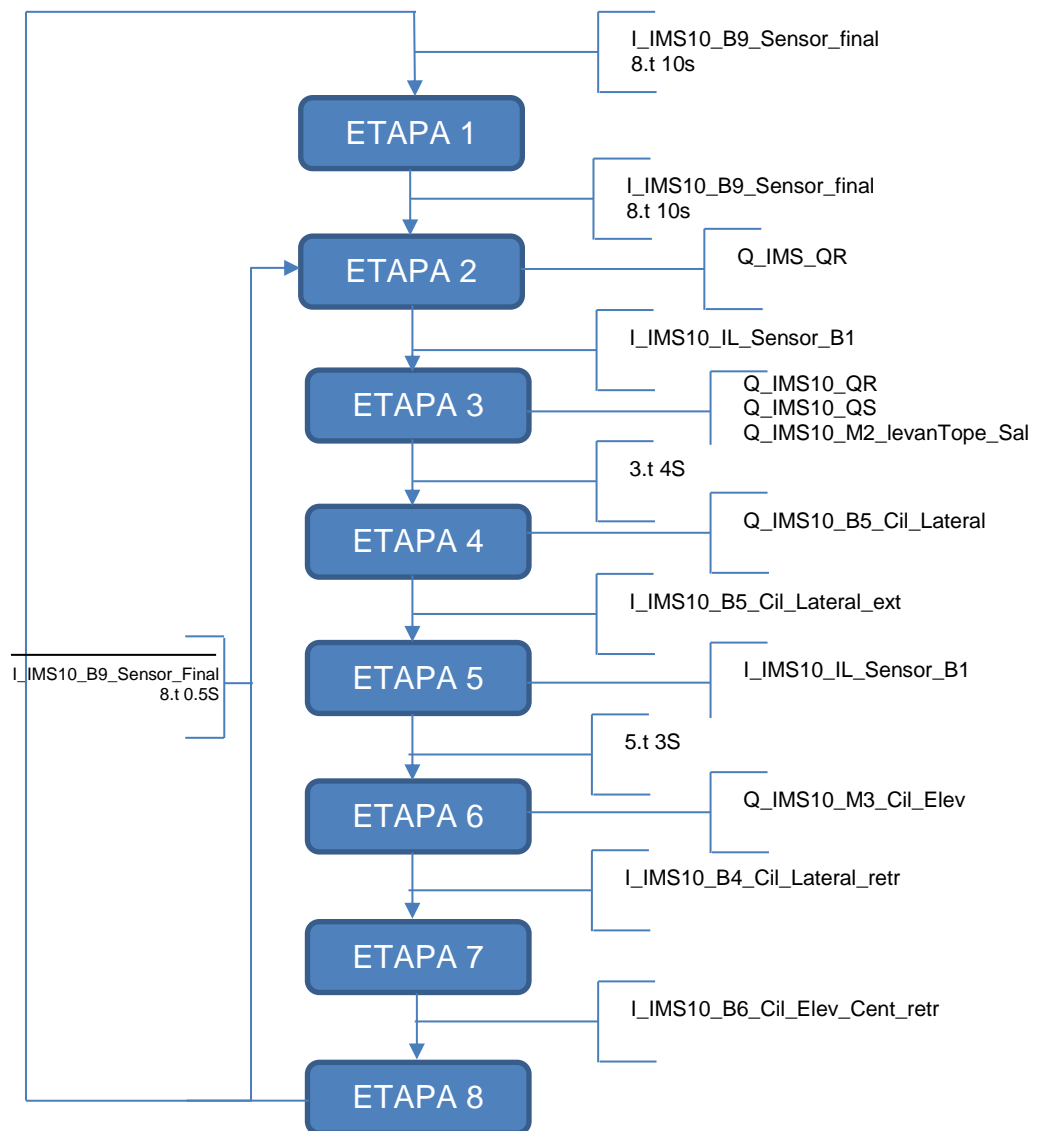


	suspendido sobre los cilindros separadores, de esta forma se almacenará en la estructura. Hay que considerar que el cilindro paralelo (M4) seguirá activo.
<b>Condición de transición 5 a 6</b>	Tiempo de espera de 8 segundos para que se pueda realizar las acciones anteriores.
<b>Etapas 6</b>	Se desactiva el cilindro paralelo (M4) para que la estructura que lleva al pallet para almacenado descienda.
<b>Condición de transición 6 a 7</b>	Tiempo de espera de 5 segundos para que el pistón del cilindro paralelo (M4) este contraído.
<b>Etapas 7</b>	El cilindro elevador (M3) es desactivado y de esta forma la planta regresará a su estado inicial.
<b>Condición de transición 7 a 8</b>	Tiempo de espera de 10 segundos.
<b>Etapas 8</b>	No se realiza acción.
<b>Condición de transición:</b> - 8 a 1 - 8 a 2	Se revisa el estado de sensor de fin de carrera SFC (B9), si está en estado lógico alto, se vuelve a la <b>Etapas 1</b> , y si está en estado lógico bajo regresa a la <b>Etapas 2</b> .

**Tabla 2.1: Graficet del proceso almacenar en el IMS10.**

En la tabla se explica todo el proceso para almacenar 4 pallets cargados con sus respectivos porta-sixpacks y el sixpack de botellas dentro de la estructura del IMS10, además se observan los sensores y actuadores que se utilizan en cada condición de transición y en cada etapa. En este diagrama Graficet se tiene dos lazos, en el primero se está dentro cuando el sensor SFC este desactivado, esta condición nos

indica que el IMS10 todavía se encuentra vacío o con menos de 4 pallets, y está en condiciones de seguir almacenando más pallets. También, se validó la condición que no se entre al proceso almacenar cuando este activa la señal de SFC, es decir, que no trate de almacenar pallets cuando el IMS10 se encuentra lleno. El diagrama secuencial del Grafset ya mostrado en la tabla 2.1, se muestra a continuación en la Figura 2.19.



**Figura 2.19: Diagrama secuencial Grafset del proceso almacenar.**

En base al diagrama secuencial se programó el código para que cumpla con lo requerido en el proceso Almacenar, los pines que se utilizaran en la tarjeta Pyboard equivalentes a los sensores son

mostrados en la tabla 2.2 a continuación. Se detallan los pines de la tarjeta Pyboard, los pines del DB-9 y su correspondiente descripción.

<b>Pin Pyboard</b>	<b>Pin DB-25</b>	<b>Descripción</b>
X1	-	Salida PWM de la tarjeta al Puente H.
X2	-	Habilitador para el cambio de giro del Puente H.
X3	-	Habilitador para el cambio de giro del Puente H.
Y1	-	Sensor magnético ubicado a la izquierda del IMS10, sus terminales se encuentran conectados en la tarjeta esclavo de comunicación PROFIBUS-DP.
Y2	-	Sensor magnético ubicado a la derecha del IMS10, sus terminales se encuentran conectados en la tarjeta esclavo de comunicación PROFIBUS-DP.
Y3	14	Cilindros paralelos.
Y4	15	Cilindro elevador.
Y5	16	Cilindro de parada.
Y6	17	Cilindro separador.
Y7	7	Sensor fin de carrera.

**Tabla 2.2: Detalle de los pines en la tarjeta Pyboard y DB- 25 del IMS10.**

El programa principal del proceso almacenar es adjuntado en Anexo 1 en el literal h, con el nombre de “Almacenar-IMS10 Manual”, y se utilizó los sensores y actuadores mencionados anteriormente. En este programa se implementó retardos de tiempo (delays), de la librería interna de Pyboard, la cual es la librería pyb. Además, hay que aclarar que este programa fue diseñado explícitamente para funcionar independientemente de cualquier otro proceso. Por lo tanto, de este programa básico se creará la función en lenguaje MicroPython para el proceso Almacenar.

## 2.2.6 Proceso liberar en el subsistema IMS10.

En este proceso de liberar pallets, se utilizó los sensores SM1, SM2, SFC y los actuadores del subsistema IMS10. Así mismo, la banda transportadora moverá los pallets una vez estos sean depositados sobre esta, luego de sacarlos de la estructura metálica del subsistema. Además, como medida de precaución se desconectó la tarjeta de esclavo de comunicación PROFIBUS-DP, ya que no será necesario al no utilizar el PLC, sino que la tarjeta Pyboard hará la función de controlador del proceso.

Para la programación del código se tendrá en consideración la metodología Grafcet, de manera que se planteó todo el proceso en etapas y condiciones de transición, el proceso es detallado a continuación en la tabla 2.3.

<b>Etapa 1</b>	No se realiza acción
<b>Condición de transición 1 a 2</b>	Estado de Marcha_Liberar en alto. En este caso este botón será USR, que se encuentra en la tarjeta Pyboard. Además, el sensor SFC debe estar activo ya que esto indica que la estación se encuentra llena de pallets.
<b>Etapa 2</b>	Se activa el cilindro paralelo (M4), el cual subirá la estructura que recibirá el pallet.
<b>Condición de transición 2 a 3</b>	Tiempo de espera de 5 segundos, hasta que los pistones del cilindro paralelo se extiendan correctamente.
<b>Etapa 3</b>	Se activa el cilindro elevador (M3), y de esta forma se estará al ras de los cilindros separadores, listo para recibir el pallet.
<b>Condición de transición 3 a 4</b>	Tiempo de espera de 4 segundos para que se pueda realizar la acción anterior.
<b>Etapa 4</b>	Se activan los cilindros separadores (), mientras los cilindros paralelos (M4) y elevador (M3) continúan activos, de esta manera el pallet caerá sobre la estructura sobre los pistones.

<b>Condición de transición 4 a 5</b>	Tiempo de espera de 2 segundos
<b>Etapa 5</b>	Se desactiva los cilindros paralelos (M4), de esta forma el pallet descenderá lentamente, el cilindro elevador (M3) continuará activo y de igual forma el cilindro separador ( ).
<b>Condición de transición 5 a 6</b>	Tiempo de espera de 3 segundos para que se pueda realizar las acciones anteriores.
<b>Etapa 6</b>	Se desactiva el cilindro separador ( M5), de esta manera no descenderán más de un pallet ya que los cilindros separadores sostendrán a los demás pallets dentro de la estación de almacenamiento.
<b>Condición de transición 6 a 7</b>	Tiempo de espera de 5 segundos para que el pistón del cilindro paralelo (M4) este contraído.
<b>Etapa 7</b>	El cilindro elevador (M3) es desactivado y de esta forma la planta regresará a su estado inicial. Y el pallet ya estará sobre la banda transportadora.
<b>Condición de transición 7 a 8</b>	Tiempo de espera de 5 segundos.
<b>Etapa 8</b>	Se mueve banda transportadora hacia la derecha a velocidad baja. X1 en alto (PWM), X2 en alto (En1) y X3 en bajo (En2).
<b>Condición de transición 8 a 9</b>	Se revisa el estado de sensor magnético SM2 si está activo pasa a la siguiente etapa.
<b>Etapa 9</b>	La banda transportadora se detiene, es decir, X1 estará desactivado.
<b>Condición de transición 9 a 10</b>	Tiempo de espera de 1 segundo.
<b>Etapa 10</b>	Se mueve banda transportadora hacia la izquierda a velocidad baja. X1 en alto (PWM), X2 en bajo (En1) y X3 en alto (En2).

<b>Condición de transición 10 a 11</b>	Sensor magnético SM1 activo.
<b>Etapa 11</b>	Incrementa en uno el contador Numero_Pallets y la banda transportadora se mueve hacia la izquierda a velocidad alta. X1 en alto (PWM), X2 en bajo (En1) y X3 en alto (En2).
<b>Condición de transición 11 a 12</b>	Tiempo de espera de 10 segundos.
<b>Etapa 12</b>	La banda transportadora se detiene, es decir, X1 estará desactivado.
<b>Condición de transición:</b> <b>- 12 a 1</b> <b>- 12 a 2</b>	Se revisa el número del contador Numero_Pallets si este es mayor que cuatro, la secuencia vuelve a la <b>Etapa 1</b> , o si es menor que cuatro este regresa a la <b>Etapa 2</b> .

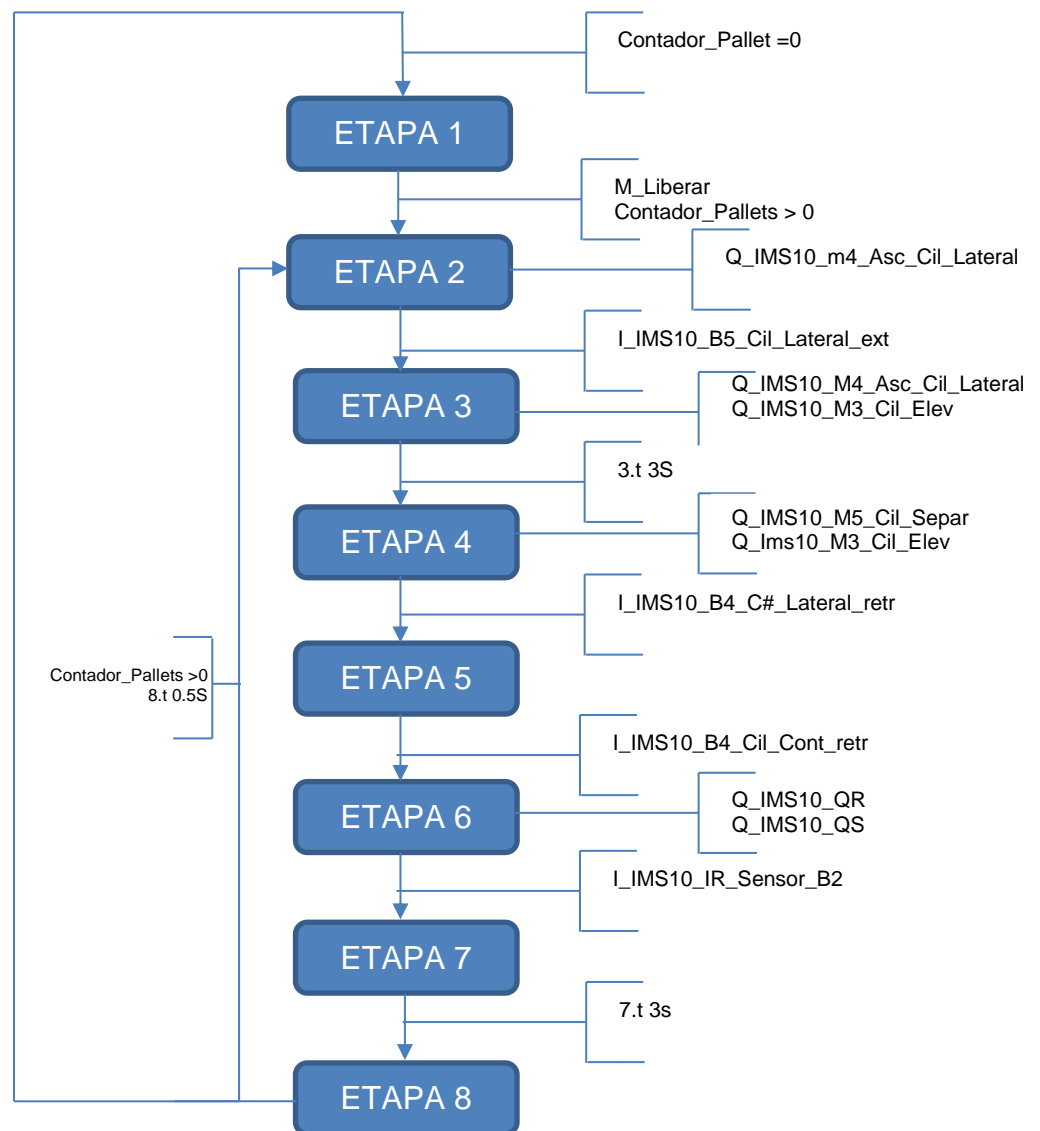
**Tabla 2.3: Grafcet del proceso liberar en el IMS10.**

Se tuvo en consideración para evitar dejar caer botellas de los sixpacks al suelo, hacer descender al pallet lentamente, para esto primero se tiene que contraer el pistón del cilindro paralelo, de esta forma el pallet caerá despacio y los sixpacks no se moverán del pallets en el cual están apoyados Por último se debe contraer el cilindro elevador ya que este pistón es de fuerza, utilizado para romper el seguro de los cilindros separadores, por lo cual es un pistón de acción rápida y este movimiento hace mover de su posición al pallet si no se mantiene el orden de secuencia indicado para bajarlo de la estación.

En este diagrama Grafcet se tiene dos lazos, en el primero se está dentro, cuando el contador Numero\_Pallets es menor que cuatro, esta condición nos indica que el IMS10 todavía posee al menos un pallet en su estación, por esta razón aún se debe seguir la secuencia del proceso liberar. El segundo lazo nos conduce al inicio del proceso a la espera

del botón Marcha\_Liberar, y se vuelve a esta etapa cuando el contador Numero\_Pallets es mayor que cuatro, el cual es el número de pallets con porta-sixpack máximo que se puede almacenar en la estación del IMS10. Este contador incrementa en uno cuando se activa el sensor SM1, en consecuencia, este contador debe inicializarse en cero al inicio del código.

El diagrama secuencial del Grafcet mostrado en la tabla 2.2, se muestra a continuación en la Figura 2.20.



**Figura 2.20: Diagrama secuencial grafcet del proceso liberar.**

El diagrama secuencial indicado en la figura 2.20 da las bases para la programación del código que cumpla lo requerido en el proceso Liberar, se le añadió muchas más etapas a las planeadas ya que el proceso debía ser controlado con mayor cuidado en la parte de los tiempos de retardos, donde se agarran los pallets que aún quedan en la estación cuando el primero cae, ya que si en tiempo es menor, las pinzas agarran la base del pallet que aún no ha descendido, otro caso sería cuando el tiempo es mayor, si esto ocurre los cilindros separadores no sujetarían los demás pallets que descienden a continuación del primero y entonces caerían dos pallets o todos a la vez sobre el cilindro elevador, de modo que si se tienen los sixpacks sobre los pallets, estos podrían llegar a romperse. Para evitar esta situación se añadió 4 etapas extras para un mayor control.

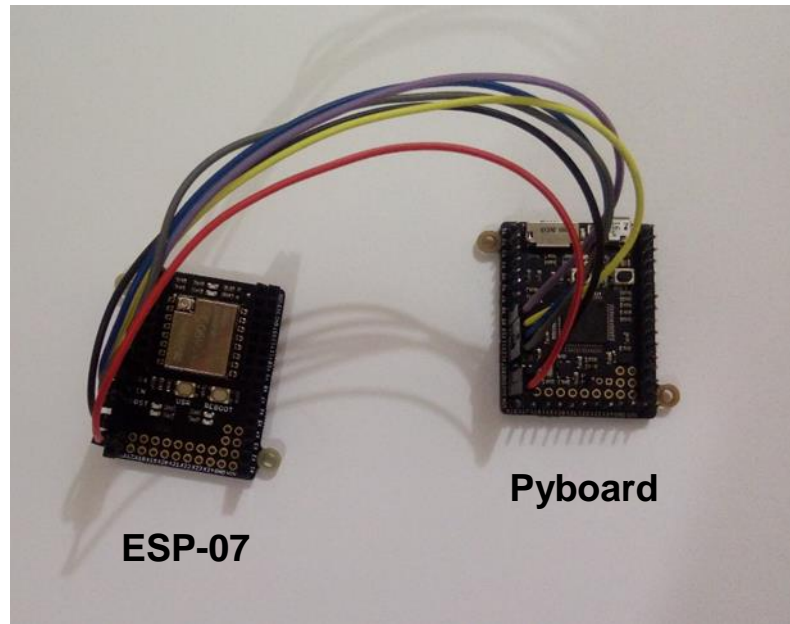
El programa principal del proceso liberar se adjunta en la parte de Anexo 1, literal i, bajo el nombre de “Liberar-IMS10-Manual”, además en este se utilizaron los sensores y actuadores señalados en la tabla 2.2, este programa fue diseñado independientemente de cualquier otro proceso, inclusive del proceso almacenar. Por lo cual de este programa se creará la función en lenguaje MicroPython para el proceso de Liberar.

## **2.3 Diseño de la página web.**

### **2.3.1 Conexión a internet de Pyboard a sitio web.**

Para realizar la conexión a internet se utilizaron los módulos Wi-Fi, ESP-07 y ESP-01, los cuales poseen un microcontrolador ESP8266, este módulo se conecta con la tarjeta Pyboard en los pines X9, X10, X11, X12, 3V3 y GND. Las conexiones con los módulos se muestran en las figuras 3.21 y 3.22. Además, X9 corresponde a Tx, X10 a Rx, X11 a En, X12 a RST, 3V3 a la alimentación y GND a la tierra del módulo.





**Figura 2.21: Conexión entre Pyboard y módulo Wi-Fi ESP-07.**

La programación de la tarjeta Pyboard para utilizar los módulos Wi-Fi se la realizó utilizando comunicación UART y ocupando a los módulos Wi-Fi como esclavos de comunicación serial, de esta forma se envían los comandos AT por Tx y por Rx de la Pyboard, se reciben los datos que responda el servidor. Inclusive, los comandos AT lo manejan los módulos por default de fábrica, por este motivo se utiliza un código llamado `pywifi.py` proporcionado por el fabricante para el envío de los comandos AT, esto sirve de interfaz de comunicación entre la Pyboard y el módulo Wi-Fi, este código pertenece al fabricante TonyLabs y se encuentra en la parte de Anexo 2 para su revisión.

A partir del código del fabricante, se realizaron pruebas para ver si el servidor responde adecuadamente, pero este no daba la respuesta deseada utilizando las herramientas proporcionadas por el archivo del fabricante, por lo que se dispuso a crear dos funciones nuevas dentro de `pywifi.py`, estas funciones son SEND y POST.

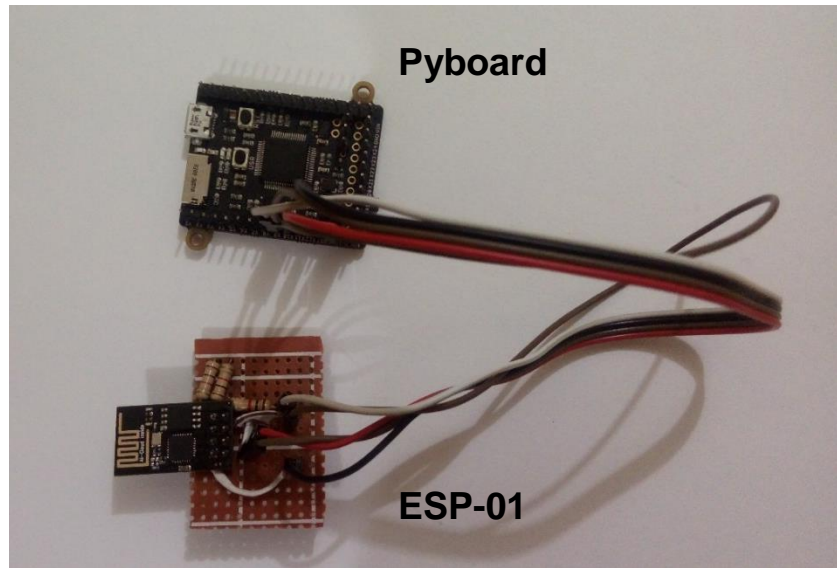


Figura 2.22: Conexión entre Pyboard y el módulo WI.FI ESP01.

### Función SEND.

Esta función realiza la recepción de datos gracias a que primero se envía un formulario utilizando un requerimiento con método GET, el formulario no es especificado dentro de la función, sino que debe ser enviado desde el esclavo serial ESP-07 al servidor por medio de Wi-Fi y el servidor debe responder con un OK al requerimiento. Luego se envía el dato que se pide y se cierra la sesión de la conexión, si no se cierra esta sesión podrían aparecer errores de “already connection” como el mostrado en la figura 3.23. Al presentarse este fallo, la tarjeta sale de la ejecución del archivo pywifi.py.

```
(71,)
['71']
37 - TX: b'AT+CIPSEND=71\r\n'
3469 - RX: b'AT+CIPSEND=71\r\n\r\n'
13231 - RX: b'\r\n'
23234 - RX: b'OK\r\n'
23768 - 'OK' received!
1033208 - RX: b'> '
('TCP', 'controlims10.co.nf', 80)
['TCP', '"controlims10.co.nf"', '80']
52 - TX: b'AT+CIPSTART="TCP","controlims10.co.nf",80\r\n'
8472 - RX: b'AT+CIPSTART="TCP","controlims10.co.nf",80\r\n\r\n'
19018 - RX: b'ALREADY CONNECTED\r\n'
28998 - RX: b'\r\n'
38995 - RX: b'ERROR\r\n'
Traceback (most recent call last):
  File "main.py", line 37, in <module>
  File "pywifi.py", line 477, in start_connection
  File "pywifi.py", line 250, in _set_command
  File "pywifi.py", line 146, in _send_command
CommandError: Command error!
MicroPython v1.9.2-87-gda8c4c26 on 2017-09-13; PYBv1.1 with STM32F405RG
Type "help()" for more information.
```

Figura 2.23: Error al presentarse Already Connect.

En la función se debe leer todas las líneas que responda el servidor, hasta llegar a la línea que contiene el dato, esta línea es almacenada como un dato tipo arreglo y luego utilizamos la función `extractJsonUART ()` que se encuentra en el archivo `json.py`, el cual se programó para extraer el dato dentro de la línea de respuesta lleno de caracteres irrelevantes y este valor es devuelto por la función `send()`.

### **Función POST.**

Esta función utiliza el método POST, con el cual se enviarán datos desde la tarjeta Pyboard hacia la página web, el servidor debe responder OK cuando la recepción de datos es exitosa, además al igual que la otra función en esta se leen las líneas que responde el servidor, hasta llegar a la línea en la que se cierra la sesión de la conexión y el módulo estará en capacidad para enviar más información. La estructura de las funciones SEND y POST son las mismas, lo que cambia es el formulario de envío y la respuesta del servidor.

Ambos códigos que se agregaron al archivo `pywifi.py`, se incluirán en Anexo 1, literal j y literal k.

### **Formularios de requerimiento a servidor.**

#### **GET:**

```
'GET /banda_auto/getParo.php HTTP/1.0\r\nHost: controlims10.co.nf\r\n\r\n',  
debug=True
```

Este formulario se envía en la función `esp.send()` para que el esclavo serial ESP-07, pueda enviar la petición al servidor y luego este responda, la respuesta de este formulario, será el valor de la variable requerida, por lo tanto este será almacenado. En este caso se extrae el valor de la variable Paro del URL “controlims10/banda\_auto/getParo.php”, el cual puede estar en 0 o 1.

### POST:

```
'POST /banda_auto/updateSensores.php HTTP/1.0\r\nHost: controlims10.co.nf\r\nAccept: */*\r\nContent-Length: 25\r\nContent-Type: application/json\r\n\r\n{"sm1":1}', debug=True
```

Este formulario es enviado en la función `esp.post()` , de esta forma el esclavo serial ESP-07 envía la petición al servidor y actualiza la variable en la base de datos y responderá con un OK si la actualización fue exitosa. En este caso se actualiza el valor del sensor magnético 1 (SM1) de 0 o 1.

### 2.3.2 Arquitectura de la página web.

La página web está desarrollada en Angular y PrimeNG por parte del front end con un servicio Restful en PHP que se comunica con una base MySQL. Para la autenticación se utilizó Firebase que permite un login con cuentas asociadas a Google. La página web y su servicio restful se encuentra alojado en un servidor gratuito BIZ.NF.

Angular ofrece modularidad, ya que con la ayuda de crear módulos por cada componente que puede ser reutilizado en el código, esto nos facilita el desarrollo estructural de la página web, además facilita las relaciones entre servicios, permitiéndonos consumir el servicio Restful por parte de nuestro backed. Para el diseño de la página se utilizó la librería de PrimeNG que ofrece componentes prediseñados.



Figura 2.24: Herramienta utilizadas para la creación de la página web.

## CAPÍTULO 3

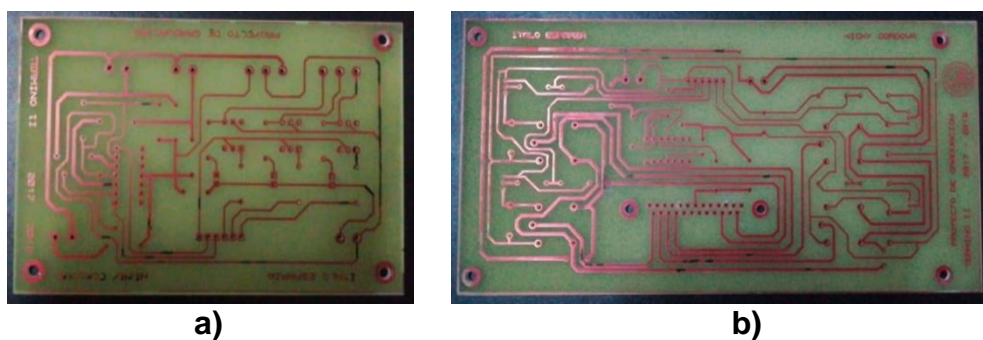
### 3. RESULTADOS.

En el presente capítulo se presentarán detalladamente los resultados que han sido obtenidos durante el desarrollo del proyecto. Además, se lo ha dividido en secciones, que engloba todos los puntos planteados en el alcance.

#### 3.1 Hardware.

En el apartado del Hardware se observaron los siguientes resultados, que englobaron tanto, mediciones de señales por parte de los actuadores y sensores, material para la elaboración de las placas electrónicas, etapa de potencia, acondicionamiento de señales, protecciones, así como, precio de los materiales.

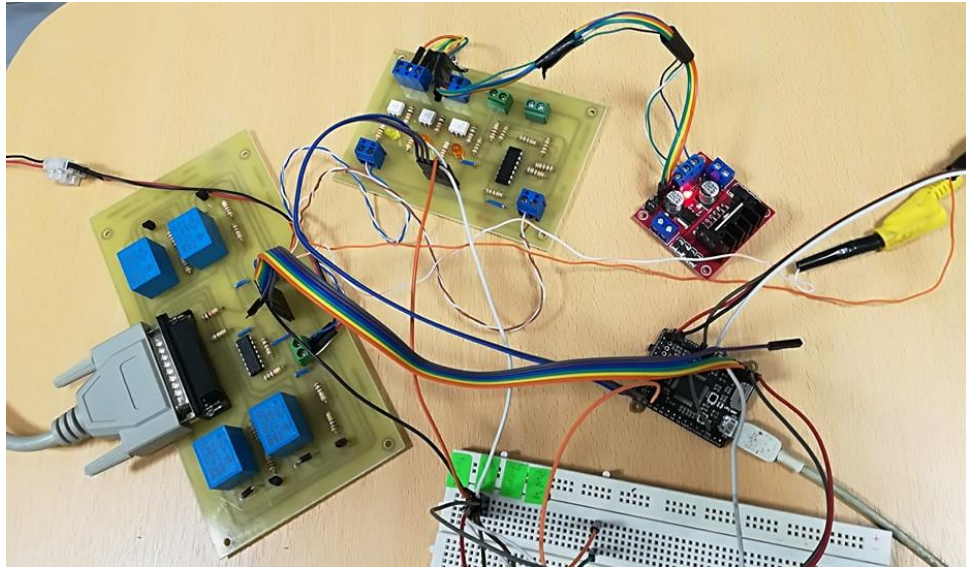
La corriente que demandaron todas las placas electrónicas en el proyecto fue de aproximadamente 1.5 amperios, las cuales fueron proporcionadas por un cargador que sirvió como fuente y cuyo voltaje fue de 5 VDC; adicionalmente, se observó un ligero calentamiento en el cargador al estar mucho tiempo en uso, debido a que actúa como fuente de alimentación para todos los circuitos del proyecto



**Figura 3.1: Circuitos impresos a) Tarjeta que maneja banda transportadora b) Tarjeta que controla el subsistema IMS10.**

El material que se escogió para la elaboración de las placas electrónicas, debido a su larga duración, resistencia a altas temperatura y relación calidad-precio, fue de fibra de vidrio. Además, las impresiones de las dos tarjetas en este material cuyas dimensiones fueron de 6,75 x 10,5 cm en la tarjeta que

maneja la banda transportadora, y 9 x 18,5 cm en la tarjeta que controla el subsistema IMS10, originó un costo de 41,35 dólares.



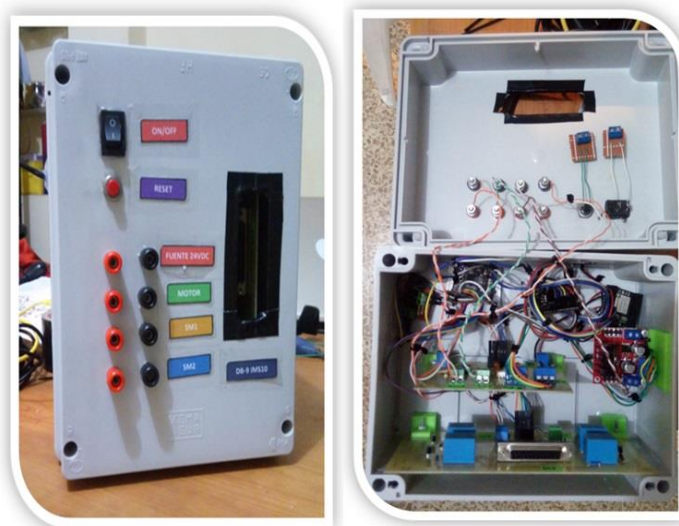
**Figura 3.2: Acabado final, conexiones entre tarjetas y módulo.**

Los voltajes de salida del circuito de acondicionamiento de señales, entre los sensores magnéticos y la Pyboard, fue entre 3,25 a 0 VDC valores aproximados que son tolerados por la tarjeta, además, su corriente de salida no fue de mucha importancia solo bastaba que sea pequeña. Sin embargo, el sensor de fin de carrera sí manejaba una corriente aproximada a 1 amperio, aunque no se encontró inconveniente alguno para este caso ya que se utilizó un resistor con una resistencia de 100 kilo ohmios para mitigar la corriente. Por otro lado, el circuito se lo implementó con 4 relés, 2 Opamp, 16 resistencias, 4 transistores BJT y 4 diodos para toda la etapa de acondicionamiento de señales.

Los niveles de voltajes lógicos que llegaban al driver LM298, para cambiar el sentido de giro del motor, estaban comprendidos entre los 0 a 4,8 VDC, así mismo la corriente y voltaje de salida que demandaba el motor del módulo fue de 1,5 amperios y 24 VDC, cumpliendo con el rango de características recomendados por el fabricante. Cabe aclarar que el driver en un corto periodo de uso no presentó un sobrecalentamiento considerable gracias a su disipador, por el contrario, al utilizarlo por un tiempo prolongado el integrado se calentaba, originando un riesgo en la

tarjeta. Por esta razón las dimensiones del disipador de calor deberían de ser mayores.

La frecuencia de operación óptima del motor de la banda transportadora oscilaba entre los 2500 a 4000 Hz. A frecuencias inferiores de 2500 Hz el motor no se movía, además emitía un ruido molesto. Así mismo, a frecuencias superiores de 4000 Hz la velocidad del motor no variaba, aun cambiando su ancho de pulso. Por otro lado, el rango del ancho de pulso estuvo comprendido entre los 40 a 100 % por seguridad. Dentro de los parámetros indicados, el motor funcionaba correctamente, sin falla alguna y con un movimiento suave y sin saltos; además no presentaba ningún sonido que sugiriese algún mal funcionamiento.



**Figura 3.3: Sistema embebido en su caja de protección.**

El tiempo de recepción de señales desde los sensores magnéticos y el fin de carrera hacia la Pyboard fue inmediato y no se encontraron errores de falsas activaciones, al utilizar la configuración PULL\_UP, en los pines de entrada a la tarjeta, de los mencionados sensores. Sin embargo, cuando se utilizó la configuración PULL\_DOWN en los pines de entrada, cuando el motor exigía corriente de 1,5 amperios, el ruido magnético ocasionaba que se detecte los sensores como si estuvieran en alto.









También, el tiempo de almacenado para cada pallet fue aproximadamente de 30 segundos, que inicia a partir que el primer pallet pasa y activa el

primer sensor magnético, por otro lado, el tiempo de liberar para cada pallet fue aproximadamente de 25 segundos, que inicia cuando se da la orden de liberar desde la página web, por último, todo el proceso completo, tuvo tiempo estimado de 4 minutos, siendo estos valores los óptimos y previenen posibles errores en el desarrollo del proyecto.

El costo de inversión de las etapas de control y de fuerza, bordearon los 100 dólares, un ahorro significativo en comparación a otros métodos de automatización.

### 3.2 Software.

Con respecto al software se obtuvieron los siguientes resultados, donde se analizarán, el tamaño de la programación en ambos controladores, tiempos de envío y recepción de señales, tiempo y frecuencias de ejecución de funciones.

Nombre	Tipo	Tamaño	Nombre	Tipo	Tamaño
 funciones	Archivo PY	12 KB	 funciones	Archivo PY	2 KB
 json	Archivo PY	1 KB	 json	Archivo PY	1 KB
 main	Archivo PY	2 KB	 main	Archivo PY	2 KB
 pywifi	Archivo PY	22 KB	 pywifi	Archivo PY	22 KB

a)

b)

**Figura 3.4: a) Archivos del controlador 1, b) Archivos del controlador 2.**

El tamaño total de la programación que contiene la primera tarjeta es aproximadamente de 47 kb, así mismo la segunda tarjeta que posee 4 archivos tiene un tamaño de 27 kb.

El tiempo aproximado que le toma la función conexión\_wifi para conectarse al internet es de 4 segundos, suponiendo una conexión a la red estable. Esta función es utilizada en ambas tarjetas, hay que destacar que los módulos WI-Fi se deben resetear, cuando se aplique el paro por parte de la página web, debido a que si no se resetean al seguir con la alimentación de la tarjeta Pyboard seguirá conectada a la red Wi-Fi anterior y por la programación realizada esta presenta errores y se quedó sin tener respuesta del servidor, caso contrario será si se resetea el módulo, en esta



misma función conexión\_wifi, se manda a bajo lógico el pin X12 ( pin Reset del módulo Wi-Fi), y luego de un tiempo de retardo a alto lógico para empezar el proceso de conexión a la red.

La velocidad de transmisión de datos que se configuró a la función pywifi.ESP8266 fue de 115200 baudios, con lo cual el módulo wifi se pudo conectar a la red y al utilizar esta velocidad de transmisión se pueden tener mejores resultados y tener más tiempo de reacción de parte del módulo Wi-Fi, esto significa menos tiempo de espera para actualizar los sensores o menos tiempo de espera para recibir el estado de botón de paro o marcha desde la página web, optimizando el tiempo de ejecución del código.

La frecuencia de interrupción del timer 10 para que constantemente este revisando el Paro fue de 8 Hz, es decir cada 0,125 segundos; este tiempo fue analizado y probado para no tener problemas de ejecución de ambos procesos, ya que al no poder usar otros métodos de multi procesamiento, se recurrió al uso de dos tarjetas de control.

Cada cierto tiempo el paro se activaba y por este motivo las tarjetas se reseteaban, originando un mal funcionamiento de las tarjetas de control, esto se debía a que el nivel lógico del pin Y4 de la tarjeta 2, se mostraba diferente al pin X4, esto no era producido por error de programación, sino por una falta de equivalencia en los niveles de voltaje, se resuelve con la conexión de los pines 3V3 y GND, de ambas tarjetas.

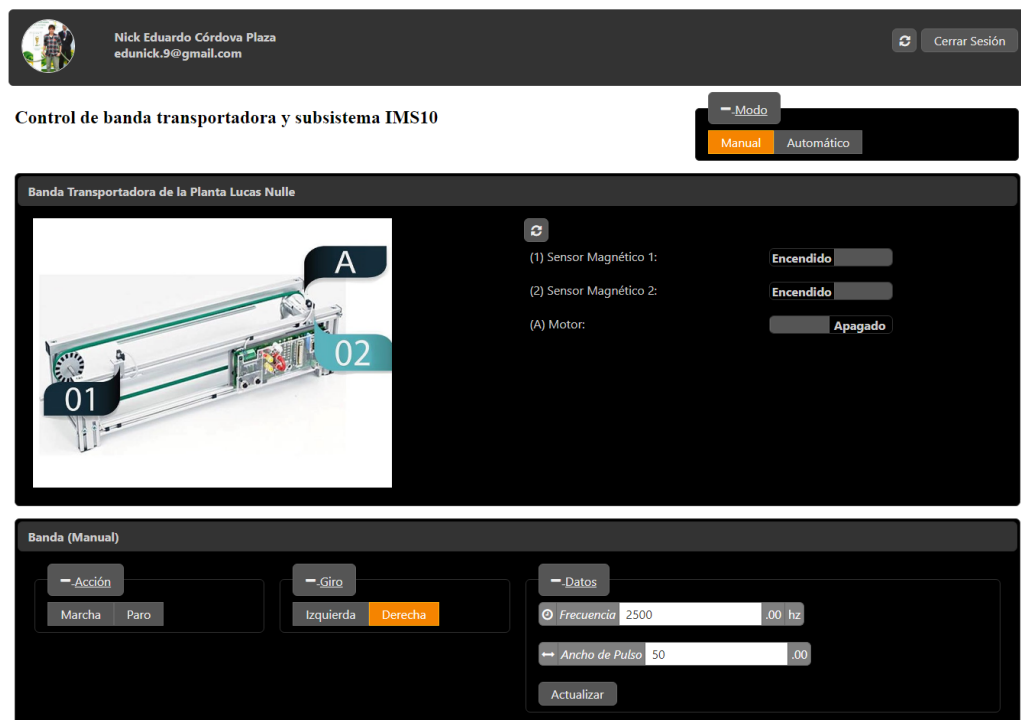
### **3.3 Página web.**

Tomando en cuenta que se compartieron datos desde un celular, para que los módulos Wi-Fi se conectaran a la red, y abarcando temas como la seguridad de la página web, análisis de los scadas tanto en modo manual y automático y tiempo de activación de los indicadores en la página web.



**Figura 3.5: Página de inicio de la página web controlims10.**

En la imagen anterior, se muestra la única medida de seguridad que se implementó en la página web fue el uso de una cuenta Gmail previamente registrados en el servidor para que este autorice el ingreso de un usuario al momento de acceder al sitio web. Los usuarios que pueden acceder a la página por el momento son los dos integrantes de este proyecto, esto se realizó como medida de protección y controlar el uso del proceso, este servicio es utilizado por la aplicación Firebase, la cual es una aplicación confiable y ya utilizada en la comunidad.

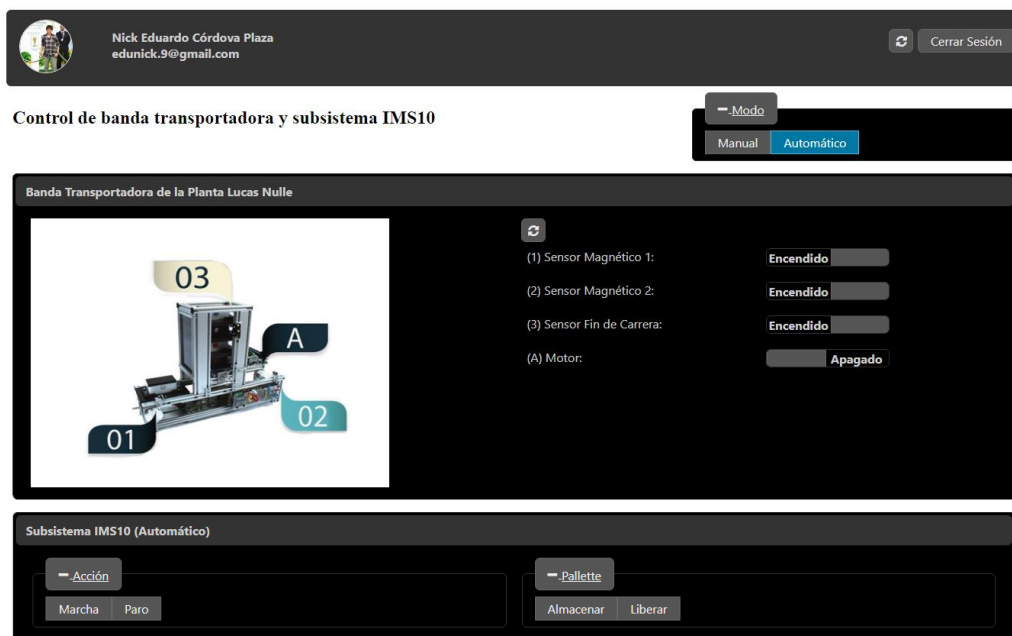


**Figura 3.6: Página web del modo manual.**

Como se puede observar en la imagen, el SCADA en modo manual muestra indicadores de estado de los sensores magnéticos SM1 y SM2, así como el del motor. Además de un tablero de control, donde se puede controlar la

velocidad, por medio de su frecuencia o ancho de pulso, y sentido de giro del motor.

En el modo manual, el tiempo de envío de señales desde los sensores hacia la página web comprendió entre 1 a 3 segundos, además el tiempo que tomó en enviar las ordenes ya sea izquierda, derecha, marcha o paro, hacia la Pyboard tomo un tiempo estimado de 2 segundos. Estos tiempos son dependientes de la velocidad de la red Wi-Fi, entre más estable y con mayor alcance, había mejores resultados en el cambio de estado de sensores y acciones de marcha, paro, cambio de giro y velocidad del motor.



**Figura 3.7: Página web del modo automático.**

Como se puede observar en la figura, el SCADA en modo automático, presenta casi los mismos indicadores de estados lógicos que en modo manual, solo que en esta sección se añadió un indicador para el sensor de fin de carrera. Además, en la parte inferior se encuentra el tablero de control, donde permite controlar el subsistema IMS10, para que realice la etapa de almacenado o de liberado, así como botoneras para dar marcha el proceso o detenerlo.

En el modo automático, el tiempo de envío de señales desde los sensores hacia la página web comprendió entre 1 a 3 segundos, además el tiempo que tomó en enviar las órdenes de liberar y almacenar hacia la Pyboard fue

de 1 segundo. En este caso, entre menos datos se envíen utilizando las funciones send y post, menor será el tiempo en que la tarjeta controle el proceso. Esto es más notorio en la toma de decisiones por parte del usuario, como lo es el paro, marcha, entre otros.

El número de interacciones permitidos en el servidor fue 15000 por hora, al ser un servidor gratuito creemos que esta es una cantidad aceptable de peticiones realizadas. Por otro lado, para poder utilizar un servidor por un tiempo ilimitado en un proceso que lo demande, este costo sería de 36 dólares por 6 meses, un valor aceptable por el tiempo de uso que nos proporciona. Solo 2 veces desde el inicio del proyecto se presentó este inconveniente, durante las pruebas realizadas de todo el proyecto. También se redujo la cantidad de peticiones al servidor para evitar exceder el límite estipulado por hora de requerimientos.

## CONCLUSIONES Y RECOMENDACIONES

En conclusión, el diseño de una adecuada etapa de acondicionamiento de señales, junto con la programación para la interacción de actuadores y sensores, permitieron el control del subsistema IMS10 con la Pyboard. Además, la solución de implementar amplificadores operacionales específicamente el LM324, para reducir las señales de voltaje de 24 a 5 VDC, provenientes de los sensores magnéticos y el fin de carrera, resulto ser viable, tanto en el aspecto económico, como en lo práctico. Como única observación, dentro de la programación fue necesario añadir una configuración PULL UP al momento de declarar los pines de entrada de los sensores, debido a voltajes parásitos que se producían en los pines de entrada.

Los valores de frecuencias y ancho de pulso (ciclo de trabajo) del motor oscilaron entre 2500 a 40000 Hz y 40 a 100% respectivamente, así mismo, para detener la banda sólo fue necesario un 40% del ancho de pulso, dando como resultado un desenvolvimiento idóneo del motor de la banda transportadora, cabe recalcar que, al hacerlo trabajar a frecuencias menores de 2500 Hz, paulatinamente se iba generando una inestabilidad en su velocidad, así como, la emisión de sonidos agudos producido por las bajas frecuencias.

A lo largo del desarrollo del proyecto se logró controlar eficientemente el subsistema IMS10 y la banda transportadora, donde fue necesario el uso de dos controladores, debido a que, el procesador de la Pyboard no era capaz de realizar varios procesos a la vez, es decir, carecía de la funcionalidad de multihilos, por esta razón se procedió a utilizar dos controladores, así mismo, cada uno desenvolvía funciones específicas, por un lado una tarjeta se encargaba de recibir las señales de control provenientes de la página web para controlar el subsistema IMS10, y por otro lado la segunda tarjeta se encargaba de recibir las señales de los sensores para enviarlas a la página web e incluso la señal de paro que reseteaba ambas tarjetas, esto originó poder recibir los estados de los sensores en tiempo real y la ejecución del paro para que resetee ambos controladores en cualquier momento, sin importar en que parte del proceso se encuentre.

Finalmente, este tipo de control mediante el uso de la Pyboard y sus periféricos, presentan una alternativa de control y automatización para procesos industriales,

siendo más baratos, pues, el hardware se lo puede diseñar con materiales disponibles en el mercado ecuatoriano, también es de código abierto originado un ahorro significativo por la no compra de licencia, y flexibles al momento de encontrar información en la gran comunidad que posee Python, solucionando un problema económico para aquellas pequeñas empresas que desean invertir en automatizar sus procesos.

Se recomienda la implementación de un servidor local para la interacción entre la página web y la tarjeta Pyboard, par así, mejorar la conectividad y velocidad, en la transmisión de la información.

Se recomienda encriptar los datos que son enviados a la página web, para de tal manera, desarrollar un SCADA que presente los datos a tiempo real, así como, proteger de mejor manera la información ante posibles eventos de hackeo o manipulación.

## BIBLIOGRAFÍA

Lucas-Nülle GmbH. (2016). IMS 10 Almacenamiento intermedio [Online]. Disponible en: <https://www.lucas-nuelle.es/2273/apg/1491/IMS-10-Almacenamiento-intermedio.htm>

Lucas-Nülle GmbH. (2016). Estación de almacenamiento intermedio de material [Online]. Disponible en: <https://www.lucas-nuelle.es/2274/pid/2244/apg/1492/Estacioacut;n-de-almacenamiento-intermedio-de-material---.htm>

Lucas-Nülle GmbH. (2016). Segmento de cinta transportadora doble de 24V [Online]. Disponible en: <https://www.lucas-nuelle.es/2274/pid/2159/apg/1457/Segmento-de-cinta-transportadora-doble-de-24V---.htm>

ELECTRONILAB. (2017). Driver dual para motores (Full-Bridge) – L298N [Online]. Disponible en: <https://electronilab.co/tienda/driver-dual-para-motores-full-bridge-l298n/>

VISHAY INTERTECHNOLOGY, “Description, Features, Applications” en Optocoupler, Phototransistor Output, with Base Connection, Document Number: 83725, Rev. 1.8, 07-Jan-10, pp. 1-2.

Texas Instruments, “Description, Features, Applications” en LMx24-N, LM2902-N Low Power, Quad-Operational Amplifiers, Document Number: SNOSC16D, Rev. January 2015, pp. 1-3.

RS Componets Ltd. (2017). SPDT-NO/NC Roller Lever Microswitch, 15 A @ 250 V ac [Online]. Disponible en: <http://uk.rs-online.com/web/p/microswitches/0517781/>

MicroPython.org (2017, Noviembre 9). Pyboard (v1.9.3) [Online]. Disponible en: [https://store.micropython.org/#/products/PYBv1\\_1](https://store.micropython.org/#/products/PYBv1_1)

MicroPython.org (2017, Noviembre 9). Quick reference for the Pyboard (v1.9.3.) [Online]. Disponible en: <http://docs.micropython.org/en/latest/pyboard/pyboard/quickref.html#quick-reference-for-the-pyboard>

D. Norris, “Interface, Files, and Libraries” en Python for Microcontrollers Getting Started with MicroPython, 1er ed. United Stated, McGraw – Hill Education, 2016, pp. 96 – 97

MicroPython.org (2017, Noviembre 13). General information about the pyboard (v1.9.3) [Online]. Disponible en: <http://docs.micropython.org/en/latest/pyboard/pyboard/general.html>

Adafruit.com (2017, Noviembre 12). MicroPython Basics: What is MicroPython? (1er ed). [Online]. Disponible en: <https://cdn-learn.adafruit.com/downloads/pdf/micropython-basics-what-is-micropython.pdf>

D. George, P. Sokolovsky, and contributors. (2017, Septiembre 11). MicroPython Documentation (v1.9.2) [Online]. Disponible en: <https://media.readthedocs.org/pdf/micropython/latest/micropython.pdf>

MicroPython.org (2017, Noviembre 13). Getting started with MicroPython on the ESP8266 (v1.9.3) [Online]. Disponible en: <https://docs.micropython.org/en/latest/esp8266/esp8266/tutorial/intro.html#requirements>

ESP-07S User Manual, rev:1.0. Shenzhen Ai-Thinker Technology Co. Ltd, 2016



# ANEXOS

## Anexo 1: Pruebas Programación Pyboard.

### a) Programa intensidad de leds.

```
# Programa intensidad de leds
import pyb
ledprueba = pyb.LED(4)
sw = pyb.Switch()
x = 0

while True:

    if sw():
        x = min(255, x+ 1)
    else:
        x = max(0, x- 1)
        ledprueba.intensity(x)
        pyb.delay(30)
```

### b) Regulación de velocidad por PWM.

```
from pyb import Pin, Timer
import pyb
sw=pyb.Switch()
p1 = Pin('X1')           # X1 has TIM2, CH1
tim = Timer(2, freq=10)
ch1 = tim.channel(1, Timer.PWM, pin=p1)

p2 = Pin('X2')           # X2 has TIM2, CH2
tim = Timer(2, freq=10)
ch2 = tim.channel(2, Timer.PWM, pin=p2)
while True:
    if sw():
        ch1.pulse_width_percent(50)
        ch2.pulse_width_percent(0)
    else:
        ch1.pulse_width_percent(0)
        ch2.pulse_width_percent(50)
```

**c) Control banda IPA5.**

```
import machine
from pyb import Pin, Timer
import pyb

p2 = machine.Pin('X2', machine.Pin.OUT)
p3 = machine.Pin('X3', machine.Pin.OUT)
p1 = Pin('X1') # X1 has TIM2, CH1
led1 = pyb.LED(4)
tim = Timer(2, freq=2500)
ch1 = tim.channel(1, Timer.PWM, pin=p1)
sw = pyb.Switch()
# Programa Principal
while True:

    if sw():
        # Movimiento contrario al inicial
        p2.high()
        p3.low()
    else:
        # Movimiento inicial
        led1.on()
        p2.low()
        p3.high()
        ch1.pulse_width_percent(50)
```

**d) Prueba sensores.**

```
import machine
from pyb import Pin
import pyb

ledprueba1 = pyb.LED(3)
ledprueba2 = pyb.LED(4)
y1 = machine.Pin('Y1', machine.Pin.IN, machine.Pin.PULL_DOWN) #
Sensor1
```

```
y2 = machine.Pin('Y2', machine.Pin.IN, machine.Pin.PULL_DOWN) #
Sensor2
```

```
while True:
    if y1.value():
        ledprueba1.on()

    else:
        ledprueba1.off()

    if y2.value():
        ledprueba2.on()

    else:
        ledprueba2.off()
```

#### e) Control-secuencial Banda con Sensores.

```
import machine
from pyb import Pin, Timer
import pyb

y1 = machine.Pin('Y1', machine.Pin.IN, machine.Pin.PULL_DOWN) #
Sensor1
y2 = machine.Pin('Y2', machine.Pin.IN, machine.Pin.PULL_DOWN) #
Sensor2
p2 = machine.Pin('X2', machine.Pin.OUT)
p3 = machine.Pin('X3', machine.Pin.OUT)
p1 = Pin('X1') # X1 has TIM2, CH1
led1 = pyb.LED(4)
tim = Timer(2, freq=2500)
ch1 = tim.channel(1, Timer.PWM, pin=p1)

# Programa Principal
while True:

    # Inicializamos Variables
    bandera = 1
    bandera1 = 0
    bandera2 = 0
    pyb.LED(1).off()
    pyb.LED(2).off()

    if y1.value(): # Detecta sensor1
        bandera1 = 1
```

```

else:
    bandera1 = 0

if bandera1 == 1:

    while (bandera == 1):

        led1.off()          # LED Azul se apaga

        if y2.value():      # Detecta Sensor2
            bandera2 = 1
        else:
            bandera2 = 0

        if (bandera2 == 1):
            # Movimiento contrario al inicial
            p2.high()
            p3.low()
            pyb.delay(15000)
            bandera = 0
            bandera1 = 0
            bandera2 = 0
        else:
            pyb.LED(1).on() # LED Verde se enciende
            led1.off()
            p2.low()
            p3.high()
            ch1.pulse_width_percent(45) # Disminuye Velocidad

    else:
        # Movimiento inicial
        led1.on()
        p2.low()
        p3.high()
        ch1.pulse_width_percent(55) # Velocidad maxima

```

#### f) Control secuencial anidado.

```

import machine
from pyb import Pin, Timer
import pyb

y1 = machine.Pin('Y1', machine.Pin.IN, machine.Pin.PULL_DOWN) #
Sensor1
y2 = machine.Pin('Y2', machine.Pin.IN, machine.Pin.PULL_DOWN) #
Sensor2
p2 = machine.Pin('X2', machine.Pin.OUT)
p3 = machine.Pin('X3', machine.Pin.OUT)
p1 = Pin('X1') # X1 has TIM2, CH1

```

```

led1 = pyb.LED(4)
bandera = 1
bandera1 = 0
bandera2 = 0
tim = Timer(2, freq=2500) #Encontrar una frecuencia adecuada para que no
suene
ch1 = tim.channel(1, Timer.PWM, pin=p1)

# Primer Loop
while True:
    pyb.LED(1).off()
    pyb.LED(2).off()
    # Movimiento inicial
    led1.on()          #LED azul
    p2.low()
    p3.high()
    ch1.pulse_width_percent(55) # Velocidad maxima

    if y1.value():    # Detecta Sensor1
        bandera1 = 1

        if bandera1 == 1:

            while (bandera == 1):
                pyb.LED(1).on() #LED verde
                led1.off()
                ch1.pulse_width_percent(50) # Disminuye Velocidad

            if y2.value():    # Detecta Sensor2
                pyb.LED(2).on()
                bandera2 = 1

            if (bandera2 == 1):
                # Movimiento contrario al inicial
                p2.high()
                p3.low()
                pyb.delay(15000) #Delay de 15 segundos
                bandera = 0
                bandera1 = 0
                bandera2 = 0

```

**g) Prueba actuadores IMS10.**

```

import machine
import pyb

sw = pyb.Switch()
y3 = machine.Pin('Y3', machine.Pin.OUT) # Actuador1

```

```
y4 = machine.Pin('Y4', machine.Pin.OUT) # Actuador2
```

```
while True:
```

```
    bandera = 0
```

```
    y3.low()
```

```
    y4.low()
```

```
    if sw():
```

```
        y3.high()
```

```
        bandera = 1
```

```
    if bandera == 1 and sw():
```

```
        y4.high()
```

#### **h) Almacener-IMS10 Manual.**

```
# Programa Principal para ALMACENADO DE PALLETS
```

```
while True:
```

```
    # Inicializamos
```

```
    bandera2 = 1
```

```
    p2.low()
```

```
    p3.low()
```

```
    y3.low()
```

```
    y4.low()
```

```
    y5.low()
```

```
    if sw(): # Marcha
```

```
        bandera1 = 1
```

```
    while (bandera1 == 1 and bandera2 == 1):
```

```
        # Se pregunta por el Sensor_Almacen_Lleno
```

```
        if y7.value():
```

```
            bandera2 = 0
```

```
            break
```

```
        else:
```

```
            bandera2 = 1
```

```
        # Movimiento de la banda transportadora
```

```
        while (not y1.value()):
```

```
            ch1.pulse_width_percent(60)
```

```
            p2.low()
```

```

    p3.high()

# Detecto el sensor magnetico B1
# Disminuye la velocidad y sigue avanzando
# Ademas se activa el cilindro de parada
ch1.pulse_width_percent(50)
p2.low()
p3.high()
y5.high()
pyb.delay(10000)

# Se detiene la banda
p2.low()
p3.low()
pyb.delay(4000)

# Se activa el Cilindro paralelo y el de elevacion
# Para subir el pallet y almacenarlo
y5.low() # Se desactiva el Cilindro de parada
y3.high() # Se activa el Cilindro paralelo
pyb.delay(4000)
y4.high() # Se activa el Cilindro elevador
pyb.delay(6000)
y3.low() # Se desactiva Cilindro paralelo
pyb.delay(4000)
y4.low() # Se desactiva Cilindro elevador
pyb.delay(10000)

```

**i) Liberar-IMS10-Manual.**

```

# Programa Principal para LIBERAR PALLETS
while True:

```

```

    # Inicializamos
    p2.low()
    p3.low()
    y3.low()
    y4.low()
    y5.low()

    if y7.value():
        bandera4 = 1
    else:
        bandera4 = 0

    if sw(): # Marcha
        bandera1 = 1
        bandera2 = 1
        cnt = 0

```

```

1): while (bandera1 == 1 and cnt < 4 and bandera2 == 1 and bandera4 ==
1):
    bandera3 = 1
    # Se activan los cilindros paralelos y el elevador
    # Para poder agarrar el pallet
    pyb.delay(2000)
    y3.high()
    pyb.delay(4000)
    y4.high()
    pyb.delay(8000)

    # Se abre el cilindro separador
    # De esta forma caera el pallet
    y6.high()
    pyb.delay(5000)
    y3.low()
    pyb.delay(4000)
    y6.low()
    pyb.delay(2000)
    y4.low()
    pyb.delay(15000)

    # Movimiento de la banda transportadora
    ch1.pulse_width_percent(60)
    p2.high()
    p3.low()

    while bandera3 == 1:
        if y1.value():
            cnt = cnt + 1
            bandera3 = 0

    pyb.delay(10000)
    p2.low()
p3.low()

```

#### j) Función SEND.

```

def send(self, data, debug=False):
    """Send data over the current connection."""
    self._set_command(CMDS_IP['SEND'], len(data), debug=debug)
    #print(b'>' + data)
    self.uart.write(data)
    self.uart.readline()
    self.uart.readline() # Bytes received
    self.uart.readline() # /n
    self.uart.readline() # Send OK
    self.uart.readline() # /n

```



```

self.uart.readline() # IPD, HTTP, response Request
self.uart.readline() # Date
self.uart.readline() # Server
self.uart.readline() # Content length
self.uart.readline() # Connection
self.uart.readline() # Content-type
self.uart.readline() # /n
result = self.uart.readline() # json - output
return json.extractJsonUART(str(result))

```

### k) Función POST.

```

def post(self, data, debug=False):
    """Send data over the current connection."""
    self._set_command(CMDS_IP['SEND'], len(data), debug=debug)
    #print(b'>' + data)
    self.uart.readline()
    self.uart.write(data)
    self.uart.readline()
    self.uart.readline() # Bytes received
    self.uart.readline() # /n
    self.uart.readline() # Send OK
    self.uart.readline() # /n
    self.uart.readline() # IPD, HTTP, response Request
    self.uart.readline() # Date
    self.uart.readline() # Server
    self.uart.readline() # Content length
    self.uart.readline() # Connection
    self.uart.readline() # Content-type
    self.uart.readline() # /n
    self.uart.readline()
    result = self.uart.readline() # json - output
    print(result)
    return json.extractJsonUART(str(result))

```

### l) Json.py.

```

def extractJsonUART(val):
    txt_json = ""
    str_ant = ""
    for c in val:
        str_ant = str_ant + c
        if (c == '{'):
            str_cont = val.replace(str_ant, "")
            for t in str_cont:
                if (t == '}'):
                    break
            txt_json = txt_json + t
    return txt_json

```

## Anexo 2: Librería UART-Comandos AT para ESP8266

### a) Librería Pywifi- TonyLabs.

```
from pyb import UART
from pyb import delay
from pyb import micros, elapsed_micros

# This hashmap collects all generic AT commands
CMDS_GENERIC = {
    'TEST_AT': b'AT',
    'RESET': b'AT+RST',
    'VERSION_INFO': b'AT+GMR',
    'DEEP_SLEEP': b'AT+GSLP',
    'ECHO': b'ATE',
    'FACTORY_RESET': b'AT+RESTORE',
    'UART_CONFIG': b'AT+UART'
}

# All WIFI related AT commands
CMDS_WIFI = {
    'MODE' : b'AT+CWMODE',
    'CONNECT': b'AT+CWJAP',
    'LIST_APS': b'AT+CWLAP',
    'DISCONNECT': b'AT+CWQAP',
    'AP_SET_PARAMS': b'AT+CWSAP',
    'AP_LIST_STATIONS': b'AT+CWLIF',
    'DHCP_CONFIG': b'AT+CWDHCP',
    'SET_AUTOCONNECT': b'AT+CWAUTOCONN',
    'SET_STATION_MAC': b'AT+CIPSTAMAC',
    'SET_AP_MAC': b'AT+CIPAPMAC',
    'SET_STATION_IP': b'AT+CIPSTA',
    'SET_AP_IP': b'AT+CIPAP',
    'START_SMART': b'AT+CWSMARTSTART'
}

# IP networking related AT commands
CMDS_IP = {
    'STATUS': b'AT+CIPSTATUS',
    'START': b'AT+CIPSTART',
    'SEND': b'AT+CIPSEND',
    'CLOSE': b'AT+CIPCLOSE',
    'GET_LOCAL_IP': b'AT+CIFSR', #Get ESP8266 IP address from the router
connected
    'SET_MUX_MODE': b'AT+CIPMUX',
    'CONFIG_SERVER': b'AT+CIPSERVER',
    'SET_TX_MODE': b'AT+CIPMODE',
    'SET_TCP_SERVER_TIMEOUT': b'AT+CIPSTO',
    'UPGRADE': b'AT+CIUPDATE',
    'PING': b'AT+PING'
}
```

```

# WIFI network modes the ESP8266 knows to handle
WIFI_MODES = {
    1: 'Station',
    2: 'Access Point',
    3: 'Access Point + Station',
}
# Reverse feed lookup table
for key in WIFI_MODES.keys():
    WIFI_MODES[WIFI_MODES[key]] = key

# WIFI network security protocols known to the ESP8266 module
WIFI_ENCRYPTION_PROTOCOLS = {
    0: 'OPEN',
    1: 'WEP',
    2: 'WPA_PSK',
    3: 'WPA2_PSK',
    4: 'WPA_WPA2_PSK'
}
# Reverse feed lookup table
for key in WIFI_ENCRYPTION_PROTOCOLS.keys():

WIFI_ENCRYPTION_PROTOCOLS[WIFI_ENCRYPTION_PROTOCOLS[key
]] = key

class CommandError(Exception):
    pass

class CommandFailure(Exception):
    pass

class UnknownWIFIModeError(Exception):
    pass

class ESP8266(object):

    def __init__(self, uart=1, baud_rate=115200):
        """Initialize this module. uart may be an integer or an instance
        of pyb.UART. baud_rate can be used to set the Baud rate for the
        serial communication."""
        if uart:
            if type(uart) is int:
                self.uart = UART(uart, baud_rate)
            elif type(uart) is UART:
                self.uart = uart
            else:
                raise Exception("Argument 'uart' must be an integer or pyb.UART
object!")
        else:
            raise Exception("Argument uart must not be 'None'!")

```

```

def _send_command(self, cmd, timeout=0, debug=False):
    """Send a command to the ESP8266 module over UART and return the
    output.
    After sending the command there is a 1 second timeout while
    waiting for an answer on UART. For long running commands (like AP
    scans) there is an additional 3 seconds grace period to return
    results over UART.
    Raises an CommandError if an error occurs and an CommandFailure
    if a command fails to execute."""
    if debug:
        start = micros()
        cmd_output = []
        okay = False
        if cmd == " or cmd == b":
            raise CommandError("Unknown command " + cmd + "!")
        # AT commands must be finalized with an '\r\n'
        cmd += '\r\n'
        if debug:
            print("%8i - TX: %s" % (elapsed_micros(start), str(cmd)))
        self.uart.write(cmd)
        # wait at maximum one second for a command reaction
        cmd_timeout = 100
        while cmd_timeout > 0:
            if self.uart.any():
                cmd_output.append(self.uart.readline())
                if debug:
                    print("%8i - RX: %s" % (elapsed_micros(start), str(cmd_output[-
1])))
                if cmd_output[-1].rstrip() == b'OK':
                    if debug:
                        print("%8i - 'OK' received!" % (elapsed_micros(start)))
                    okay = True
                    delay(10)
                    cmd_timeout -= 1
            if cmd_timeout == 0 and len(cmd_output) == 0:
                if debug == True:
                    print("%8i - RX timeout of answer after sending AT command!" %
(elapsed_micros(start)))
                else:
                    print("RX timeout of answer after sending AT command!")
        # read output if present
        while self.uart.any():
            cmd_output.append(self.uart.readline())
            if debug:
                print("%8i - RX: %s" % (elapsed_micros(start), str(cmd_output[-1])))
            if cmd_output[-1].rstrip() == b'OK':
                if debug:
                    print("%8i - 'OK' received!" % (elapsed_micros(start)))
                okay = True

```

```

# handle output of AT command
if len(cmd_output) > 0:
    if cmd_output[-1].rstrip() == b'ERROR':
        raise CommandError('Command error!')
    elif cmd_output[-1].rstrip() == b'OK':
        okay = True
    elif not okay:
        # some long running commands do not return OK in case of success
        # and/or take some time to yield all output.
        if timeout == 0:
            cmd_timeout = 300
        else:
            if debug:
                print("%8i - Using RX timeout of %i ms" %
(elapsed_micros(start), timeout))
            cmd_timeout = timeout / 10
            while cmd_timeout > 0:
                delay(10)
                if self.uart.any():
                    cmd_output.append(self.uart.readline())
                    if debug:
                        print("%8i - RX: %s" % (elapsed_micros(start),
str(cmd_output[-1])))
                    if cmd_output[-1].rstrip() == b'OK':
                        okay = True
                        break
                    elif cmd_output[-1].rstrip() == b'FAIL':
                        raise CommandFailure()
                cmd_timeout -= 1
            if not okay and cmd_timeout == 0 and debug:
                print("%8i - RX-Timeout occured and no 'OK' received!" %
(elapsed_micros(start)))
        return cmd_output

```

```
@classmethod
```

```
def _join_args(cls, *args, debug=True):
```

```
    """Joins all given arguments as the ESP8266 needs them for the
argument string in a 'set' type command.
```

```
    Strings must be quoted using "" and no spaces outside of quoted
strings are allowed."""
```

```
    while type(args[0]) is tuple:
```

```
        if len(args) == 1:
```

```
            args = args[0]
```

```
    if debug:
```

```
        print(args)
```

```
    str_args = []
```

```
    for arg in args:
```

```
        if type(arg) is str:
```

```
            str_args.append("'" + arg + "'")
```

```
        elif type(arg) is bytes:
```

```

        str_args.append(arg.decode())
    elif type(arg) is bool:
        str_args.append(str(int(arg)))
    else:
        str_args.append(str(arg))
if debug:
    print(str_args)
return ','.join(str_args).encode()

```

```

@classmethod
def _parse_accesspoint_str(cls, ap_str):
    """Parse an accesspoint string description into a hashmap
    containing its parameters. Returns None if string could not be
    split into 3 or 5 fields."""
    if type(ap_str) is str:
        ap_str = ap_str.encode()
        ap_params = ap_str.split(b',')
        if len(ap_params) == 5:
            (enc_mode, ssid, rssi, mac, channel) = ap_params
            ap = {
                'encryption_protocol': int(enc_mode),
                'ssid': ssid,
                'rssi': int(rssi),
                'mac': mac,
                'channel': int(channel)
            }
        elif len(ap_params) == 3:
            (enc_mode, ssid, rssi) = ap_params
            ap = {
                'encryption_protocol': int(enc_mode),
                'ssid': ssid,
                'rssi': int(rssi),
            }
        else:
            ap = None
    return ap

```

```

"""
@classmethod
def _parse_station_ip(string)

    if type(string) is str:

        return string
    else:

        return None
"""

```

```

def _query_command(self, cmd, timeout=0, debug=False):

```

```

        """Sends a 'query' type command and return the relevant output
        line, containing the queried parameter."""
        return self._send_command(cmd + b'?', timeout=timeout,
debug=debug)[1].rstrip()

def _set_command(self, cmd, *args, timeout=0, debug=False):
    """Send a 'set' type command and return all lines of the output
    which are not command echo and status codes.
    This type of AT command usually does not return output except
    the echo and 'OK' or 'ERROR'. These are not returned by this
    method. So usually the result of this method must be an empty list!"""
    return self._send_command(cmd + b'= ' + ESP8266._join_args(args,
debug=debug), timeout=timeout, debug=debug)[1:-2]

def _execute_command(self, cmd, timeout=0, debug=False):
    """Send an 'execute' type command and return all lines of the
    output which are not command echo and status codes."""
    return self._send_command(cmd, timeout=timeout, debug=debug)[1:-2]

def test(self, debug=False):
    """Test the AT command interface."""
    return self._execute_command(CMDS_GENERIC['TEST_AT'],
debug=debug) == []

def version(self, debug=False):
    """Test the AT command interface."""
    return self._execute_command(CMDS_GENERIC['VERSION_INFO'],
debug=debug) == []

def reset(self, debug=False):
    """Reset the module and read the boot message.
    ToDo: Interpret the boot message and do something reasonable with
    it, if possible."""
    boot_log = []
    if debug:
        start = micros()
    self._execute_command(CMDS_GENERIC['RESET'], debug=debug)
    # wait for module to boot and messages appearing on self.uart
    timeout = 300
    while not self.uart.any() and timeout > 0:
        delay(10)
        timeout -= 1
    if debug and timeout == 0:
        print("%8i - RX timeout occured!" % (elapsed_micros(start)))
    # wait for messages to finish
    timeout = 300
    while timeout > 0:
        if self.uart.any():
            boot_log.append(self.uart.readline())
            if debug:

```

```

        print("%8i - RX: %s" % (elapsed_micros(start), str(boot_log[-1])))
        delay(20)
        timeout -= 1
    if debug and timeout == 0:
        print("%8i - RTimeout occurred while waiting for module to boot!" %
(elapsed_micros(start)))
        return boot_log[-1].rstrip() == b'ready'

def get_mode(self):
    """Returns the mode the ESP WIFI is in:
    1: station mode
    2: accesspoint mode
    3: accesspoint and station mode
    Check the hashmap esp8266.WIFI_MODES for a name lookup.
    Raises an UnknownWIFIModeError if the mode was not a valid or
    unknown.
    """
    mode = int(self._query_command(CMDS_WIFI['MODE']).split(b':')[1])

    if mode in WIFI_MODES.keys():
        return mode
    else:
        raise UnknownWIFIModeError("Mode '%s' not known!" % mode)

def set_mode(self, mode, debug=False):
    """Set the given WIFI mode.
    Raises UnknownWIFIModeError in case of unknown mode."""
    if mode not in WIFI_MODES.keys():
        raise UnknownWIFIModeError("Mode '%s' not known!" % mode)
    return self._set_command(CMDS_WIFI['MODE'], mode, debug=debug)

def get_accesspoint(self, debug=False):
    """Read the SSID of the currently joined access point.
    The SSID 'No AP' tells us that we are not connected to an access
    point!"""
    answer = self._query_command(CMDS_WIFI["CONNECT"],
debug=debug)
    #print("Answer: " + str(answer))
    if answer == b'No AP':
        result = None
    else:
        result = answer.split(b'+ ' + CMDS_WIFI['CONNECT'][3:] + b':')[1][1:-1]
    return result

def connect(self, ssid, psk, debug=False):
    """Tries to connect to a WIFI network using the given SSID and
    pre shared key (PSK). Uses a 20 second timeout for the connect
    command.
    Bugs: AT firmware v0.21 has a bug to only join a WIFI which SSID
    is 10 characters long."""

```



```
self._set_command(CMDS_WIFI['CONNECT'], ssid, psk, debug=debug,
timeout=20000)
```

```
def disconnect(self, debug=False):
    """Tries to connect to a WIFI network using the given SSID and
    pre shared key (PSK)."""
    return self._execute_command(CMDS_WIFI['DISCONNECT'],
debug=debug) == []
```

```
@classmethod
def _parse_list_ap_results(cls, ap_scan_results):
    aps = []
    for ap in ap_scan_results:
        try:
            ap_str = ap.rstrip().split(CMDS_WIFI['LIST_APS'][-4:] +
b':')[1].decode()[1:-1]
        except IndexError:
            # Catching this exception means the line in scan result
            # was probably rubbish
            continue
        # parsing the ap_str may not work because of rubbish strings
        # returned from the AT command. None is returned in this case.
        ap = ESP8266._parse_accesspoint_str(ap_str)
        if ap:
            aps.append(ap)
    return aps
```

```
def list_all_accesspoints(self, debug=False):
    """List all available access points.
    TODO: The IoT AT firmware 0.9.5 seems to sporadically yield
    rubbish or mangled AP-strings. Check needed!"""
    return
ESP8266._parse_list_ap_results(self._execute_command(CMDS_WIFI['LIST
_APS'], debug=debug))
```

```
def list_accesspoints(self, *args):
    """List accesspoint matching the parameters given by the
    argument list.
    The arguments may be of the types string or integer. Strings can
    describe MAC addresses or SSIDs while the integers refer to
    channel names."""
    return
ESP8266._parse_list_ap_results(self._set_command(CMDS_WIFI['LIST_AP
S'], args))
```

```
def set_accesspoint_config(self, ssid, password, channel, encrypt_proto,
debug=False):
    """Configure the parameters for the accesspoint mode. The module
    must be in access point mode for this to work.
    After setting the parameters the module is reset to
```

activate them.

The password must be at least 8 characters long up to a maximum of 64 characters.

WEP is not allowed to be an encryption protocol.

Raises CommandFailure in case the WIFI mode is not set to mode 2 (access point) or 3 (access point and station) or the WIFI parameters are not valid."""

```
if self.get_mode() not in (2, 3):
```

```
    raise CommandFailure('WIFI not set to an access point mode!')
```

```
if type(ssid) is not str:
```

```
    raise CommandFailure('SSID must be of type str!')
```

```
if type(password) is not str:
```

```
    raise CommandFailure('Password must be of type str!')
```

```
if len(password) > 64 or len(password) < 8:
```

```
    raise CommandFailure('Wrong password length (8..64)!')
```

```
if channel not in range(1, 15) and type(channel) is not int:
```

```
    raise CommandFailure('Invalid WIFI channel!')
```

```
if encrypt_proto not in (0, 2, 3, 4) or type(encrypt_proto) is not int:
```

```
    raise CommandFailure('Invalid encryption protocol!')
```

```
self._set_command(CMDS_WIFI['AP_SET_PARAMS'], ssid, password,  
channel, encrypt_proto, debug=debug)
```

```
self.reset()
```

```
def get_accesspoint_config(self):
```

```
    """Reads the current access point configuration. The module must  
    be in an access point mode to work.
```

```
    Returns a hashmap containing the access point parameters.
```

```
    Raises CommandFailure in case of wrong WIFI mode set."""
```

```
if self.get_mode() not in (2, 3):
```

```
    raise CommandFailure('WIFI not set to an access point mode!')
```

```
(ssid, password, channel, encryption_protocol) =
```

```
self._query_command(CMDS_WIFI['AP_SET_PARAMS'],  
debug=False).split(b':')[1].split(b',')
```

```
return {
```

```
    'ssid': ssid,
```

```
    'password': password,
```

```
    'channel': int(channel),
```

```
    'encryption_protocol': int(encryption_protocol)
```

```
}
```

```
def list_stations(self):
```

```
    """List IPs of stations which are connected to the access point.
```

```
    ToDo: Parse result and return python list of IPs (as str)."""
```

```
return self._execute_command(CMDS_WIFI['AP_LIST_STATIONS'],  
debug=False)
```

```
def set_dhcp_config(self, mode, status, debug=False):
```

```
    """Set the DHCP configuration for a specific mode.
```

Oddities:

The mode seems not to be the WIFI mode known from the methods `set_mode()` and `get_mode()`. The mode are as follows according to the Espressif documentation:

- 0: access point (softAP)
- 1: station
- 2: access point and station

The second argument (status) is strange as well:

- 0: enable
- 1: disable

```
"""
```

```
# Invert status to make the call to this method reasonable.
```

```
if type(status) is int:
```

```
    status = bool(status)
```

```
if type(status) is bool:
```

```
    status = not status
```

```
    return self._set_command(CMDS_WIFI['DHCP_CONFIG'], mode,
status, debug=debug)
```

```
def set_autoconnect(self, autoconnect, debug=False):
```

```
    """Set if the module should connect to an access point on
startup."""
```

```
    return self._set_command(CMDS_WIFI['SET_AUTOCONNECT'],
autoconnect, debug=debug)
```

```
def get_station_ip(self, debug=False):
```

```
    """get the IP address of the module in station mode.
```

```
    The IP address must be given as a string. No check on the
correctness of the IP address is made."""
```

```
    return self._query_command(CMDS_WIFI['SET_STATION_IP'],
debug=debug)
```

```
def set_station_ip(self, ip_str, debug=False):
```

```
    """Set the IP address of the module in station mode.
```

```
    The IP address must be given as a string. No check on the
correctness of the IP address is made."""
```

```
    return self._set_command(CMDS_WIFI['SET_STATION_IP'], ip_str,
debug=debug)
```

```
def get_accesspoint_ip(self, debug=False):
```

```
    """get the IP address of the module in access point mode.
```

```
    The IP address must be given as a string. No check on the
correctness of the IP address is made."""
```

```
    return self._query_command(CMDS_WIFI['SET_AP_IP'], debug=debug)
```

```
def set_accesspoint_ip(self, ip_str, debug=False):
```

```
    """Set the IP address of the module in access point mode.
```

```
    The IP address must be given as a string. No check on the
correctness of the IP address is made."""
```

```
    return self._set_command(CMDS_WIFI['SET_AP_IP'], ip_str,
debug=debug)
```

```

def get_connection_status(self):
    """Get connection information.
    ToDo: Parse returned data and return python data structure."""
    return self._execute_command(CMDS_IP['STATUS'])

def start_connection(self, protocol, dest_ip, dest_port, debug=False):
    """Start a TCP or UDP connection.
    ToDo: Implement MUX mode. Currently only single connection mode is
    supported!"""
    self._set_command(CMDS_IP['START'], protocol, dest_ip, dest_port,
debug=debug)

def send(self, data, debug=False):
    """Send data over the current connection."""
    self._set_command(CMDS_IP['SEND'], len(data), debug=debug)
    print(b'>' + data)
    self.uart.write(data)

def ping(self, destination, debug=False):
    """Ping the destination address or hostname."""
    return self._set_command(CMDS_IP['PING'], destination, debug=debug)

```

## Anexo 3: Programa final – Tarjeta 1

La tarjeta Pyboard #1 se encarga del modo automático (del proceso Almacenar y Liberar) y del modo manual, es decir todo el proceso de control secuencial del subsistema IMS10. Los archivos contenidos en la memoria flash de la tarjeta son detallados a continuación.

### a) Main.py

```
import pyb
import funciones
import pywifi

pyb.LED(1).on()    # Led INICIO
pyb.delay(500)
pyb.LED(1).off()

pyb.LED(2).on()    # Led WIFI
pyb.delay(500)
pyb.LED(2).off()
# Conexion WIFI
esp = pywifi.ESP8266(1, 115200)
funciones.conexion_wifi()

# MAIN

while True:
    pyb.LED(3).on() # Led MAIN
    pyb.delay(1000)
    pyb.LED(3).off()

    esp.start_connection(protocol='TCP',          dest_ip='controlims10.co.nf',
dest_port=80, debug=True)
    marcha_automatico = esp.send('GET  /banda_auto/getMarcha.php
HTTP/1.0\r\nHost: controlims10.co.nf\r\n\r\n', debug=True)
    pyb.delay(20)

    esp.start_connection(protocol='TCP',          dest_ip='controlims10.co.nf',
dest_port=80, debug=True)
    marcha_manual = esp.send('GET  /banda_manual/getMarcha.php
HTTP/1.0\r\nHost: controlims10.co.nf\r\n\r\n', debug=True)
    pyb.delay(20)

    if (marcha_automatico == '1'):
        pyb.LED(4).on()
        pyb.delay(1000)
        pyb.LED(4).off()

    esp.start_connection(protocol='TCP',          dest_ip='controlims10.co.nf',
dest_port=80, debug=True)
```

```

    almacenar = esp.send('GET /banda_auto/getAlmacenar.php
HTTP/1.0\r\nHost: controlims10.co.nf\r\n\r\n', debug=True)
    pyb.delay(20)

    esp.start_connection(protocol='TCP', dest_ip='controlims10.co.nf',
dest_port=80, debug=True)
    liberar = esp.send('GET /banda_auto/getLiberar.php HTTP/1.0\r\nHost:
controlims10.co.nf\r\n\r\n', debug=True)
    pyb.delay(20)

    if (almacenar == '1'):
        funciones.almacenar()

    if (liberar == '1'):
        funciones.liberar()

    if marcha_manual == '1':
        funciones.movimiento_banda()

```

## b) Funciones.py

```

import machine
from pyb import Pin, Timer, ExtInt
import pyb
import pywifi
import machine
import json

def opcion_reset():
    x4 = machine.Pin('X4', machine.Pin.IN, machine.Pin.PULL_UP) #
    Paro_Pyboard 2

    if x4.value():
        pyb.hard_reset()

def conexion_wifi():
    # Luz de inicio
    pyb.LED(4).on() # BLUE LED ON Si fue exitoso
    pyb.delay(1000)
    pyb.LED(4).off() # Se apagan LEDs luego de 2s

    # Se resetea el dispositivo
    rst = Pin('X12', Pin.OUT)
    rst.low()
    pyb.delay(20)
    rst.high()
    pyb.delay(500)
    esp = pywifi.ESP8266(1, 115200) # Configuracion de modulo

    # Prueba de arranque en modo AT

```

```

if esp.test():
    pyb.LED(1).on() # GREEN LED ON si fue exitoso
else:
    pyb.LED(4).on() # BLUE LED ON si hubo error

pyb.delay(1000)
pyb.LED(4).off() # Se apagan LEDs luego de 2s
pyb.LED(1).off()
# Otra prueba de arranque en modo AT
if esp.test():
    pyb.LED(1).on() # GREEN LED ON si fue exitoso
else:
    pyb.LED(4).on() # BLUE LED ON si hubo error

pyb.delay(500)
pyb.LED(4).off() # Se apagan LEDs luego de 2s
pyb.LED(1).off()

esp.set_mode(1)

if esp.get_mode() == 1:
    pyb.LED(1).on() # GREEN LED ON si fue exitoso
else:
    pyb.LED(4).on() # GREEN BLUE ON si hubo error

pyb.delay(1000)
pyb.LED(4).off() # Se apagan LEDs luego de 2s
pyb.LED(1).off()

# Connecting to network

# esp.connect(ssid='Sapo', psk='sapo1234')
# esp.connect(ssid='yesmania-Cnt', psk='70944321')
esp.connect(ssid='tupapi', psk='12345678')
# esp.connect(ssid='MANUEL ESPARZA', psk='ESPARZA24')
pyb.delay(20)

# Conexion exitosa, se encienden los dos LEDs

pyb.LED(1).on() # GREEN LED ON
pyb.LED(4).on() # BLUE LED ON
pyb.delay(1000)
pyb.LED(4).off() # Se apagan LEDs luego de 2s
pyb.LED(1).off()

def almacenar():
    y1 = machine.Pin('Y1', machine.Pin.IN, machine.Pin.PULL_UP) #
    Sensor_Magnetico 1

```

```

    y2 = machine.Pin('Y2', machine.Pin.IN, machine.Pin.PULL_UP)      #
Sensor_Magnetico 2
    y7 = machine.Pin('Y7', machine.Pin.IN, machine.Pin.PULL_UP)      #
Sensor_Almacen_LLeno
    y3 = machine.Pin('Y3', machine.Pin.OUT) # Cilindros paralelos
    y4 = machine.Pin('Y4', machine.Pin.OUT) # Cilindro elevador
    y5 = machine.Pin('Y5', machine.Pin.OUT) # Cilindro de parada
    y6 = machine.Pin('Y6', machine.Pin.OUT) # Cilindro separador
    p2 = machine.Pin('X2', machine.Pin.OUT) # EN GIRO1
    p3 = machine.Pin('X3', machine.Pin.OUT) # EN GIRO2
    p1 = Pin('X1') # X1 has TIM2, CH1 # PWM
    tim = Timer(2, freq=2500)
    ch1 = tim.channel(1, Timer.PWM, pin=p1)
    esp = pywifi.ESP8266(1, 115200)
    bandera2 = 1

# Inicializamos interrupcion de timer para Paro
tim2 = Timer(10, freq=1000)
tim2.freq(8) # Cada 0.125 seg

# Programa Principal para ALMACENADO DE PALLETS
while bandera2 == 1:

    # Inicializamos
    p2.low()
    p3.low()
    y3.low()
    y4.low()
    y5.low()

    if True: # Marcha
        bandera1 = 1

        while (bandera1 == 1 and bandera2 == 1):

            # Iniciamos llamada de interrupcion para Paro
            tim2.callback(lambda t: opcion_reset())

            # Se pregunta por el Sensor_Almacen_Lleno
            if y7.value():
                bandera2 = 0

            esp.start_connection(protocol='TCP',
dest_ip='controlims10.co.nf', dest_port=80, debug=True)
            esp.post('POST /banda_auto/updateSFC.php HTTP/1.0\r\nHost:
controlims10.co.nf\r\nAccept: */*\r\nContent-Length: 9\r\nContent-Type:
application/json\r\n\r\n{"sfc":1}', debug=True)
            pyb.delay(20)

```



```

        break

        # Movimiento de la banda transportadora
        ch1.pulse_width_percent(50)
        p2.low()
        p3.high()

        esp.start_connection(protocol='TCP', dest_ip='controlims10.co.nf',
        dest_port=80, debug=True)
        esp.post('POST /banda_auto/updateMotor.php HTTP/1.0\r\nHost:
        controlims10.co.nf\r\nAccept: */*\r\nContent-Length: 11\r\nContent-Type:
        application/json\r\n\r\n{"motor":1}', debug=True)
        pyb.delay(20)

        while (not y1.value()):
            pass

        # Detecto el sensor magnetico B1
        # Disminuye la velocidad y sigue avanzando
        # Ademas se activa el cilindro de parada
        ch1.pulse_width_percent(30)
        p2.low()
        p3.high()
        y5.high()

        esp.start_connection(protocol='TCP', dest_ip='controlims10.co.nf',
        dest_port=80, debug=True)
        esp.send('POST /banda_auto/updateSM1.php HTTP/1.0\r\nHost:
        controlims10.co.nf\r\nAccept: */*\r\nContent-Length: 9\r\nContent-Type:
        application/json\r\n\r\n{"sm1":1}', debug=True)
        pyb.delay(20)

        pyb.delay(5000)

        # Se detiene la banda
        p2.low()
        p3.low()

        esp.start_connection(protocol='TCP', dest_ip='controlims10.co.nf',
        dest_port=80, debug=True)
        esp.send('POST /banda_auto/updateMotor.php HTTP/1.0\r\nHost:
        controlims10.co.nf\r\nAccept: */*\r\nContent-Length: 11\r\nContent-Type:
        application/json\r\n\r\n{"motor":0}', debug=True)
        pyb.delay(20)

        pyb.delay(4000)

        # Se activa el Cilindro paralelo y el de elevacion
        # Para subir el pallet y almacenarlo
        y5.low() # Se desactiva el Cilindro de parada

```

```

y3.high() # Se activa el Cilindro paralelo
pyb.delay(4000)
y4.high() # Se activa el Cilindro elevador
pyb.delay(6000)
y4.low() # Se desactiva Cilindro elevador
pyb.delay(5000)
y3.low() # Se desactiva Cilindro paralelo

pyb.delay(15000)
tim2.callback(None)

```

```

def liberar():
    y1 = machine.Pin('Y1', machine.Pin.IN, machine.Pin.PULL_UP) #
    Sensor_Magnetico1
    y2 = machine.Pin('Y2', machine.Pin.IN, machine.Pin.PULL_UP) #
    Sensor_Magnetico2
    y7 = machine.Pin('Y7', machine.Pin.IN, machine.Pin.PULL_UP) #
    Sensor_Almacen_LLeno
    y3 = machine.Pin('Y3', machine.Pin.OUT) # Cilindros paralelos
    y4 = machine.Pin('Y4', machine.Pin.OUT) # Cilindro elevador
    y5 = machine.Pin('Y5', machine.Pin.OUT) # Cilindro de parada
    y6 = machine.Pin('Y6', machine.Pin.OUT) # Cilindro separador
    p2 = machine.Pin('X2', machine.Pin.OUT) # EN GIRO1
    p3 = machine.Pin('X3', machine.Pin.OUT) # EN GIRO2
    p1 = Pin('X1') # X1 has TIM2, CH1 # PWM
    tim = Timer(2, freq=2500)
    ch1 = tim.channel(1, Timer.PWM, pin=p1)
    esp = pywifi.ESP8266(1, 115200)
    bandera3 = 1

    # Inicializamos interrupcion de timer para Paro
    tim2 = Timer(10, freq=1000)
    tim2.freq(8) # Cada 0.125 seg

    # Programa Principal para LIBERAR PALLETS
    while bandera3 == 1:

        # Inicializamos
        p2.low()
        p3.low()
        y3.low()
        y4.low()
        y5.low()

        if y7.value(): # Para que no se entre si no esta lleno
            bandera3 = 1 # Habria que considerar opcion para
        else: # ingresar con 1 o 2 pallets
            bandera3 = 0

```

```

break

if True:    # Marcha
    bandera1 = 1
    cnt = 0
    while (cnt < 4 and bandera1 == 1 and bandera3 == 1):
        bandera2 = 1

        # Iniciamos llamada de interrupcion para Paro
        tim2.callback(lambda t: opcion_reset())

        # Se activan los cilindros paralelos y el elevador
        # Para poder agarrar el pallet
        pyb.delay(2000)
        y3.high()
        pyb.delay(4000)
        y4.high()
        pyb.delay(8000)

        # Se abre el cilindro separador
        # De esta forma caera el pallet
        y6.high()
        pyb.delay(2000)
        y3.low()
        pyb.delay(8000)
        y6.low()
        pyb.delay(2000)
        y4.low()
        pyb.delay(10000)

        # Movimiento de la banda transportadora
        # hasta que detecte sensor2

        esp.start_connection(protocol='TCP', dest_ip='controlims10.co.nf',
dest_port=80, debug=True)
        esp.post('POST /banda_auto/updateMotor.php HTTP/1.0\r\nHost:
controlims10.co.nf\r\nAccept: */*\r\nContent-Length: 11\r\nContent-Type:
application/json\r\n\r\n{"motor":1}', debug=True)
        pyb.delay(20)

        ch1.pulse_width_percent(50)
        p2.low()
        p3.high()

        while (not y2.value()):
            pass

        p2.low()
        p3.low()

```

```

        esp.start_connection(protocol='TCP', dest_ip='controlims10.co.nf',
dest_port=80, debug=True)
        esp.post('POST /banda_auto/updateSM2.php HTTP/1.0\r\nHost:
controlims10.co.nf\r\nAccept: */*\r\nContent-Length: 9\r\nContent-Type:
application/json\r\n\r\n{"sm2":1}', debug=True)
        pyb.delay(20)

        # Se mueve la banda en sentido contrario y mas rapido
        ch1.pulse_width_percent(60)
        p2.high()
        p3.low()

        while bandera2 == 1:
            if y1.value():
                cnt = cnt + 1
                bandera2 = 0

        esp.start_connection(protocol='TCP', dest_ip='controlims10.co.nf',
dest_port=80, debug=True)
        esp.post('POST /banda_auto/updateSM1.php HTTP/1.0\r\nHost:
controlims10.co.nf\r\nAccept: */*\r\nContent-Length: 9\r\nContent-Type:
application/json\r\n\r\n{"sm1":1}', debug=True)
        pyb.delay(20)

        pyb.delay(10000)
        p2.low()
        p3.low()

        if cnt == 4:
            bandera3 = 0

        tim2.callback(None)

def movimiento_banda():
    y1 = machine.Pin('Y1', machine.Pin.IN, machine.Pin.PULL_UP) # Sensor1
    y2 = machine.Pin('Y2', machine.Pin.IN, machine.Pin.PULL_UP) # Sensor2
    p2 = machine.Pin('X2', machine.Pin.OUT)
    p3 = machine.Pin('X3', machine.Pin.OUT)
    p1 = Pin('X1') # X1 has TIM2, CH1
    tim = Timer(2, freq=2500)
    ch1 = tim.channel(1, Timer.PWM, pin=p1)
    esp = pywifi.ESP8266(1, 115200)
    #ancho = '50'

    # Inicializamos interrupcion de timer para Paro
    tim2 = Timer(10, freq=1000)
    tim2.freq(8) # Cada 0.125 seg

    # Programa Principal

```

```

while True:

    # Iniciamos llamada de interrupcion para Paro1
    tim2.callback(lambda t: opcion_reset())

    # Cambios por parte del usuario

    # GIRO, FRECUENCIA, ANCHO DE PULSO
    esp.start_connection(protocol='TCP', dest_ip='controlims10.co.nf',
dest_port=80, debug=True)
    giro,frecuencia,ancho = json.getValues(esp.send('GET
/banda_manual/getGiroFreqPulso.php
controlims10.co.nf\r\n\r\n', debug=True))
    pyb.delay(20)

    print("DATOS")
    print(giro)
    print(ancho)
    print(frecuencia)

    tim = Timer(2, freq=int(frecuencia))

    # Si se activa alguno de los sensores la velocidad es minima para
cualquier frecuencia
    if y1.value() or y2.value():
        ancho = '45'

    # MOVIMIENTO
    if (giro == '0'):
        # Movimiento secundario
        ch1.pulse_width_percent(int(ancho))
        p2.high()
        p3.low()
    else:
        # Movimiento principal
        ch1.pulse_width_percent(int(ancho))
        p2.low()
        p3.high()
    tim2.callback(None)

```

### c) Json.py

```

def extractJsonUART(val):
    txt_json = ""
    str_ant = ""
    for c in val:
        str_ant = str_ant + c
        if (c == '{'):
            str_cont = val.replace(str_ant, "")

```

```

        for t in str_cont:
            if (t == '}'):
                break
            txt_json = txt_json + t
    return txt_json

```

```

def getValues(val):
    return val.split(",")

```

#### d) Pywifi.py

Este archivo fue creado a partir de la librería pywifi pero contiene la función send que fue modificada y la función post que fue añadida.

```

from pyb import UART
from pyb import delay
from pyb import micros, elapsed_micros
import json

# This hashmap collects all generic AT commands
CMDS_GENERIC = {
    'TEST_AT': b'AT',
    'RESET': b'AT+RST',
    'VERSION_INFO': b'AT+GMR',
    'DEEP_SLEEP': b'AT+GSLP',
    'ECHO': b'ATE',
    'FACTORY_RESET': b'AT+RESTORE',
    'UART_CONFIG': b'AT+UART'
}

# All WIFI related AT commands
CMDS_WIFI = {
    'MODE' : b'AT+CWMODE',
    'CONNECT': b'AT+CWJAP',
    'LIST_APS': b'AT+CWLAP',
    'DISCONNECT': b'AT+CWQAP',
    'AP_SET_PARAMS': b'AT+CWSAP',
    'AP_LIST_STATIONS': b'AT+CWLIF',
    'DHCP_CONFIG': b'AT+CWDHCP',
    'SET_AUTOCONNECT': b'AT+CWAUTOCONN',
    'SET_STATION_MAC': b'AT+CIPSTAMAC',
    'SET_AP_MAC': b'AT+CIPAPMAC',
    'SET_STATION_IP': b'AT+CIPSTA',
    'SET_AP_IP': b'AT+CIPAP',

```

```

'START_SMART':b'AT+CWSMARTSTART'
}

# IP networking related AT commands
CMDS_IP = {
    'STATUS': b'AT+CIPSTATUS',
    'START': b'AT+CIPSTART',
    'SEND': b'AT+CIPSEND',
    'CLOSE': b'AT+CIPCLOSE',
    'GET_LOCAL_IP': b'AT+CIFSR', #Get ESP8266 IP address from the router
connected
    'SET_MUX_MODE': b'AT+CIPMUX',
    'CONFIG_SERVER': b'AT+CIPSERVER',
    'SET_TX_MODE': b'AT+CIPMODE',
    'SET_TCP_SERVER_TIMEOUT': b'AT+CIPSTO',
    'UPGRADE': b'AT+CIUPDATE',
    'PING': b'AT+PING'
}

# WIFI network modes the ESP8266 knows to handle
WIFI_MODES = {
    1: 'Station',
    2: 'Access Point',
    3: 'Access Point + Station',
}
# Reverse feed lookup table
for key in WIFI_MODES.keys():
    WIFI_MODES[WIFI_MODES[key]] = key

# WIFI network security protocols known to the ESP8266 module
WIFI_ENCRYPTION_PROTOCOLS = {
    0: 'OPEN',
    1: 'WEP',
    2: 'WPA_PSK',
    3: 'WPA2_PSK',
    4: 'WPA_WPA2_PSK'
}
# Reverse feed lookup table
for key in WIFI_ENCRYPTION_PROTOCOLS.keys():

WIFI_ENCRYPTION_PROTOCOLS[WIFI_ENCRYPTION_PROTOCOLS[key]]
= key

```

```

class CommandError(Exception):
    pass

class CommandFailure(Exception):
    pass

class UnknownWIFIModeError(Exception):
    pass

class ESP8266(object):

    def __init__(self, uart=1, baud_rate=115200):
        """Initialize this module. uart may be an integer or an instance
        of pyb.UART. baud_rate can be used to set the Baud rate for the
        serial communication."""
        if uart:
            if type(uart) is int:
                self.uart = UART(uart, baud_rate)
            elif type(uart) is UART:
                self.uart = uart
            else:
                raise Exception("Argument 'uart' must be an integer or pyb.UART
object!")
        else:
            raise Exception("Argument uart must not be 'None'!")

    def _send_command(self, cmd, timeout=0, debug=False):
        """Send a command to the ESP8266 module over UART and return the
        output.
        After sending the command there is a 1 second timeout while
        waiting for an anser on UART. For long running commands (like AP
        scans) there is an additional 3 seconds grace period to return
        results over UART.
        Raises an CommandError if an error occurs and an CommandFailure
        if a command fails to execute."""
        if debug:
            start = micros()
        cmd_output = []
        okay = False
        if cmd == " or cmd == b":
            raise CommandError("Unknown command " + cmd + "!")
        # AT commands must be finalized with an '\r\n'
        cmd += '\r\n'

```



```

if debug:
    print("%8i - TX: %s" % (elapsed_micros(start), str(cmd)))
self.uart.write(cmd)
# wait at maximum one second for a command reaction
cmd_timeout = 100
while cmd_timeout > 0:
    if self.uart.any():
        cmd_output.append(self.uart.readline())
        if debug:
            print("%8i - RX: %s" % (elapsed_micros(start), str(cmd_output[-
1])))
        if cmd_output[-1].rstrip() == b'OK':
            if debug:
                print("%8i - 'OK' received!" % (elapsed_micros(start)))
            okay = True
            delay(10)
        cmd_timeout -= 1
if cmd_timeout == 0 and len(cmd_output) == 0:
    if debug == True:
        print("%8i - RX timeout of answer after sending AT command!" %
(elapsed_micros(start)))
    else:
        print("RX timeout of answer after sending AT command!")
# read output if present
while self.uart.any():
    cmd_output.append(self.uart.readline())
    if debug:
        print("%8i - RX: %s" % (elapsed_micros(start), str(cmd_output[-1])))
    if cmd_output[-1].rstrip() == b'OK':
        if debug:
            print("%8i - 'OK' received!" % (elapsed_micros(start)))
        okay = True
# handle output of AT command
if len(cmd_output) > 0:
    if cmd_output[-1].rstrip() == b'ERROR':
        raise CommandError('Command error!')
    elif cmd_output[-1].rstrip() == b'OK':
        okay = True
    elif not okay:
        # some long running commands do not return OK in case of success
        # and/or take some time to yield all output.
        if timeout == 0:
            cmd_timeout = 300

```

```

else:
    if debug:
        print("%8i - Using RX timeout of %i ms" %
              (elapsed_micros(start), timeout))
        cmd_timeout = timeout / 10
        while cmd_timeout > 0:
            delay(10)
            if self.uart.any():
                cmd_output.append(self.uart.readline())
                if debug:
                    print("%8i - RX: %s" % (elapsed_micros(start),
str(cmd_output[-1])))
                if cmd_output[-1].rstrip() == b'OK':
                    okay = True
                    break
                elif cmd_output[-1].rstrip() == b'FAIL':
                    raise CommandFailure()
                cmd_timeout -= 1
            if not okay and cmd_timeout == 0 and debug:
                print("%8i - RX-Timeout occured and no 'OK' received!" %
                      (elapsed_micros(start)))
        return cmd_output

```

```
@classmethod
```

```

def _join_args(cls, *args, debug=True):
    """Joins all given arguments as the ESP8266 needs them for the
    argument string in a 'set' type command.
    Strings must be quoted using "" and no spaces outside of quoted
    strings are allowed."""
    while type(args[0]) is tuple:
        if len(args) == 1:
            args = args[0]
    if debug:
        print(args)
    str_args = []
    for arg in args:
        if type(arg) is str:
            str_args.append("'" + arg + "'")
        elif type(arg) is bytes:
            str_args.append(arg.decode())
        elif type(arg) is bool:
            str_args.append(str(int(arg)))
        else:

```

```

        str_args.append(str(arg))
    if debug:
        print(str_args)
    return ','.join(str_args).encode()

@classmethod
def _parse_accesspoint_str(cls, ap_str):
    """Parse an accesspoint string description into a hashmap
    containing its parameters. Returns None if string could not be
    split into 3 or 5 fields."""
    if type(ap_str) is str:
        ap_str = ap_str.encode()
        ap_params = ap_str.split(b',')
        if len(ap_params) == 5:
            (enc_mode, ssid, rssi, mac, channel) = ap_params
            ap = {
                'encryption_protocol': int(enc_mode),
                'ssid': ssid,
                'rssi': int(rssi),
                'mac': mac,
                'channel': int(channel)
            }
        elif len(ap_params) == 3:
            (enc_mode, ssid, rssi) = ap_params
            ap = {
                'encryption_protocol': int(enc_mode),
                'ssid': ssid,
                'rssi': int(rssi),
            }
        else:
            ap = None
    return ap

"""
@classmethod
def _parse_station_ip(string)

    if type(string) is str:

        return string
    else:

        return None

```

'''

```
def _query_command(self, cmd, timeout=0, debug=False):
    """Sends a 'query' type command and return the relevant output
    line, containing the queried parameter."""
    return self._send_command(cmd + b'?', timeout=timeout,
debug=debug)[1].rstrip()

def _set_command(self, cmd, *args, timeout=0, debug=False):
    """Send a 'set' type command and return all lines of the output
    which are not command echo and status codes.
    This type of AT command usually does not return output except
    the echo and 'OK' or 'ERROR'. These are not returned by this
    method. So usually the result of this method must be an empty list!"""
    return self._send_command(cmd + b'=' + ESP8266._join_args(args,
debug=debug), timeout=timeout, debug=debug)[1:-2]

def _execute_command(self, cmd, timeout=0, debug=False):
    """Send an 'execute' type command and return all lines of the
    output which are not command echo and status codes."""
    return self._send_command(cmd, timeout=timeout, debug=debug)[1:-2]

def test(self, debug=False):
    """Test the AT command interface."""
    return self._execute_command(CMDS_GENERIC['TEST_AT'],
debug=debug) == []

def version(self, debug=False):
    """Test the AT command interface."""
    return self._execute_command(CMDS_GENERIC['VERSION_INFO'],
debug=debug) == []

def reset(self, debug=False):
    """Reset the module and read the boot message.
    ToDo: Interpret the boot message and do something reasonable with
    it, if possible."""
    boot_log = []
    if debug:
        start = micros()
    self._execute_command(CMDS_GENERIC['RESET'], debug=debug)
    # wait for module to boot and messages appearing on self.uart
    timeout = 300
    while not self.uart.any() and timeout > 0:
```

```

        delay(10)
        timeout -= 1
    if debug and timeout == 0:
        print("%8i - RX timeout occurred!" % (elapsed_micros(start)))
    # wait for messages to finish
    timeout = 300
    while timeout > 0:
        if self.uart.any():
            boot_log.append(self.uart.readline())
            if debug:
                print("%8i - RX: %s" % (elapsed_micros(start), str(boot_log[-1])))
            delay(20)
            timeout -= 1
    if debug and timeout == 0:
        print("%8i - RTimeout occurred while waiting for module to boot!" %
              (elapsed_micros(start)))
    return boot_log[-1].rstrip() == b'ready'

def get_mode(self):
    """Returns the mode the ESP WIFI is in:
    1: station mode
    2: accesspoint mode
    3: accesspoint and station mode
    Check the hashmap esp8266.WIFI_MODES for a name lookup.
    Raises an UnknownWIFIModeError if the mode was not a valid or
    unknown.
    """
    mode = int(self._query_command(CMDS_WIFI['MODE']).split(b':')[1])

    if mode in WIFI_MODES.keys():
        return mode
    else:
        raise UnknownWIFIModeError("Mode '%s' not known!" % mode)

def set_mode(self, mode, debug=False):
    """Set the given WIFI mode.
    Raises UnknownWIFIModeError in case of unknown mode."""
    if mode not in WIFI_MODES.keys():
        raise UnknownWIFIModeError("Mode '%s' not known!" % mode)
    return self._set_command(CMDS_WIFI['MODE'], mode, debug=debug)

def get_accesspoint(self, debug=False):
    """Read the SSID of the currently joined access point.

```

```

    The SSID 'No AP' tells us that we are not connected to an access
    point!"""
    answer = self._query_command(CMDS_WIFI["CONNECT"],
debug=debug)
    #print("Answer: " + str(answer))
    if answer == b'No AP':
        result = None
    else:
        result = answer.split(b'+ ' + CMDS_WIFI['CONNECT'][3:] + b':')[1][1:-1]
    return result

def connect(self, ssid, psk, debug=False):
    """Tries to connect to a WIFI network using the given SSID and
    pre shared key (PSK). Uses a 20 second timeout for the connect
    command.
    Bugs: AT firmware v0.21 has a bug to only join a WIFI which SSID
    is 10 characters long."""
    self._set_command(CMDS_WIFI['CONNECT'], ssid, psk, debug=debug,
timeout=20000)

def disconnect(self, debug=False):
    """Tries to connect to a WIFI network using the given SSID and
    pre shared key (PSK)."""
    return self._execute_command(CMDS_WIFI['DISCONNECT'],
debug=debug) == []

@classmethod
def _parse_list_ap_results(cls, ap_scan_results):
    aps = []
    for ap in ap_scan_results:
        try:
            ap_str = ap.rstrip().split(CMDS_WIFI['LIST_APS'][-4:] +
b':')[1].decode()[1:-1]
        except IndexError:
            # Catching this exception means the line in scan result
            # was probably rubbish
            continue
        # parsing the ap_str may not work because of rubbish strings
        # returned from the AT command. None is returned in this case.
        ap = ESP8266._parse_accesspoint_str(ap_str)
        if ap:
            aps.append(ap)
    return aps

```

```

def list_all_accesspoints(self, debug=False):
    """List all available access points.
    TODO: The IoT AT firmware 0.9.5 seems to sporadically yield
    rubbish or mangled AP-strings. Check needed!"""
    return
ESP8266._parse_list_ap_results(self._execute_command(CMDS_WIFI['LIST_
APS'], debug=debug))

def list_accesspoints(self, *args):
    """List accesspoint matching the parameters given by the
    argument list.
    The arguments may be of the types string or integer. Strings can
    describe MAC addresses or SSIDs while the integers refer to
    channel names."""
    return
ESP8266._parse_list_ap_results(self._set_command(CMDS_WIFI['LIST_APS'
], args))

def set_accesspoint_config(self, ssid, password, channel, encrypt_proto,
debug=False):
    """Configure the parameters for the accesspoint mode. The module
    must be in access point mode for this to work.
    After setting the parameters the module is reset to
    activate them.
    The password must be at least 8 characters long up to a maximum of
    64 characters.
    WEP is not allowed to be an encryption protocol.
    Raises CommandFailure in case the WIFI mode is not set to mode 2
    (access point) or 3 (access point and station) or the WIFI
    parameters are not valid."""
    if self.get_mode() not in (2, 3):
        raise CommandFailure('WIFI not set to an access point mode!')
    if type(ssid) is not str:
        raise CommandFailure('SSID must be of type str!')
    if type(password) is not str:
        raise CommandFailure('Password must be of type str!')
    if len(password) > 64 or len(password) < 8:
        raise CommandFailure('Wrong password length (8..64)!)')
    if channel not in range(1, 15) and type(channel) is not int:
        raise CommandFailure('Invalid WIFI channel!')
    if encrypt_proto not in (0, 2, 3, 4) or type(encrypt_proto) is not int:
        raise CommandFailure('Invalid encryption protocol!')

```

```

self._set_command(CMDS_WIFI['AP_SET_PARAMS'], ssid, password,
channel, encrypt_proto, debug=debug)
self.reset()

```

```

def get_accesspoint_config(self):
    """Reads the current access point configuration. The module must
    be in an access point mode to work.
    Returns a hashmap containing the access point parameters.
    Raises CommandFailure in case of wrong WIFI mode set."""
    if self.get_mode() not in (2, 3):
        raise CommandFailure('WIFI not set to an access point mode!')
    (ssid, password, channel, encryption_protocol) =
self._query_command(CMDS_WIFI['AP_SET_PARAMS'],
debug=False).split(b':')[1].split(b',')
    return {
        'ssid': ssid,
        'password': password,
        'channel': int(channel),
        'encryption_protocol': int(encryption_protocol)
    }

```

```

def list_stations(self):
    """List IPs of stations which are connected to the access point.
    ToDo: Parse result and return python list of IPs (as str)."""
    return self._execute_command(CMDS_WIFI['AP_LIST_STATIONS'],
debug=False)

```

```

def set_dhcp_config(self, mode, status, debug=False):
    """Set the DHCP configuration for a specific mode.

```

Oddities:

The mode seems not to be the WIFI mode known from the methods set\_mode() and get\_mode(). The mode are as follows according to the Espressif documentation:

- 0: access point (softAP)
- 1: station
- 2: access point and station

The second argument (status) is strange as well:

- 0: enable
- 1: disable

```
"""
```

```

# Invert status to make the call to this method reasonable.
if type(status) is int:

```



```

        status = bool(status)
    if type(status) is bool:
        status = not status
    return self._set_command(CMDS_WIFI['DHCP_CONFIG'], mode, status,
debug=debug)

def set_autoconnect(self, autoconnect, debug=False):
    """Set if the module should connect to an access point on
    startup."""
    return self._set_command(CMDS_WIFI['SET_AUTOCONNECT'],
autoconnect, debug=debug)

def get_station_ip(self, debug=False):
    """get the IP address of the module in station mode.
    The IP address must be given as a string. No check on the
    correctness of the IP address is made."""
    return self._query_command(CMDS_WIFI['SET_STATION_IP'],
debug=debug)

def set_station_ip(self, ip_str, debug=False):
    """Set the IP address of the module in station mode.
    The IP address must be given as a string. No check on the
    correctness of the IP address is made."""
    return self._set_command(CMDS_WIFI['SET_STATION_IP'], ip_str,
debug=debug)

def get_accesspoint_ip(self, debug=False):
    """get the IP address of the module in access point mode.
    The IP address must be given as a string. No check on the
    correctness of the IP address is made."""
    return self._query_command(CMDS_WIFI['SET_AP_IP'], debug=debug)

def set_accesspoint_ip(self, ip_str, debug=False):
    """Set the IP address of the module in access point mode.
    The IP address must be given as a string. No check on the
    correctness of the IP address is made."""
    return self._set_command(CMDS_WIFI['SET_AP_IP'], ip_str,
debug=debug)

def get_connection_status(self):
    """Get connection information.
    ToDo: Parse returned data and return python data structure."""
    return self._execute_command(CMDS_IP['STATUS'])

```

```

def start_connection(self, protocol, dest_ip, dest_port, debug=False):
    """Start a TCP or UDP connection.
    ToDo: Implement MUX mode. Currently only single connection mode is
    supported!"""
    self._set_command(CMDS_IP['START'], protocol, dest_ip, dest_port,
debug=debug)

def stop_connection(self, protocol, dest_ip, dest_port, debug=False):
    """Stop a TCP or UDP connection.
    ToDo: Implement MUX mode. Currently only single connection mode is
    supported!"""
    self._set_command(CMDS_IP['STOP'], protocol, dest_ip, dest_port,
debug=debug)

def send(self, data, debug=False):
    """Send data over the current connection."""
    self._set_command(CMDS_IP['SEND'], len(data), debug=debug)
    #print(b'>' + data)
    self.uart.write(data)
    self.uart.readline()
    self.uart.readline() # Bytes received
    self.uart.readline() # /n
    self.uart.readline() # Send OK
    self.uart.readline() # /n
    self.uart.readline() # IPD, HTTP, response Request
    self.uart.readline() # Date
    self.uart.readline() # Server
    self.uart.readline() # Content length
    self.uart.readline() # Connection
    self.uart.readline() # Content-type
    self.uart.readline() # /n
    result = self.uart.readline() # json - output
    return json.extractJsonUART(str(result))

def post(self, data, debug=False):
    """Send data over the current connection."""
    self._set_command(CMDS_IP['SEND'], len(data), debug=debug)
    #print(b'>' + data)
    self.uart.readline()
    self.uart.write(data)
    self.uart.readline()
    self.uart.readline() # Bytes received

```

```

self.uart.readline() # /n
self.uart.readline() # Send OK
self.uart.readline() # /n
self.uart.readline() # IPD, HTTP, response Request
self.uart.readline() # Date
self.uart.readline() # Server
self.uart.readline() # Content length
self.uart.readline() # Connection
self.uart.readline() # Content-type
self.uart.readline() # /n
self.uart.readline()
result = self.uart.readline() # json - output
print(result)
return json.extractJsonUART(str(result))

def ping(self, destination, debug=False):
    """Ping the destination address or hostname."""
    return self._set_command(CMDS_IP['PING'], destination, debug=debug)

```

## Anexo 4: Programa final – Tarjeta 2

La tarjeta Pyboard #2 se encarga de verificar el estado del paro de la página web y también se encarga de actualizar el estado actual de los sensores y mostrarlo en el sitio web. Esta tarjeta contiene al igual que la tarjeta 1, los archivos pywifi.py y json.py que son utilizados en la función conexión\_wifi. Los archivos contenidos en la memoria flash de la tarjeta son detallados a continuación.

### a) Main.py

```
import pyb
import funciones
import pywifi
import machine

y1 = machine.Pin('Y1', machine.Pin.IN, machine.Pin.PULL_UP)      #
Sensor_Magnetico 1
y2 = machine.Pin('Y2', machine.Pin.IN, machine.Pin.PULL_UP)      #
Sensor_Magnetico 2
y3 = machine.Pin('Y3', machine.Pin.IN, machine.Pin.PULL_UP)      #
Sensor_Almacen_LLeno
y4 = machine.Pin('Y4', machine.Pin.OUT)                          # Paro externo

# Conexion WIFI
esp = pywifi.ESP8266(1, 115200)
funciones.conexion_wifi()

# MAIN

while True:

    # Revisa estado de paro de la pagina web
    esp.start_connection(protocol='TCP', dest_ip='controlims10.co.nf',
dest_port=80, debug=True)
    paro_auto = esp.send('GET /banda_auto/getParo.php HTTP/1.0\r\nHost:
controlims10.co.nf\r\n\r\n', debug=True)
    pyb.delay(20)

    if paro_auto == '1':
        y4.high()
        pyb.delay(100)
        y4.low()

    # Revisa estado de sensor 1
    if y1.value():
```

```

    sm1 = '1'
else:
    sm1 = '0'

# Revisa estado de sensor 2
if y2.value():
    sm2 = '1'
else:
    sm2 = '0'

# Revisa estado de sensor 3
if y3.value():
    sfc = '1'
else:
    sfc = '0'

# Actualiza estado de sensores

esp.start_connection(protocol='TCP', dest_ip='controlims10.co.nf',
dest_port=80, debug=True)
esp.post('POST /banda_auto/updateSensores.php HTTP/1.0\r\nHost:
controlims10.co.nf\r\nAccept: */*\r\nContent-Length: 25\r\nContent-Type:
application/json\r\n\r\n{"sm1":'+sm1+', "sm2":'+sm2+', "sfc":'+sfc+'}',
debug=True)
pyb.delay(20)

```

## b) Funciones.py

```

import machine
from pyb import Pin, Timer
import pyb
import pywifi

def conexion_wifi():
    # Luz de inicio
    pyb.LED(4).on() # BLUE LED ON Si fue exitoso
    pyb.delay(1000)
    pyb.LED(4).off() # Se apagan LEDs luego de 2s

# Se resetea el dispositivo
rst = Pin('X12', Pin.OUT)
rst.low()
pyb.delay(20)

```

```

rst.high()
pyb.delay(500)
esp = pywifi.ESP8266(1, 115200) # Configuracion de modulo

# Prueba de arranque en modo AT
if esp.test():
    pyb.LED(1).on() # GREEN LED ON si fue exitoso
else:
    pyb.LED(4).on() # BLUE LED ON si hubo error

pyb.delay(1000)
pyb.LED(4).off() # Se apagan LEDs luego de 2s
pyb.LED(1).off()
# Otra prueba de arranque en modo AT
if esp.test():
    pyb.LED(1).on() # GREEN LED ON si fue exitoso
else:
    pyb.LED(4).on() # BLUE LED ON si hubo error

pyb.delay(500)
pyb.LED(4).off() # Se apagan LEDs luego de 2s
pyb.LED(1).off()

esp.set_mode(1)

if esp.get_mode() == 1:
    pyb.LED(1).on() # GREEN LED ON si fue exitoso
else:
    pyb.LED(4).on() # GREEN BLUE ON si hubo error

pyb.delay(1000)
pyb.LED(4).off() # Se apagan LEDs luego de 2s
pyb.LED(1).off()

# Connecting to network

# esp.connect(ssid='Sapo', psk='sapo1234')
# esp.connect(ssid='yesmania-Cnt', psk='70944321')
esp.connect(ssid='tupapi', psk='12345678')
# esp.connect(ssid='MANUEL ESPARZA', psk='ESPARZA24')
pyb.delay(20)

# Conexion exitosa, se encienden los dos LEDs

```

```
pyb.LED(1).on() # GREEN LED ON
pyb.LED(4).on() # BLUE LED ON
pyb.delay(1000)
pyb.LED(4).off() # Se apagan LEDs luego de 2s
pyb.LED(1).off()
```