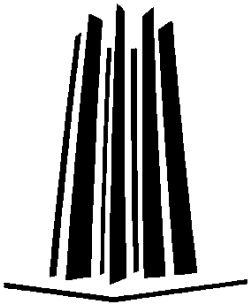
Universidad Nacional Autónoma de México



Facultad de Estudios Superiores Aragón

Título*:*

DESARROLLO E IMPLEMENTACIÓN DE UNA INTERFAZ HARDWARE-SOFTWARE SOBRE UNA CABINA DE SIMULACIÓN PARA LA CAPACITACIÓN Y ENTRENAMIENTO DE CONDUCTORES DEL STC METRO

Que para obtener el título de:

INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

Presenta:

Alejandro Sánchez Pérez

Juan Luis Ortiz Leyva

Asesor de tesis:

Dr. Ismael Díaz Rangel

Ciudad Nezahualcóyotl, Estado de México 2019

Agradecimientos:

La universidad ha sido un reto para mí, desde el primer día para adaptarme a una nueva vida académica como los últimos días de la carrera para alcanzar la meta. Con ello quiero agradecer a mi familia y más a mis padres por brindarme su apoyo, ya que, sin ellos, no habría podido llegar tan lejos en mis estudios. Dar gracias a mis mejores amigos por animarme a seguir, como también agradecer a mis profesores que, durante toda la carrera, me instruyeron y apoyaron para seguir avanzando.

También quiero dar un gran agradecimiento al área de soporte técnico del STC Metro por darme la oportunidad de hacer el servicio social en sus instalaciones y de tomar el proyecto del simulador de tren junto a mi amigo Alejandro para poder desarrollar la tesis.

De forma especial agradezco a mi tutor el Doctor Ismael Díaz Rangel por apoyarme y ayudarme junto a mi amigo para la elaboración de la tesis. Y un gran agradecimiento a mi amigo Alejandro Sánchez Pérez por su amistad y la oportunidad de hacer este trabajo con él.

Por ultimo quiero dedicar este trabajo a las siguientes personas que estuvieron conmigo en el transcurso de la carrera:

A mi padre Antonio Ortiz y a mi madre Irma Leyva por enseñarme la importancia del estudio y esfuerzo.

A mi padrino Arturo Becerra y a mi madrina María Luisa Leyva por sus enseñanzas y tiempo.

A mi ahijado Luis Jesús y mi primo Luis Arturo por mantenerme alegre después de pasar días pesados por exámenes y proyectos.

A mis hermanos José Antonio y Juan Carlos que estuvieron cada momento junto a mi apoyándome y guiándome en mantenerme en los estudios.

A mis amigos Daniel Arturo, Omar, Jorge, Carlos y Ana Patricia por su amistad, tiempo, ánimos, del cual nunca dejare de agradecerles pues son como mi segunda familia.

Y como algo especial, quiero dedicar de todo corazón todo lo que soy, mis esfuerzos y este trabajo a mis abuelos paternos y maternos.

Juan Luis Ortiz Leyva

Agradecimientos:

A la UNAM por brindarme la oportunidad de poder estudiar en sus aulas, y a sus profesores por su excelente dedicación y calidad humana que ayudo en demasía en mi formación no cabe duda de que es la máxima casa de estudios.

A mis padres Ana María Pérez Álvarez y Alejandro Sánchez Sánchez que siempre hicieron su mejor papel y esfuerzo para así alcanzar mi desarrollo como persona y mis metas como estudiante. Por enseñarme en que toda recompensa implica dedicación y esmero, porque nunca me falto su apoyo en todos los aspectos académico, de vida formación y sustento. Gracias por ser y continuar siendo mi mayor inspiración y ejemplo.

A la coordinación de soporte técnico del STC Metro por la oportunidad de trabajar en el proyecto del simulador que sirvió para la elaboración de la presente tesis.

A nuestro asesor de tesis el Dr. Ismael Díaz por su gran ayuda brindada, así como a nuestros sinodales por el tiempo que nos dedicaron.

A quienes fueron mis compañeros de clase y amigos que hicieron mi paso por la universidad más ameno.

Alejandro Sánchez Pérez

Contenido:

Índice de ilustraciones:

Índice de tablas:

Capítulo I. Introducción

El Metro de la Ciudad de México es un sistema de transporte público tipo tren metropolitano que sirve a extensas áreas de la Ciudad de México y parte del Estado de México. Su operación y explotación está a cargo del organismo público descentralizado denominado Sistema de Transporte Colectivo (STC), y su construcción está a cargo de la Secretaría de Obras y Servicios del Distrito Federal. Hasta el 12 de agosto de 2013, su construcción fue gestionada por el denominado Proyecto Metro del Distrito Federal, un organismo desconcentrado de la citada secretaría. Se conoce coloquialmente como Metro, por la contracción del término tren metropolitano.

Desde sus comienzos, la simulación de conducción es una herramienta eficiente para la capacitación de los próximos conductores del Sistema de Transporte Colectivo (STC), este nos permite que el conductor se familiarice con todas las funciones de la cabina y sea capaz de enfrentarse a diferentes situaciones reales, lo cual se traduce en un ahorro significativo de dinero debido a la reducción de los accidentes y también generará un servicio más eficiente, ya que el conductor estará preparado para cualquier avería que se pueda presentar. Al utilizar los simuladores en el entrenamiento del personal, no se corre ningún riesgo, ni en el equipo ni en las instalaciones del SCT metro y la práctica puede ser repetida para el correcto aprendizaje de los conductores.

Sin embargo, el costo de los simuladores es elevado, por esta razón, se buscó una solución económica y funcional que cubra los puntos importantes para el aprendizaje de los nuevos conductores.

1.1 Objetivo general

Desarrollar e implementar una interfaz de hardware-software sobre una cabina de simulación para la capacitación y entrenamiento de conductores del STC Metro de la Ciudad de México.

1.2 Objetivos particulares

- Implementar una interfaz de bajo costo.
- Lograr la comunicación cabina-software.
- Mejorar el entorno visual dentro de la simulación.
- Crear un cofre de simulación de fallas.

1.3 Justificación

La capacitación en términos de conducción de los trenes del sistema de transporte masivo del metro se ha hecho mediante simuladores de cabinas, son sistemas de muy alto costo de adquisición y mantenimiento. Algunos simuladores salen de operación debido a la dificultad para conseguir refacciones, lo cual implica

desaprovechar los bienes y la posibilidad de continuar la capacitación del personal. La propuesta es importante ya que ofrece un conjunto de soluciones para recuperar simuladores, utilizando materiales y recursos de bajo costo y fácil obtención; y que además supera en algunos casos las prestaciones originales de las cabinas de simulación.

1.4 Organización del trabajo

Capítulo 2. Se describe el marco teórico de la propuesta, comenzando con la información referente a la capacitación de los conductores nuevos, se continúa con una explicación de algunos simuladores para la conducción de trenes. Posteriormente se habla de dispositivos electrónicos y de control.

Capítulo 3. Desarrollo experimental. En este capítulo se hace un planteamiento y descripción a detalle de la metodología seguida para la elaboración del sistema y su incorporación a la cabina de simulación; también se habla de cómo se realizó la mejora visual de los elementos gráficos del simulador.

Capítulo 4. Se describen las pruebas realizadas a los dispositivos y al prototipo, así como los resultados arrojados en las pruebas.

Por último, se mencionan las conclusiones y se habla sobre el trabajo a futuro para mejorar la propuesta.

Capítulo 2. Marco Teórico

2.1 Capacitación de nuevos conductores

En los comienzos del STC, la capacitación fue limitada, al no poder tener equipos para el entendimiento en el área practica de cómo manejar un tren interurbano ya que la única forma de entenderlo era con visitas en las áreas de mantenimiento, por lo que el conocimiento teórico era el punto fuerte en la capacitación de los conductores. En el transcurso del tiempo, la manera de enseñar la conducción fue mejorando, al diseñar y adquirir simuladores de otros países para poder brindar los conocimientos básicos que debe aprender el operador que se encuentra en entrenamiento.

La empresa CAF (Construcciones y Auxiliar de Ferrocarriles) originaria de España, es la encargada de la construcción de trenes incluyendo el tipo interurbano, con ello, la empresa también elaborar cabinas de simulación con las características de sus modelos para su capacitación, del cual, en México la cabina de simulación más reciente en el STC es para la línea 12 del metro (Mixcoac a Tláhuac).

Desafortunadamente en la STC, al adquirir nuevos trenes, se debe actualizar los simuladores, por lo que a los anteriores se le dejan de dar mantenimiento y quedan disfuncionales con el tiempo, haciendo que los instructores tengan que pedir al personal de mantenimiento técnico que adapten las nuevas cabinas de simulación con programas que ejecuten los archivos de los anteriores simuladores, del cual, al intentar adaptarlos, puede invalidar garantías en la parte electrónica o no encontrar programas compatibles, por ello se busca la forma de reacondicionar las viejas cabinas de simulación, para mantener en óptimas condiciones las cabinas actuales y crear nuevas áreas de capacitación en las diferentes partes de la STC en la Ciudad de México.

2.1.1 Conducción de tren

Para comprender de mejor forma el manejo del metro se mostrará los controles utilizados, cabe mencionar que aún hay más controles, sin embargo, no se mencionaran, ya que no se utilizaron durante el desarrollo de la presente propuesta.

Palanca T1 (Figura 2.1). – Está palanca nos permite seleccionar que puertas se van a abrir ya sean las izquierdas derechas o ambas, en el caso de neutro no abrirá ninguna puerta. La palanca cuenta con un LED que indica la disponibilidad de la llave.

Figura 2.1 Palanca seleccionadora de puertas

Palanca de conducción (Figura 2.2). – Selecciona el modo de conducción, los modos disponibles son Neutro, Conducción Manual Controlada (CMC), Piloto Automático (PA), y Conducción Manual (CM).

Figura 2.2 Palanca seleccionador de tracción

Apertura de puertas (Figura 2.3). – Como su nombre lo indica, funciona para abrir las puertas.

Figura 2.3 Interruptor de puertas

Interruptor de marcha (Figura 2.4). – Selección de la marcha del tren ya sea hacia atrás o adelante.

Figura 2.4 Interruptor de marcha

El manipulador (Figura 2.5). – Nos permite seleccionar la tracción de T1 a T5, neutral, el frenado del tren de F1 a F6 y FU que es el freno de emergencia y, por último, incluye el arillo de hombre muerto, la función de este arillo es detectar la presión que está dando la mano del conductor al momento de hacer su trayecto, el conductor coloca su mano sobre el manipulador y jala hacia arriba el arillo. En caso de que el conductor lo suelte, después de 3 segundos se activa de forma automática (sin necesidad de ir a FU en el manipulador) el freno de emergencia.

Figura 2.5 Manipulador de velocidad y freno

Y para concluir con los controles, tenemos el botón azul para el anuncio de cierre de puertas y botón amarillo para el cierre de puertas estos se encuentran a un lado de la puerta de la cabina (Figura 2.6).

Figura 2.6 Interruptores de puertas

El cofre de fallas (Figura 2.7). – Permite la visualización de las diferentes averías que pueden presentarse durante la conducción del tren, para la solución de estas fallas se debe realizar un procedimiento específico para cada lámpara, ya sea accionar ciertas palancas e informar al centro de mando.

Figura 2.7 Cofre de fallas

De los modos de conducción disponibles se trabajó solamente con el modo manual y pilotaje automático a continuación se dará una breve explicación de cada uno:

Pilotaje automático (PA): El pilotaje automático es el modo de conducción predeterminado en todas las vías principales y en las estaciones. En éste modo, el conductor solo controla los sistemas secundarios, como la apertura y cierre de puertas, avisos sonoros, etc. Para poder avanzar se activa un sistema de seguridad, en el cual el tren no podrá avanzar hasta que las puertas estén completamente cerradas, cuando el sensor detecta una apertura mayor a 1cm el tren no puede avanzar. El piloto automático se encarga de la velocidad del tren en toda la vía, la velocidad máxima es diferente en cada tramo, por lo que, si detecta que se está superando, automáticamente disminuirá la velocidad. Sin embargo, la velocidad máxima total como medida de seguridad es de 65 Km/h, aún si la infraestructura de las vías permitiera más, también, en caso de haber un semáforo en rojo, automáticamente aplica los frenos para evitar colisiones con otro tren. Otra de las tareas del piloto automático es llevar una frecuencia fija para el paso de los trenes por cada estación, idealmente de 180 segundos, aunque por la cantidad de gente en el Metro de la Ciudad de México, esto depende más de la demanda que haya en ese momento.

Conducción Manual (CM): En éste modo, el conductor tiene más atribuciones de entrada, el conductor ahora tiene control total sobre la velocidad del tren, la apertura

y cierre de las puertas lo puede realizar cuando sea y puede avanzar en reversa cuando sea necesario. No suele ser usado, solo en situaciones extremas de avería del tren y maniobras complicadas en caso de percances en alguna vía. Es obvio que casi nunca se utilice, debido a que, como no respeta ningún sistema de seguridad, podría existir alguna colisión con otros trenes. Aunque a veces se provocan colisiones para remolcar trenes en caso de descompostura total. Así que, para esos casos, el modo CM es más que necesario. (Barrios, 2018) Dentro del modo CM, es requerido respetar las señales que se encuentran a lo largo de la línea.

2.2 Cabinas de Simulación

Actualmente a nivel mundial las cabinas de simulación van creciendo para brindar una opción para capacitar nuevos conductores; en diversos países las cabinas se elaboran conforme a los modelos de trenes que utilizan en las líneas. La empresa CAF se especializa en la construcción y elaboración de nuevos modelos de trenes para las diferentes líneas en el mundo, la empresa también ofrece cabinas de simulación dependiendo de los modelos adquiridos.

En la Ciudad de México existen dos tipos de simuladores:

- Cabina Metro línea 12 elaborado por la empresa CAF.
- Cabina Metro SIMCAB (Simuladores de Cabina) elaborado por la STC.

2.2.1 Cabina de Simulación “SIMCAB”

La cabina de simulación (Figura 2.8), está construida mayormente (su contorno) de fibra de vidrio, en su interior se encuentra una réplica exacta de una cabina de los trenes que aún están activos y además tiene una similitud con el entorno de los controles de otras cabinas.

Figura 2.8 SIMCAB

Panel de control. Dentro de la cabina se encuentra una réplica de los controles que la conforman (Figura 2.9).

Figura 2.9 Panel de control del tren

Las estructuras internas de las palancas de velocidades y frenos (Figura 2.10), puertas y tracción (Figura 2.11), están conformadas por un seleccionador de señales.

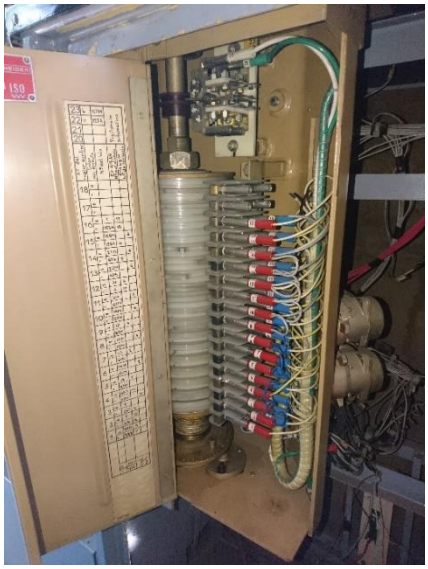


Figura 2.10 Conexión de sistema de velocidades y frenos

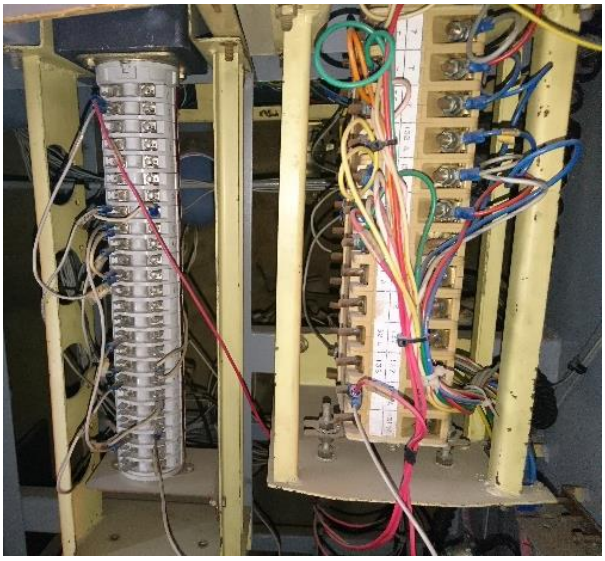


Figura 2.11 Conexión de sistema seleccionador de puertas

2.3 Simuladores de trenes

En la actualidad el software de simulación en el área ferroviaria se ha desarrollado en el tema de juegos y pocos de ellos se pueden utilizar para entender el manejo de un tren y conocer sus componentes principales en los controles. Por ejemplo, los simuladores que se pueden emplear para la enseñanza del manejo ferroviario son los siguientes:

- **Rail Works**
- **Microsoft Train Simulator**
- **BVE Trainsim**

- **OpenBVE**
- **Rail Simulator**

De los programas mencionados, se ha elegido trabajar con el software de licencia libre OpenBVE, que se caracteriza por su entorno amigable y con mayores opciones de modificación en el área visual.

2.3.1 OpenBVE

Datos del Software

El simulador de trenes Open BVE (Figura 2.12), es de licencia libre (dominio público), con la versión 1.5.1.6; cuenta con varias opciones, de las cuales las más destacadas son:

- Objetos móviles.
- Diferentes vistas.
- Descarrilamientos de tren.
- Colisiones.
- Resistencia al viento.
- Frecuencia entre trenes.
- Entre otras.

Este programa usa como base el OpenGL, el cual le permite producir gráficos 2D y 3D a partir de geometrías simples. Las rutas son del formato RW y CSV del cual este último es el más actual.

Figura 2.12 Logo de simulador OpenBVE

Este programa también cuenta con varias herramientas de visualización y edición, las cual son:

- **Track viewer**: Su función es la visualización de las rutas construidas en formato CSV o RW.

- **Structure viewer**: Para visualizar los objetos en formatos CSV y B3D como también conversión de formato X.
- **Motor editor**: Su función es de editar los sonidos de los trenes.
- **Mirror**: Rotar objetos.
- **Object converter**: Es para la conversión de objetos en formato CSV a B3D o viceversa.
- **CSV-X converter**: Transforma un grupo de objetos CSV al formato X.
- **Gauge editor**: Su función es modificar el ancho de los paneles de los trenes.

2.3.2 Instalación

El programa OpenBVE se puede descargar de forma gratuita desde su página oficial[1]. En ella se encontrará la versión más actualizada y la forma en la que se debe instalar, además se debe descargar Net framework, es un archivo complementario del simulador, que se necesita para poder trabajar de forma correcta, ya que sin él, al iniciar una nueva partida, no iniciará la simulación y se cerrará inmediatamente el programa, sin dar un mensaje de error.

2.3.3 Interfaz

Nueva Partida: En la figura 2.13 se observa que se pueden seleccionar diferentes rutas y trenes, que pueden cargarse en la simulación, además, muestra información de dichas rutas y trenes dependiendo de los datos que se tenga almacenados de estos, así como cambiar el modo de conducción que se divide en tres diferentes opciones, las cuales son:

- *Modo Árcade.* – Introduce a la simulación un sistema de puntuación que se genera dependiendo de la habilidad que tenga el conductor, que se caracteriza de los tiempos de llegada, salida, acoplamiento, límites de velocidad, como otras.

- *Modo Normal.* – El sistema de puntuación no aparece en este modo, se da más libertad al conductor, solo mostrando la hora de llegada, salida, distancia de acoplamiento entre otros mensajes que se generan en el trayecto.

- *Modo Experto.* – Tanto el sistema de puntuación como los mensajes de arribo, salida, distancia, etc.; no aparecen en este modo, ya que el conductor aun con más libertad, debe de llevar un control de tiempo propio y calcular la distancia de acoplamiento, como también la opción del pilotaje automático esta desactivada.

[1] https://openbve-project.net/

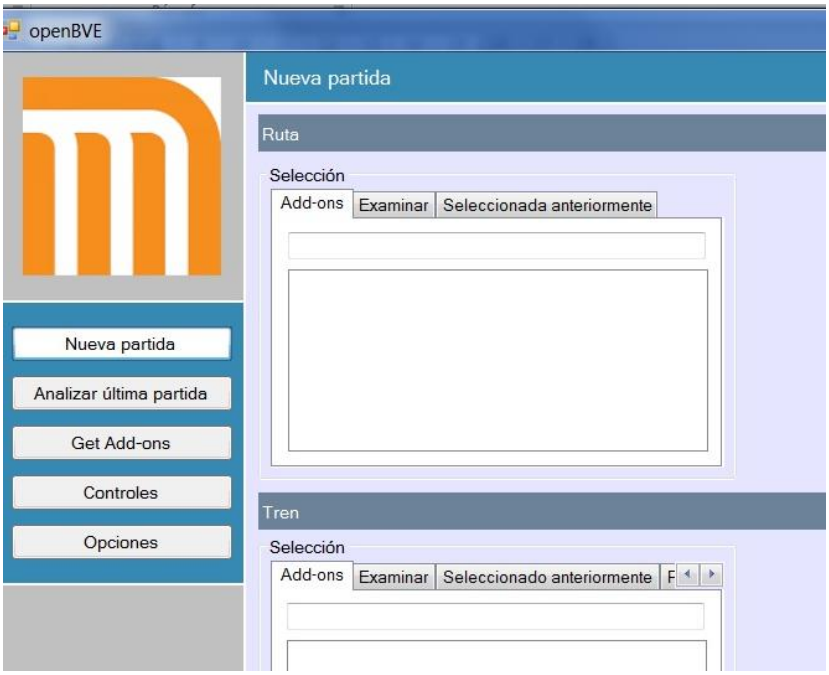


Figura 2.13 Ventana de Nueva partida

Analizar última partida: Como se puede observar en la figura 2.14, en esta ventana se muestra un reporte final al acabar la simulación, mostrando una calificación del rendimiento que tuvo el conductor, tomando en cuenta los factores que se mencionaron en el modo árcade de la ventana de Nueva partida, el reporte se genera sin importar el modo que se haya seleccionado, en éste veremos la ruta empleada, el tren seleccionado, la fecha y la hora. Más abajo se encuentra el porcentaje de conducción y a un lado un registro que mostrara si el conductor llega a la estación sin pasar el punto de paro de manera satisfactoria o con metros de más, como también horas de llegada, salidas, si respeto los límites de velocidad, etc. Al final podemos exportar el reporte a un archivo de texto para futuras consultas.

Figura 2.14 Ventana Analizar última partida

Get add-ons: En la figura 2.15 muestra un buscador de archivos, el cual nos servirá para adjuntar o instalar rutas junto con trenes, añadiendo nuevos archivos o removiéndolos.

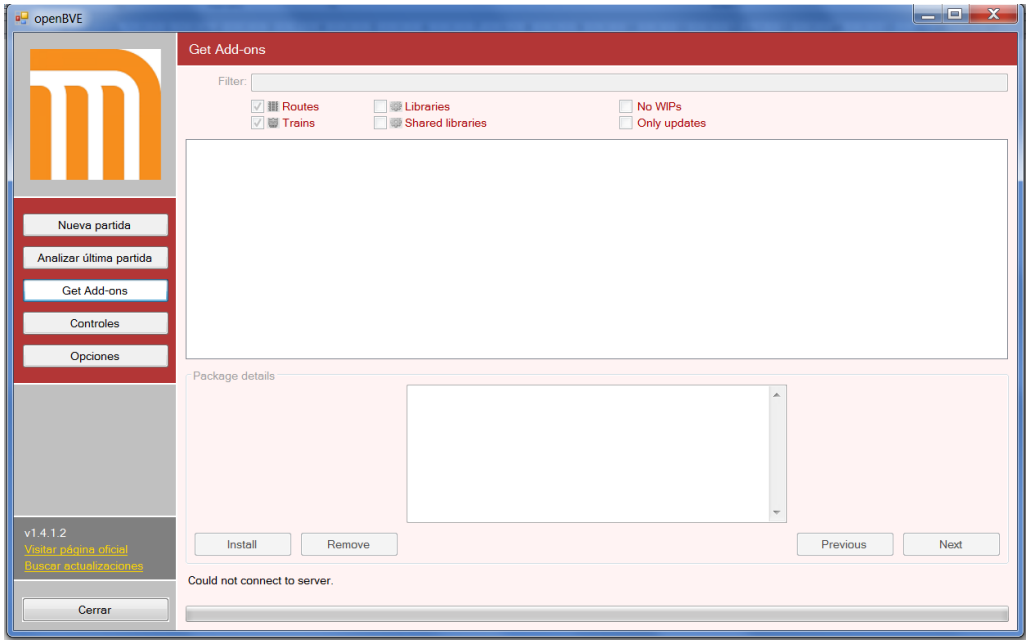


Figura 2.15 Ventana Get add-ons

Controles: En esta ventana (Figura 2.16), nos mostrara los controles que contiene el simulador, con opciones de añadir nuevos o modificar los ya existentes, también permite la visualización de los joysticks conectados.

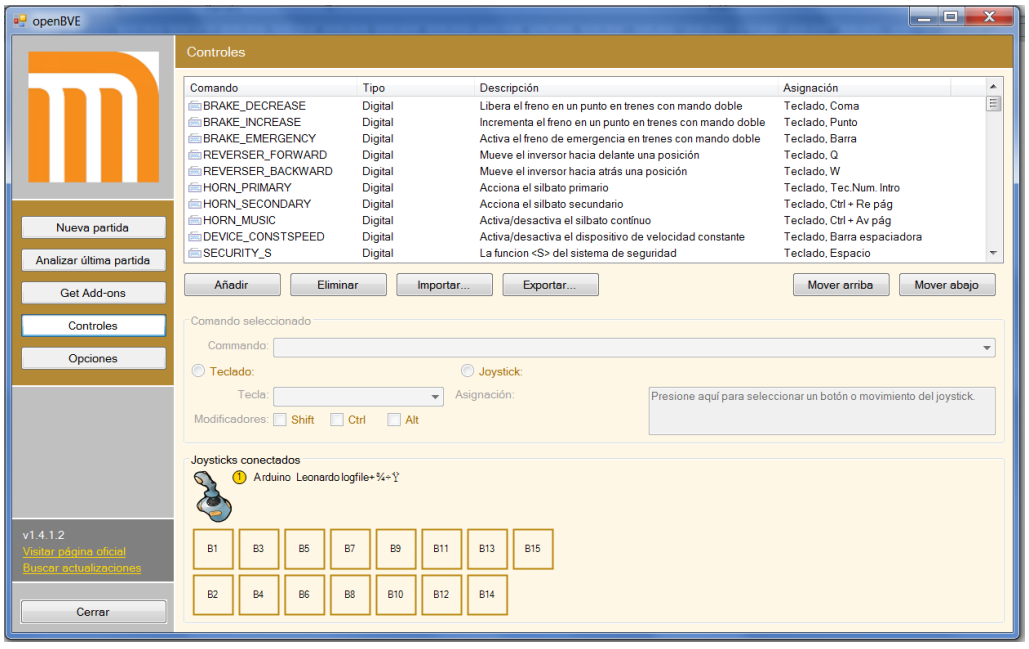


Figura 2.16 Ventana de Controles

Opciones: En la última ventana disponible de la interfaz de OpenBVE (Figura 2.17), nos muestra las siguientes opciones:

Cambio de idioma. – Muestra una lista de los idiomas disponibles con el que cuenta la interfaz.

Modo de pantalla. – Da la opción de mostrar la simulación de forma completa en toda la pantalla o como una ventana en el escritorio de la computadora.

Ventana. – En esta se pueden modificar las dimensiones de la ventana de la simulación solo cuando se haya seleccionado el modo pantalla.

Pantalla completa. – Al igual que la opción de ventana, se modifican las dimensiones, pero también la profundidad de bits de colores.

Filtrado. – Se modifican las opciones gráficas de la simulación, cambiando los filtros y logrando una mejora visual.

Efectos. – Da la opción de modificar la distancia con la que la simulación va actualizando los datos de la ruta, mostrando los objetos y paisajes que la conforman.

Sonido. – Nivel de volumen.

Controles. – Da la opción de activar o desactivar los joysticks que se encuentran conectados en el ordenador.

Detalles de la simulación. – Activa o desactiva los efectos que se puedan generar en el transcurso de la simulación, como también activar la caja negra que guardara los reportes de conducción.

Verbosity. – Da la opción de mostrar los mensajes de alerta o errores al iniciar la simulación que son generados cuando un archivo no es encontrado en dicha ruta.

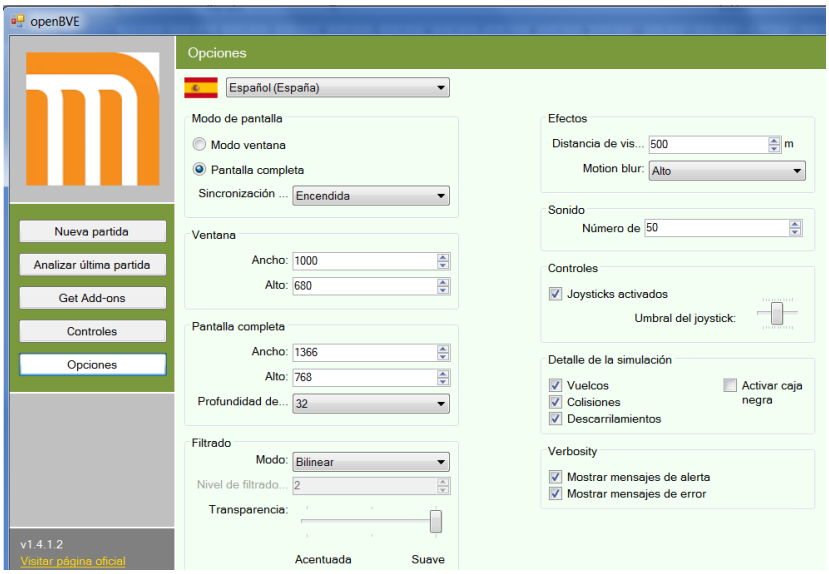


Figura 2.17 Ventana de Opciones

Líneas y trenes STCM

A continuación, se muestra una tabla de las rutas y una lista de los trenes disponibles de la ciudad de México para el simulador OpenBVE, las cuales, fueron modificadas y mejoradas visualmente para dar una mejor presentación en la capacitación.

Tabla 2-1. Líneas de trenes

	Línea A (Versión Día, Noche, Lluvia, Atardecer)
	Línea 2 (Versión Día, Noche, Lluvia)
	Línea 3 (Versión Día)
	Línea 4 (Versión Noche)
	Línea 9 (Línea subterránea)

Trenes: A continuación, se presentarán una lista de los trenes que fueron utilizados para la simulación, acompañados con una descripción de cada uno.

- Metro de la Ciudad de México modelo CAF FM 95A (Figura 2.18). Este tren fue construido por las empresas Bombardier, Concarril y CAF, y circula actualmente en la línea "A" Pantitlán - La Paz del metro de la Ciudad de México. Es el segundo tren de rodadura férrea en entrar en servicio. Fue puesto en circulación en el año 1995, su alimentación es de 750 V. Las series motrices son FM 0041 a FM 0066.

Figura 2.18 Tren CAF FM 95A

- Metro de la Ciudad de México Modelo CAF FE 07 (Figura 2.19). Este tren fue construido por la empresa española CAF y entro en servicio en el año 2007 en la línea A "Pantitlán - La Paz" del Metro de la Ciudad de México. Cuenta con 9 coches de los cuales 4 son motores, 3 remolques y 2 con cabina de conducción.

Figura 2.19 Tren CAF FE 07

- Metro de la Ciudad de México Modelo CAF FE–10 (Figura 2.20). Este tren fue construido por la empresa española CAF y circula por la línea 12 "Tláhuac - Mixcoac" del Metro de la Ciudad de México. Cuenta con 7 coches de los cuales 5 son motores, y 2 remolques con cabina de conducción.

Figura 2.20 Tren CAF FE–10

- Metro de la Ciudad de México Modelo NM 79 (Figura 2.21). El NM 79 es el Tercer modelo con más antigüedad de todo el parque vehicular del STC, circula actualmente por la línea 3 (indios verdes - Universidad). Este tren fue construido por la empresa mexicana Con carril, es el primer tren a cuál le agregaron rejillas y ventiladores dado el calor que sentían los pasajeros, también fueron los primeros en llegar ya con el pilotaje automático instalado y en tener los interiores diferentes, como en configuración de asientos y acabados en amarillo crema y asientos en verde limón. Fue puesto en servicio en 1979 y circula en las líneas 1, 7, 8 y 9.

Figura 2.21 Tren NM 79

- Metro de la Ciudad de México NM 02 (Figura 2.22). Este tren circula por la línea 2 (Cuatro Caminos - Taxqueña) y por la línea 7 (El Rosario - Barranca

del Muerto) del Metro de la Ciudad de México. Fue construido por la empresa Bombardier, con asistencia de CAF, las unidades del tren están formadas por 9 coches (6 coches motores y 3 coches remolques), que también pueden trabajar en composición de 6 coches (4 coches motores y 2 remolques).

Figura 2.22 Tren NM 02

Comandos CSV

Los archivos CSV (comma-separated values), son un tipo de documento de formato abierto, que sirven para presentar información en forma de tablas, las cuales contienen un juego de caracteres en concreto y se pueden visualizar como una hoja de cálculo (Figuras 2.23 y 2.24). Este tipo de archivos se usan en el simulador OpenBVE para la creación o colocación de objetos que se sitúan en rutas o en los trenes, también establecen los puntos de acoplamiento usado por el punto normal de paro, o los horarios de salidas y llegadas. Usando los caracteres establecidos, se le indica el lugar y orientación donde debe ser colocada la imagen u objeto que se agrega o modifica.

Dichos archivos al igual que las imágenes se guardan en carpetas para tener un orden y el programa los pueda cargar de forma más eficaz, como también para su futura modificación o visualización. Al término de cada creación o modificación, se pueden visualizar los cambios, ya sea dentro del recorrido en la simulación o usando las herramientas Track viewer, Structure viewer, Object viewer; ya mencionados anteriormente.

	A	B	C	D	E	F
1	;Pared					
2	CreateMeshBuilder					
3	AddVertex	10.25	1.2	25		
4	AddVertex	10.25	10.5	25		
5	AddVertex	10.25	10.5	0		
6	AddVertex	10.25	1.2	0		
7	AddFace	0	1	2	3	
8	AddFace	3	2	1	0	
9	GenerateNormals					
10	LoadTexture	muro4.png				
11	SetTextureC	0	0	1		
12	SetTextureC	1	0	0		
13	SetTextureC	2	4	0		
14	SetTextureC	3	4	1		
15	Setcolor	150	150	150		
16						
17	;Techo					
18	CreateMeshBuilder					
19	AddVertex	-1.75	10	2.5		
20	AddVertex	-1.75	10	0		
21	AddVertex	10.25	10	0		
22	AddVertex	10.25	10	2.5		
23	AddFace	0	1	2	3	
24	AddFace	3	2	1	0	

Figura 2.23 Código CSV

18	With Structure
19	.Rail(0) Linea A\Vias\Via 1.csv
20	.Rail(1) Linea A\Vias\Via 2.csv
21	.Rail(2) Linea A\Vias\Via 3.csv
22	.Rail(4) Linea A\Vias\Calzada derecha.csv
23	.Rail(5) Linea A\Vias\Calzada izquierda.csv
24	.Rail(6) Linea A\Vias\Carril metrobus.csv
25	.Rail(7) Linea A\Vias\Calle derecha.csv
26	.Rail(8) Linea A\Vias\Calle izquierda.csv
27	.Rail(9) Linea A\Vias\A2 CD.csv
28	.Rail(10) Linea A\Vias\A2 CI.csv
29	.Rail(11) Linea A\Vias\A2 VD.csv
30	.Rail(12) Linea A\Vias\A2 VI.csv
31	.Rail(14) Linea A\Vias\Via 4.csv
32	
33	.WallR(1) Linea A\Wall\Tunel derecha.csv
34	.WallL(1) Linea A\Wall\Tunel izquierda.csv
35	.WallR(2) Linea A\Wall\Calzada derecha.csv
36	.WallL(2) Linea A\Wall\Calzada izquierda.csv
37	.WallR(3) Linea A\Wall\Final derecho.csv
38	.WallL(3) Linea A\Wall\Final izquierdo.csv
39	.WallR(4) Linea A\Wall\Calle derecha.csv
40	.WallL(4) Linea A\Wall\Calle izquierda.csv

Figura 2.24 Código CSV

Integración de señalización

El programa OpenBVE de forma automática establece la señalización correspondiente que se carga desde sus archivos de origen (Figura 2.25), del cual, se pueden modificar desde los archivos CSV que se encuentran en las rutas que se utilizaran, estos pueden ser desde semáforos, puntos normal de paro, usados mayormente en el estado de pilotaje automático, o solo para indicar en donde se encuentran y señalar si el conductor acoplo de forma exitosa o con cuantos metros pasa de este en el estado de conducción manual, es importante mencionar que los semáforos funcionan conforme existan trenes que trabajan de forma automática para dar un realismo más exacto dentro de la simulación.

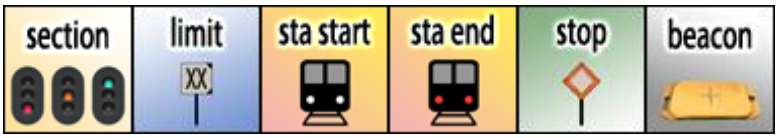


Figura 2.25 Iconos de señalización

Al igual que en el programa, la parte de la señalización también debe de agregarse a los trenes, ya que en un vagón en específico llevará el sistema de pilotaje automático, los ATS (Automatic Train Stop) y ATC (Automatic Train Control), estos dos últimos se encargaran de que el tren al acoplar en la estación o en el recorrido, den la señal a los semáforos que se encuentran en la línea para cambiar, así se dará aviso a los trenes que estén atrás para advertir si hay otro más delante de ellos para bajar la velocidad o hacer paro total de la marcha, después de que uno de los

tramos de la línea este libre, los semáforos cambiaran y señalaran a los otros trenes que podrán avanzar sin problema hasta llegar a la siguiente estación.

Es importante mencionar que el tema de señalización es fundamental, ya que, si no se integra, tanto en la línea como en el tren, se podrían generar colisiones en el recorrido, dado que los semáforos no recibirían las señales para cambiar de color, y no indicarían que no hay trenes más adelante.

2.4 Interfaz hardware

Tarjetas Opto-Acopladoras

El simulador cuenta con tres tarjetas de acoplamiento de señales, que permiten el encendido y apagado de las lámparas, así como activar las palancas, etc. que en su conjunto sirven para la conducción de un tren.

Tarjeta 1. En la figura 2.26 se muestra la tarjeta opto-acopladora de lámparas, esta maneja todas las lámparas, se conforman de 24 señales de entrada, 24 de salida y cuatro de alimentación.

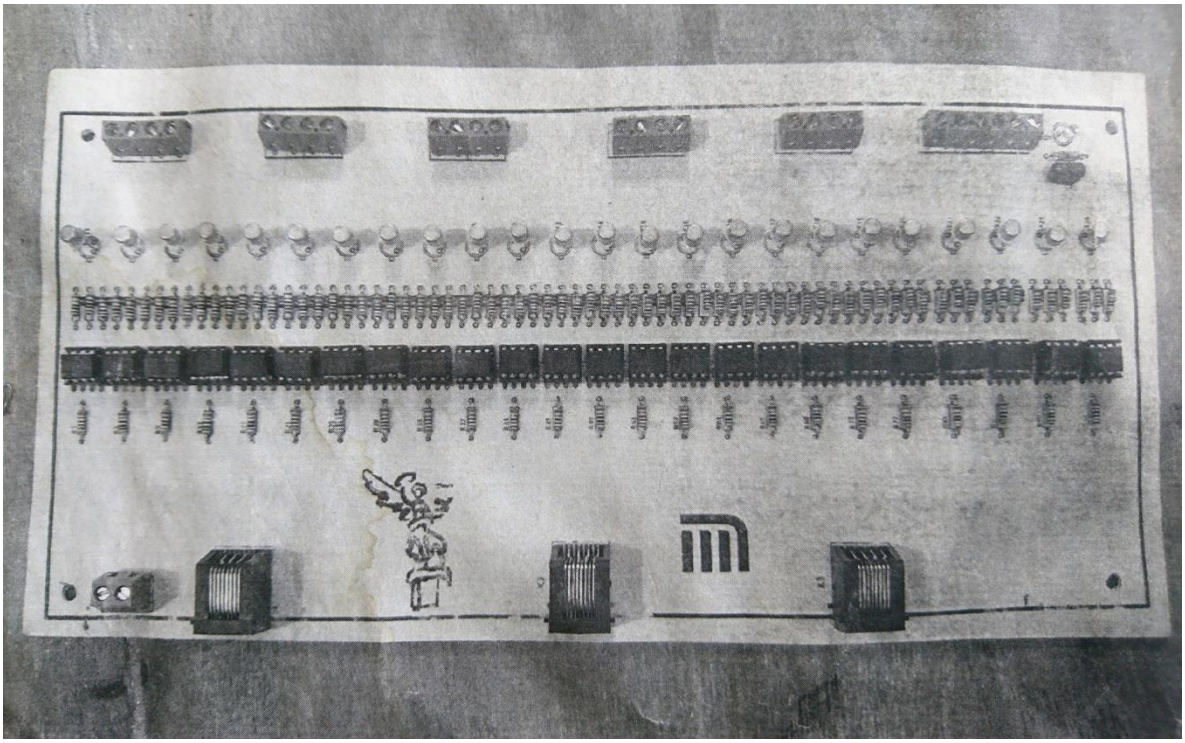


Figura 2.26 Tarjeta Opto-acopladora de lámparas

La conexión de las entradas es en la parte baja de la tarjeta y de izquierda a derecha en los tres conectores RJ45 (lo cual agrupa las señales en grupos de 8 en cada uno) y la primera bornera está destinada a GND de las señales de digitales o de 5 volts. Las señales de salida se conectan en la parte de arriba en el mismo sentido

de las entradas comenzando en la primera y dejando la última para la alimentación de 9 volts que es la alimentación a las lámparas.

Tarjeta 2. La tarjeta de opto-acoplamiento 2 se muestra en la figura 2.27, está tarjeta, en su mayoría, se encarga de activar algunas de las palancas de la cabina y solo controla dos lámparas. Esta consta de 20 señales de entrada, 20 señales de salida y 4 señales de alimentación, el orden de conexión de las señales es de izquierda a derecha, las entradas son las de arriba y las salidas las de abajo.

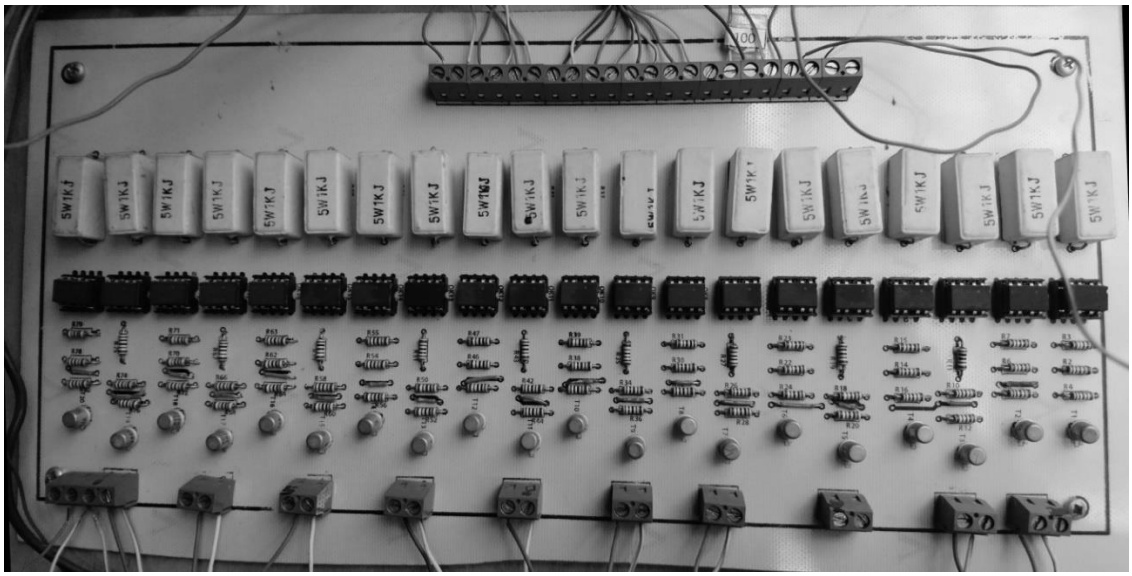


Figura 2.27 Tarjeta Opto-acopladora de palancas

Tarjeta 3. La tarjeta 3 opto-acopladora, es igual que la anterior y está destinada solo al activar de algunas de las palancas que conforman a la cabina. Las tarjetas ya montadas y conectadas se pueden ver en la figura 2.28.

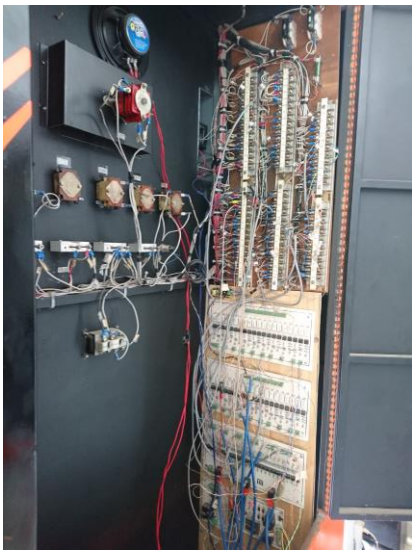


Figura 2.28 Conexiones de tarjetas Opto-acopladoras

Las palancas utilizadas en el simulador se describen en la tabla 2-2.

Las lámparas utilizadas en el simulador se describen en la tabla 2-3.

Tabla 2-2 Controles de cabina

Nombre	Descripción	Conexión tarjetas Opto-acopladoras
Arillo de hombre muerto	Auxiliar para sistema de seguridad	18
T1	Tracción 1	3
T2	Tracción 2	19
T3	Tracción 3	20
T4	Tracción 4	21
F1	Intensidad de Frenado	17
F2	Intensidad de Frenado	16
F3	Intensidad de Frenado	15
F4	Intensidad de Frenado	14
F5	Intensidad de Frenado	13
F6	Intensidad de Frenado	67P
FU	Intensidad de Frenado	55M
Llave C	Auxiliar para mandos de conducción	A3/69
Llave T1	Selección de puertas	32/37
Botón V	Anuncio de partida	59D
Botón FD	Cierre de puertas/seguridad	36 A
Llave OA	Apertura/Anulación puertas	138P/133 A
Llave VR	Marcha Atrás/Adelante	10/11
Bocina	Silbato primario	130V
Timbre	Silbato secundario	100
Auxiliar para definir modo de conducción	Conducción CM	126D

Tabla 2-3 Lámparas o avisos luminosos de errores

Lámparas	Conexión tarjetas Opto-acopladoras
Todos los carros desbloqueados	45 L
1er. Carro motor inactivo	AA
2do. Carro motor inactivo	BA
3er. Carro motor inactivo	CA
4to. Carro motor inactivo	DA
5to. Carro motor inactivo	EA
6to. Carro motor inactivo	FA
Programa no alimentado	77 (355L)
Pilotaje automático no disponible	70 (70L)

Todos los carros bloqueados	44L
No carga batería	38L
No autorización de recuperar	
Punto maniobra	40L
Carro motor fuera	41L
Corriente cortada	125S
Freno de mano aplicado	43L
Mando continuo fuera	39L
No desbloqueado	46L
ADL en falla (Lámpara VAI)	125V
Convertidor en falla	26
Compresor en falla	7ª
Lámpara de ocupación de T1	
Lámpara de testigo T2	
Lámpara de apertura preparada (LOP)	
Lámpara de preparación de apertura	
Lámpara de mantenimiento de cierra	133ª

Comunicación Cabina-Servidor y Software de simulación

El primer software de simulación que se implementó consistía en un video mostrando un recorrido en una de las líneas del sistema del Metro; empleando algoritmos de tiempo para la sincronización con los controles de la cabina. Debido a las limitaciones del equipo de cómputo el video mostraba una baja calidad de imagen, generando una dificultad en la visualización de señalización, fallas o incidentes. Por lo tanto, este será cambiado por el software OpenBVE, el cual proporciona mejoras visuales.

Para la comunicación original se había implementado una tarjeta Nacional Instruments modelo USB-6509 (Figura 2.29).

Figura 2.29 Tarjeta Nacional Instruments

Esta es una tarjeta de adquisición de datos, lee las señales de la cabina y las envía de forma serial a través del puerto USB. Esta tarjeta tiene un costo alrededor de los $300 dólares por lo cual será remplazada por un circuito que se ha diseñado con un bajo costo, el mismo que nos permitirá igualar la comunicación Cabina-PC. Sin embargo, para que se logre comunicar con el software OpenBVE que se utilizo es necesario que el circuito también haga la función de Gamepad (Control), el diseño del circuito y la programación se encuentran en sus respectivos capítulos.

Como se ha mencionado anteriormente, la comunicación se logró por medio del puerto USB de la tarjeta creada al ordenador. Para que la tarjeta funcione sin necesidad de una alimentación externa se aprovecha la alimentación que suministra el puerto USB.

2.5 Arduino

Arduino es una plataforma de prototipos electrónica de código abierto (open-source) basada en hardware y software flexibles y fáciles de usar (Figura 2.30). Está pensado para artistas, diseñadores, como hobby y para cualquiera interesado en crear objetos o entornos interactivos.

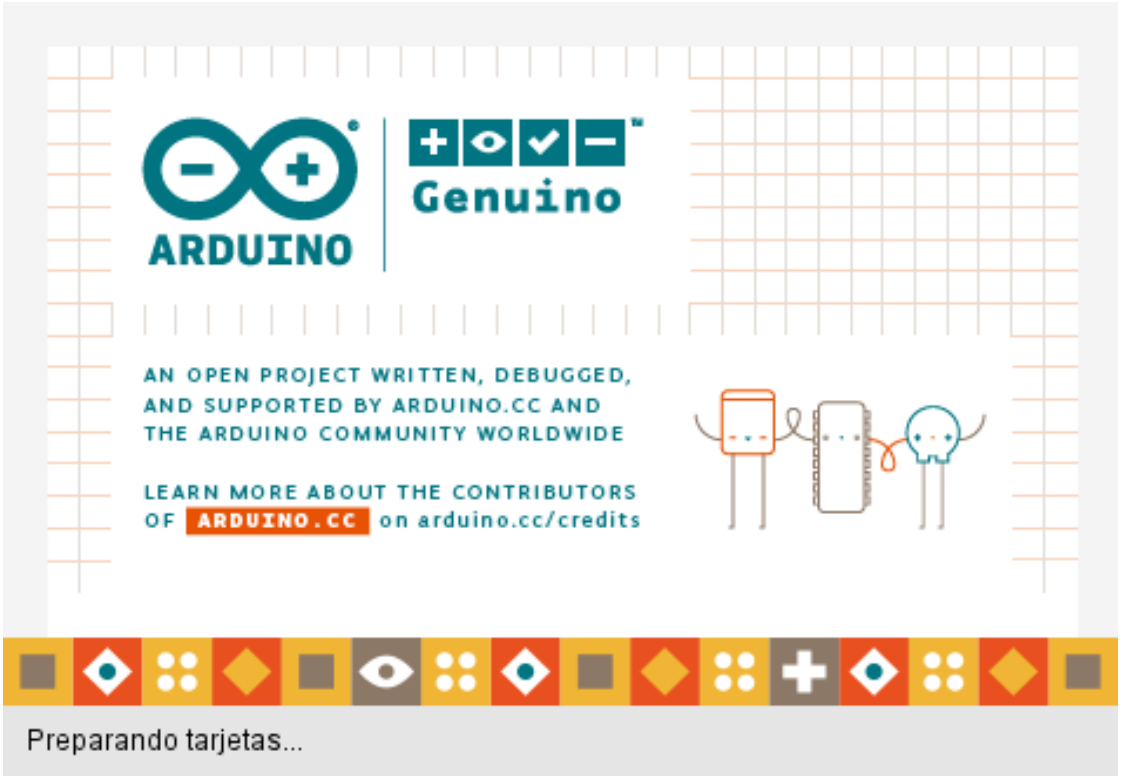


Figura 2.30 Programa Arduino

2.5.1 Entorno de programación Arduino

Para la codificación de programas se cuenta con un entorno de programación (Figura 2.31), que también permite mediante comunicación serial, la interacción entre la tarjeta y una computadora personal, con propósitos de programación o intercambio de información.

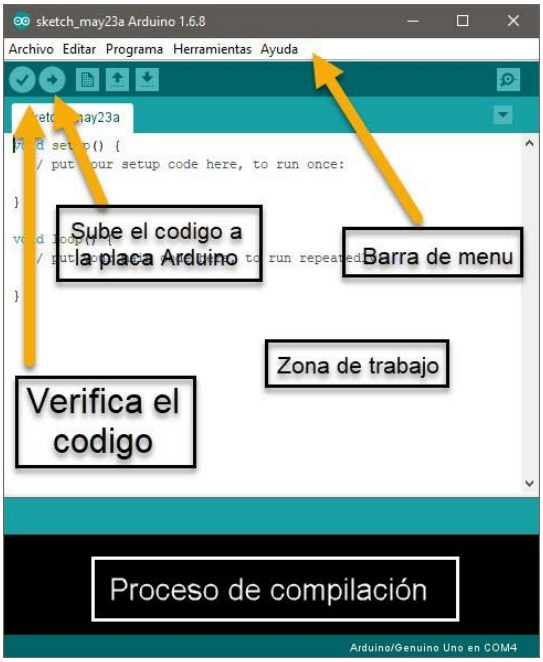


Figura 2.31 Entorno de Arduino

2.5.2 Estructura principal y variables

Al empezar un programa podemos encontrar lo siguiente (Figura 2.32):

- *Void setup*: La función de configuración (setup) debe contener la declaración de las variables. Es la primera función a ejecutar en el programa, se ejecuta solo una vez, y se utiliza para configurar o inicializar pinMode (modo de trabajo de las E/S), configuración de la comunicación en serie y otras.

- *Void loop:* La función bucle (loop) contiene el código que se ejecutara continuamente (lectura de entradas, activación de salidas, etc) Esta función es el núcleo de todos los programas de Arduino y la que realiza la mayor parte del trabajo. (playground.arduino.cc, s.f.)

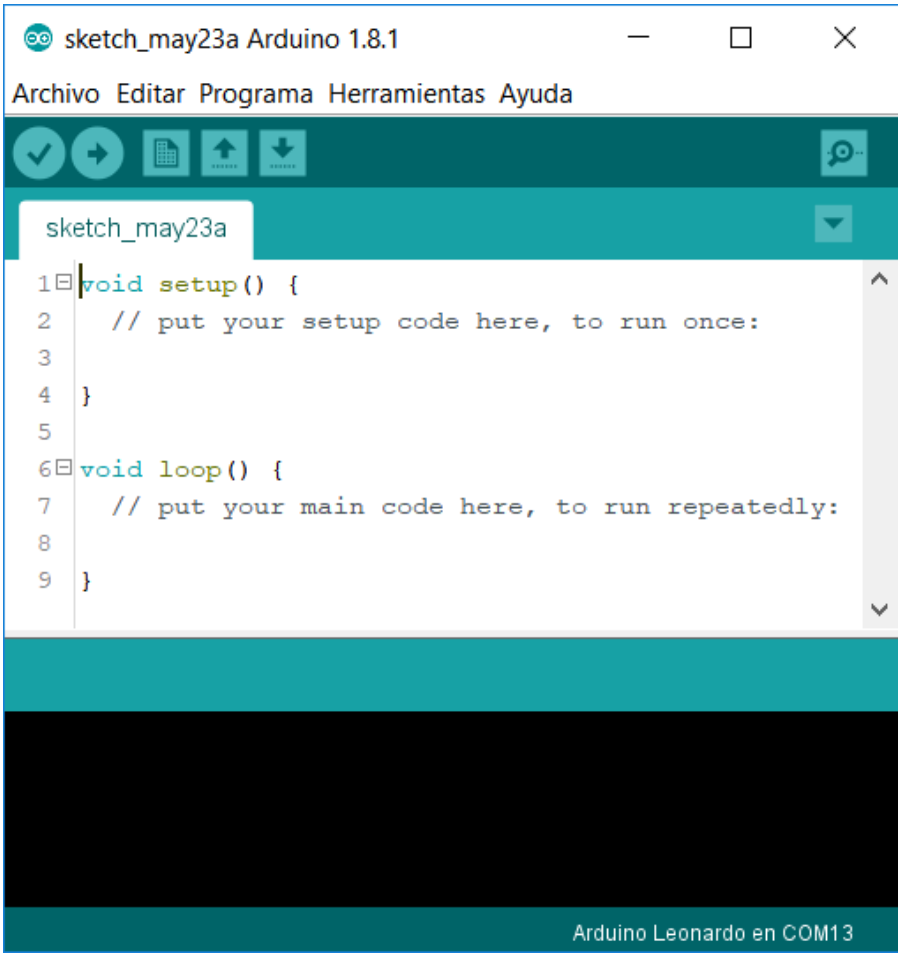


Figura 2.32 Estructura del entorno de Arduino

Variables.

Una variable es un lugar donde se puede almacenar un dato, una variable se conforma por un nombre, un valor y un tipo, existen distintos tipos de variables, las cuales se enumeran a continuación.

- Variables constantes:
    - HIGH: Corresponde a un 1 lógico.
    - LOW: Corresponde a un 0 lógico.
    - True y false: Corresponde a un 1 y a un 0 lógico respectivamente. True y false son creados para una lectura más accesible.
    - INPUT/OUTPUT. – Son entradas y salidas respectivamente, se usan para configurar los pines de la tarjeta.

En la tabla 2-4 encontramos otros tipos de variables junto con su descripción y ejemplo:

Tabla 2-4 Variables

TIPO	DESCRIPCIÓN	EJEMPLO
Void	void se usa en declaraciones de funciones. Indica que se espera que la función no devuelva información	
Int	(Integer=entero). Un número entero entre 32,767 y -32,768 codificado en dos octetos (16 bits)	int testVariable = 28927;
Long	Un entero comprendido entre 2, 147, 483,647 y – 2, 147, 483,648 y codificado en 32 bits (equivalente a 4 bytes/octetos).	long testVariable = 67876;
Float	Un número real (con decimales) almacenado en 4 bytes (es decir 32 bits) y comprendido entre 3.4028325E+38 y -3.4028325E+38	float testVariable = 3.56;
unsigned int	Un número natural (entero positivo) almacenado en 16 bits (2 bytes) y comprendido entre 0 y 65,545	unsigned int testVariable = 38948;
unsigned long	Un número natural (entero positivo) almacenado en 32 bits (4 bytes) y comprendido entre 0 y 4,294,967,296	unsigned long testVariable = 657456;
Word	Lo mismo que unsigned int	word testVariable = 51000;
Boolean	Una variable booleana que puede tener solamente dos valores: true (verdadero) o false	boolean testVariable = true;
Char	Un carácter ASCII almacenado en 8 bits (un byte). Esto permite almacenar caracteres como valores numéricos (su código ASCII asociado). El código ASCII para el carácter ‘a’ es 97, si le añadimos 3 obtendríamos el código ASCII del carácter ‘d’	char testVariable = ‘a’; char testvariable = 97;
unsigned char	Este tipo de datos es idéntico al tipo byte explicado arriba. Se utiliza para codificar números de 0 hasta 255. Ocupa 1 byte de memoria.	unsigned char testUnCh = 36;

2.5.3 Instrucciones y funciones Arduino

Para elaborar un programa usamos diferentes instrucciones estas nos ayudan a poder controlar las entradas y salidas, también contamos con funciones que nos facilitan la programación, existen distintos tipos: de tiempo, para realizar operaciones matemáticas, de comunicación, para usar entradas o salidas, etc. En cuanto a operadores tenemos aritméticos, de comparación, booleanos etc.

En las tablas 2-5 y 2-6 se encuentran algunas funciones e instrucciones.

Tabla 2-5 Funciones

Función	Descripción	Sintaxis
digitalRead()	Lee el valor de un pin digital este puede ser HIGH o LOW	digitalRead(pin)
digitalWrite()	Escribe el valor de un pin digital este puede ser HIGH o LOW	digitalWrite(pin, valor)
pinMode()	Configura el pin como una entrada (INPUT) o una salida(OUTPUT)	pinMode(pin, modo)
analogRead()	Lee el valor de un pin analógico.	analogRead (pin)
delay()	Pausa el programa por una cantidad de tiempo (en milisegundos)	Delay(microsegundos)
millis()	Nos da la cantidad de milisegundos desde que la placa Arduino comenzó a ejecutar el programa actual, después de aproximadamente 50 días regresa a cero	time = millis()
Serial.begin()	Inicializa la comunicación serial generalmente a 9600 bits por segundo (baudios)	Serial.begin(velocidad)
Serial.println()	Imprime datos en el puerto serie como texto ASCII legible por humanos seguido de un carácter de retorno de carro (ASCII 13, o '\ r') y	Serial.println(val)

un carácter de nueva línea (ASCII 10 o '\ n')

Tabla 2-6 Instrucciones

Instrucción	Descripción	Sintaxis

If	La instrucción if comprueba una condición y ejecuta la instrucción previa o el conjunto de enunciados si la condición es 'verdadera'	if (condición)
		{
		// declaración (es)
		}

For	La instrucción for se usa para repetir un bloque de instrucciones encerrado entre llaves. Un contador de incremento se usa generalmente para incrementar y terminar el ciclo	for (inicialización; condición; incremento) {
		// declaración (es);
		}

While	Un ciclo while se repetirá continuamente, e infinitamente, hasta que la expresión dentro del paréntesis, () se vuelva falsa.	while (condición) {
		// declaración (es)
		}

switch...case	switch compara el valor de una variable con los valores especificados en las sentencias case. Cuando se encuentra una declaración de caso cuyo valor coincide con el de la variable, se ejecuta el código en esa declaración de caso.	switch (var) {
		case label1:
		// statements
		break;
		case label2:
		// statements
		break;
		default:
		// statements
		}

Break	break se usa para salir de un for, while o do ... while loop, pasando por alto la condición de bucle normal. También se usa para salir de una declaración de switch…case	

(Anónimo, Referencia del lenguage C en Arduino, 2018).

2.5.4 Bibliotecas

LiquidCrystal_I2C

Esta biblioteca nos permite realizar una comunicación con un LCD de 16x2 por dos pines de señales gracias al módulo I2C, para poder crear contenidos visuales con la opción de elaborar un menú, la biblioteca se obtuvo de la siguiente página:

https://www.Arduinolibraries.info/libraries/liquid-crystal-i2-c

Las instrucciones utilizadas son:

- LiquidCrystal_I2C lcd (0x3f, 16, 2). – Instrucción con la cual el Arduino manda información al módulo I2C.
- lcd.init (). – Inicia la comunicación con el LCD.
- lcd.backlight (). – Manda la instrucción de encender la luz de fondo.
- lcd.home (). – Inicia la lectura de texto que será mostrada en el LCD.
- lcd.setCursor (Numero de columna, Numero de fila). – Acomoda el puntero, que indica donde pondrá el texto en la pantalla.
- lcd.print (“texto”). – Manda el texto a la LCD.

Joystick.h

Esta biblioteca nos permite que una computadora reconozca el Arduino como un gamepad, esto gracias al protocolo HID, la biblioteca se obtuvo de la siguiente página: https://github.com/MHeironimus/ArduinoJoystickLibrary.

Las instrucciones utilizadas son:

- Joystick_Joystick(JOYSTICK_DEFAULT_REPORT_ID,
  JOYSTICK_TYPE_GAMEPAD,

  15, 0, // Button Count, Hat Switch Count

  true, true, false, // X and Y and Z Axis

  false, false, false, // Rx, Ry, or Rz

  false, false, // rudder or throttle

  false, false, false); // accelerator, brake, or steering

Con la instrucción anterior se puede configurar el control según nuestras necesidades, se pueden cambiar parámetros como el número de botones, los ejes de los joysticks etc.

- *Joystick.begin*(). – Inicializa la emulación de un gamepad
- *Joystick.pressButton* (Numero de botón). – Pulsa el botón indicado
- *Joystick.releaseButton*(Numero de botón). – Suelta el botón indicado

2.5.5 Arduino Leonardo

El Arduino Leonardo es una tarjeta de desarrollo que utiliza un microcontrolador Atmega32U4 con conectividad USB incluida (Figura 2.33). Utilizando librerías permitirán que la tarjeta Arduino Leonardo emule un teclado, mouse, joystick o bien el protocolo USB-HID.

Figura 2.33 Placa Arduino Leonardo

Este posee 20 pines de entrada y salida digitales (de los cuales 7 pueden ser usados como salidas PWM y 12 como entradas análogas), un cristal oscilador de 16MHz, una conexión microB USB, un Jack de alimentación, un header ICSP y un botón de reinicio (Figura 2.34).

Características:

- Microcontrolador Atmega32u4.
- Voltaje de entrada: 7-12V.
- Voltaje de trabajo: 5V.
- Corriente por pin I/O: 40mA.
- 20 pines digitales I/O.
- 7 canales PWM.
- 12 ADC.
- 16MHz de velocidad de reloj.
- Memoria Flash: 32 KB (ATmega32u4) de los cuales 4 KB son usador por el bootloader.
- Memoria SRAM: 2.5 KB (ATmega32u4).
- Memoria EEPROM: 1KB (ATmega32u4).
- Dimensiones: 68.6 x 53.3mm.

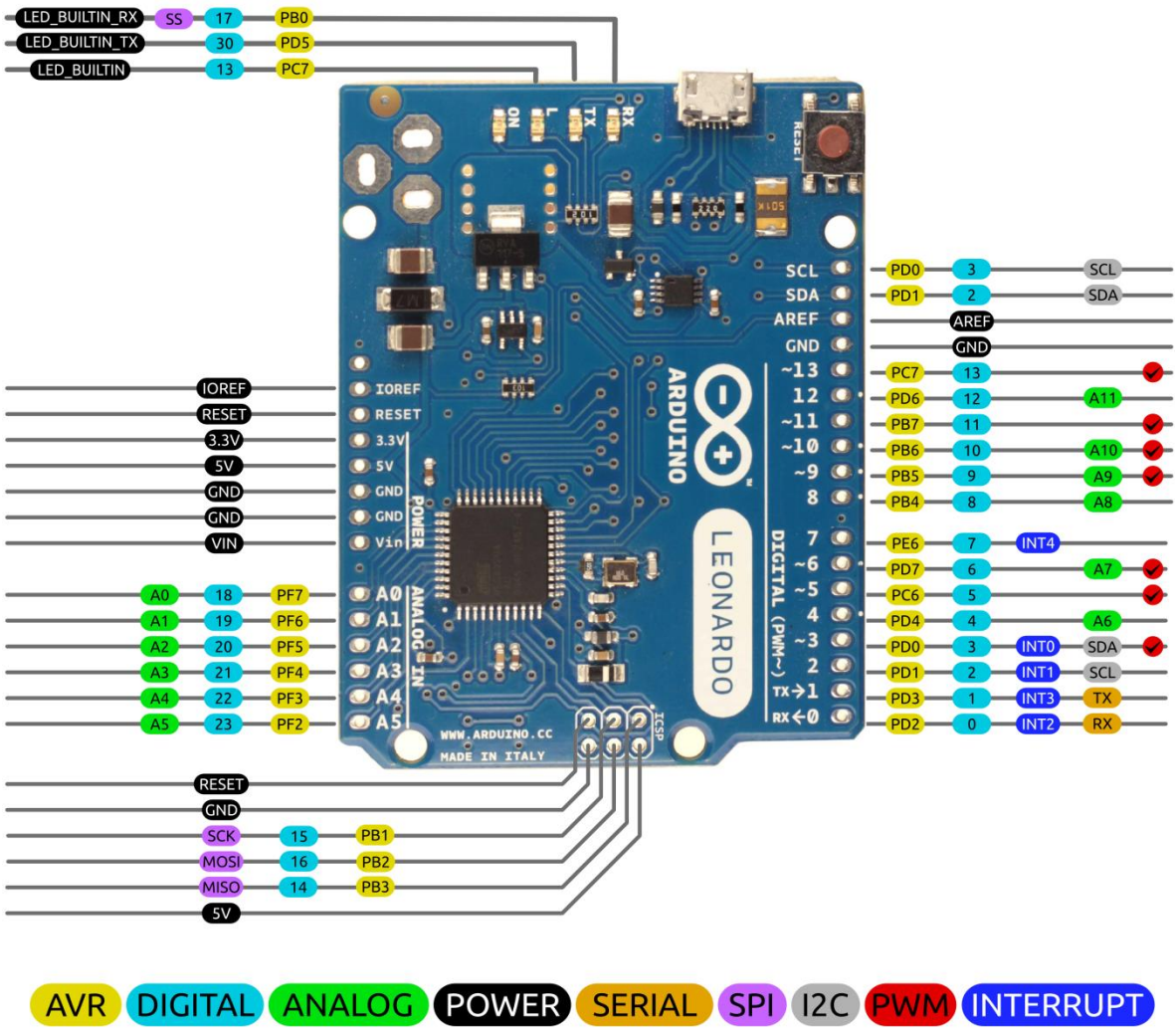


Figura 2.34 Distribución de puertos en Arduino Leonardo

Para el desarrollo del proyecto se ha optado por utilizar esta tarjeta de desarrollo, para lograr la comunicación con la cabina del simulador, utilizando los puertos digitales como también los analógicos; el Arduino trabajara como joystick (control), para así conectarse con el software OpenBVE, además, se conectara con el circuito que se ha creado para reducir el número de puertos empleados, simplificando las conexiones de las señales que manda la cabina para lograr tener una mejor comunicación, que sea más sencilla y compacta para futuras reproducciones.

Gracias a las opciones que brinda esta tarjeta de desarrollo, la adquisición será más económica y fácil de transportar a comparación con la conexión original (tarjeta Nacional Instruments).

2.5.6 Arduino Nano

El Arduino Nano es un microcontrolador completo y fácil de usar basado en el ATmega328P y funciona con un cable USB Mini-B en lugar de un USB Micro-B (Figura 2.35).

Figura 2.35 Placa Arduino Nano

El Arduino Nano puede alimentarse a través de la conexión USB Mini-B, una fuente de alimentación externa no regulada de 6-20V (pin 30) o una fuente de alimentación externa regulada de 5V (pin 27).

Cada una de las 14 terminales digitales en el Arduino Nano se puede usar como entrada o salida, usando las funciones pinMode (), digitalWrite () y digitalRead (). Operan a 5 voltios. Cada pin puede proporcionar o recibir un máximo de 40 mA y tiene una resistencia interna de pull-up (desconectada por defecto) de 20-50 KOhms. Además, algunos pines tienen funciones especializadas (Figura 2.36):

- *Comunicación serie:* Pines 0 (RX) y 1 (TX). Se usa para recibir (RX) y transmitir (TX) datos de manera serial TTL. Estos pines están conectados a los pines correspondientes del chip serie FTDI USB a TTL.

- *Interrupciones externas*: Pines 2 y 3. Se pueden configurar para activar una interrupción en un valor bajo, un flanco ascendente o descendente, o un cambio en el valor.

- *PWM*: Pines 3, 5, 6, 9, 10 y 11. Proporcionan salida PWM de 8 bits mediante la función analogWrite ().

- *Protocolo de comunicación SPI:* Pines 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).

- *LED:* Pin 13. Hay un LED integrado conectado al pin digital 13. Cuando el pin tiene un valor ALTO, el LED está encendido, cuando el pin está BAJO, está apagado.

El Nano tiene 8 entradas analógicas, cada una de las cuales proporciona 10 bits de resolución (es decir, 1024 valores diferentes). Por defecto, mide desde tierra a 5 voltios, aunque es posible cambiar el extremo superior de su rango utilizando la función analogReference (). Los pines analógicos 6 y 7 no se pueden usar como pines digitales. Además, algunos pines tienen una funcionalidad especializada:

- *I2C*: Pines 4 (SDA) y 5 (SCL). Admite la comunicación I2C (TWI) utilizando la biblioteca Wire (documentación en el sitio web de Wiring).

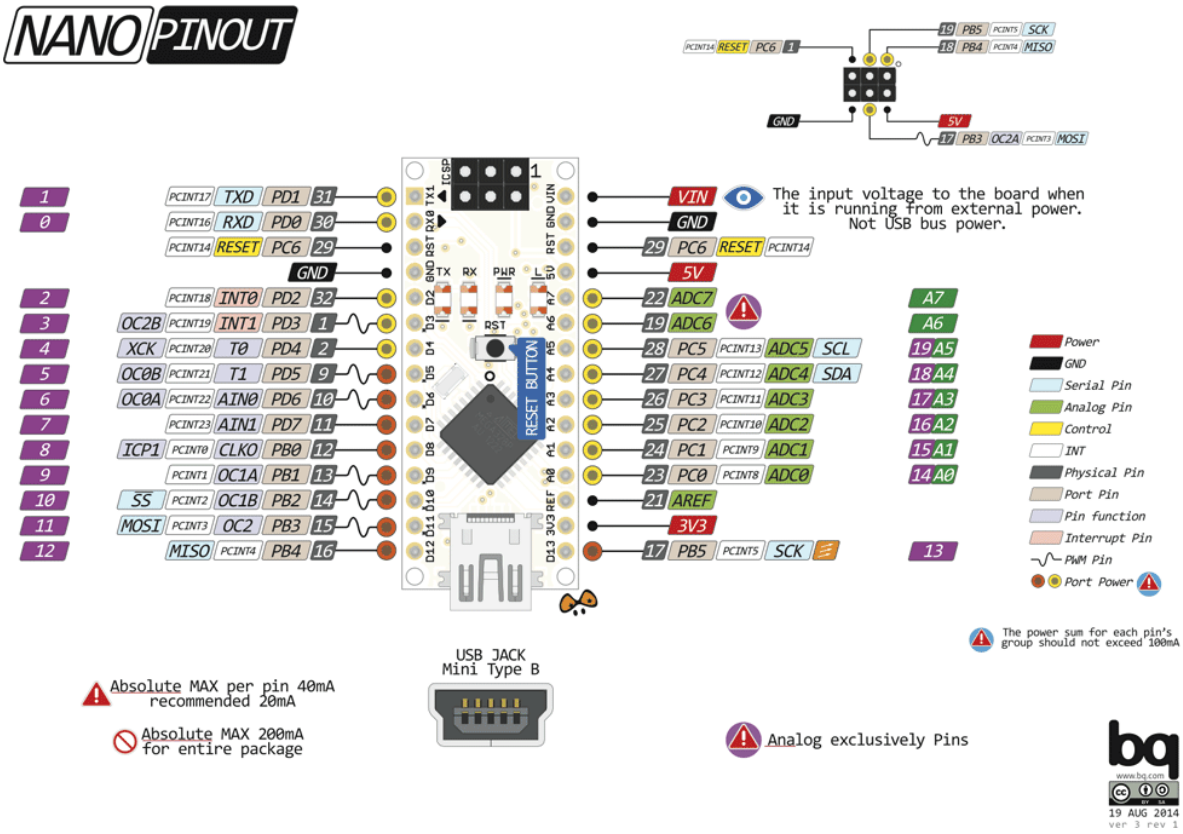


Figura 2.36 Distribución de pines en Arduino Nano

2.6 Editores de imágenes

Para el mejoramiento visual dentro de la simulación, se tomaron fotos y modificaron algunas imágenes, con el fin de dar un entorno más realista, para ello se utiliza algún software editor de imágenes y se ha optado por utilizar el software de licencia libre GIMP (Anónimo, Wikipedia, 2018), (Figura 2.37).

Figura 2.37 Logo GIMP

GIMP (GNU Image Manipulation Program) es un programa de edición de imágenes digitales en forma de mapa de bits, tanto dibujos como fotografías. Es un programa libre y gratuito. Forma parte del proyecto GNU y está disponible bajo la Licencia pública general de GNU y GNU Lesser General Public License.

GIMP tiene herramientas que se utilizan para el retoque y edición de imágenes, dibujo de formas libres, cambiar el tamaño, recortar, hacer fotomontajes, convertir a diferentes formatos de imagen, y otras tareas más especializadas. Se pueden también crear imágenes animadas en formato GIF e imágenes animadas en formato MPEG.

2.6.1 Interfaz

Figura 2.38 Entorno de GIMP

2.6.2 Herramientas principales

En la siguiente tabla se podrá observar las herramientas que se utilizaron en el proceso de mejoramiento visual dentro de las líneas.

Tabla 2-7 Herramientas Principales

Icono	Nombre	Atajo	Descripción
	Rectángulo	R	Seleccionar regiones cuadradas o rectangulares
	Elipse	E	Seleccionar zonas elípticas
	Libre	F	Seleccionar regiones de forma libre
	Difusa	U	Seleccionar regiones de color continuo
	Tijeras	I	Crea caminos para seleccionar formas
	Selección de primer plano	(ninguno)	Selecciona una región que contiene objetos de primer plano
	Herramienta de relleno	Shift + B	Rellena un área con un color o patrón.
	Mezcla (Gradiente)	L	Rellena un área con un gradiente.
	Lápiz	N	Dibuja líneas de borde duro; esto es, los pixels no están suavizados.
	Pincel	P	Pinta trazos de bordes suaves; esto es, los pixels están difuminados.
	Borrador	Shift + E	Borra pixels de una capa.
	Clonado	C	Copia pixels de una parte de una imagen a otra.
	Clonación de perspectiva	*(ninguna)*	Clona de una imagen origen tras aplicar una transformación de perspectiva.
	Mover	M	Mover capas, selecciones y otros objetos.
	Alineación	Q	Alinea o distribuye capas y otros objetos.
	Recorte	Shift + C	Elimina zonas del borde de la imagen.
	Rotación	Shift + R	Rota la capa activa, selección o ruta.
	Escalado	Shift + T	Escala la capa activa, selección o ruta.
	Recoge color	O	Establece el color a partir de los pixels de una imagen.
	Ampliación	Z	Ajusta el nivel de ampliación de la imagen.
	Balance de color	*(ninguno)*	Ajusta la distribución del color.
	Tono y saturación	*(ninguno)*	Ajusta el tono, la saturación y el brillo.

	Brillo y contraste	*(ninguno)*	Ajusta el brillo y el contraste.
	Umbral	*(ninguno)*	Reduce la capa actual o selección a solo blanco y negro.
	Niveles	*(ninguno)*	Cambia los rangos de intensidad de la capa activa o selección en cada canal.
	Curvas	*(ninguno)*	Cambia el color, brillo, contraste o transparencia de la capa activa o ruta.
	Posterizar	*(ninguno)*	Reduce el número de colores a un conjunto limitado.
	Desaturar	*(ninguno)*	Convierte todos los colores a su correspondiente sombra de gris.

2.7 Herramientas de simulación de circuitos

Para la elaboración, corrección y pruebas del circuito prototipo, se necesitó de un software de simulación de circuitos electrónicos, del cual se eligió un software de licencia libre que fuera amigable y con una biblioteca de componentes completos.

EasyEDA (Figura 2.39) es un paquete de herramientas EDA basado en la web que permite a los ingenieros de hardware diseñar, simular, compartir pública o privadamente, analizar esquemas , simulaciones y placas de circuitos impresos . Otras características incluyen la creación de una lista de materiales, archivos Gerber y archivos de selección.

Figura 2.39 EasyEDA

2.7.1 Interfaz

La interfaz para que el usuario realice el trazado de circuitos es la habitual en este tipo de aplicaciones (Figura 2.40).

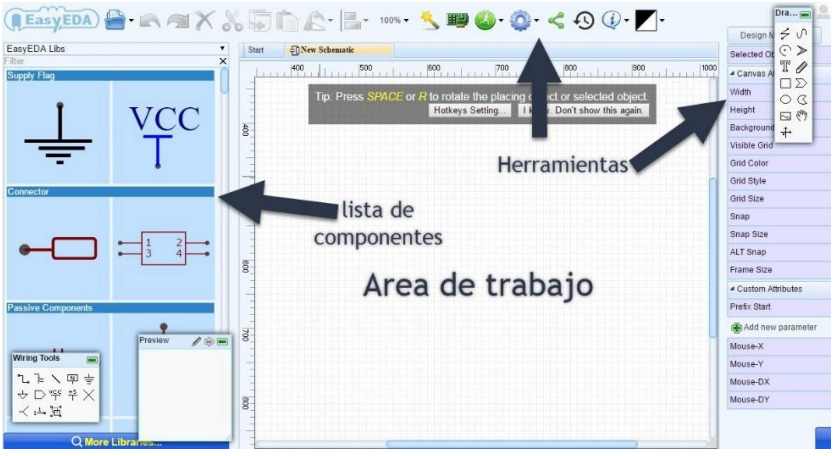


Figura 2.40 Entorno del programa EasyEDA

2.7.2 Herramientas

Barra de título: Situada en la parte superior de la pantalla, en ella se muestra el icono del programa.

Barra de menús: Permite el acceso a la mayor parte de las opciones del programa, sin embargo, algunas opciones solo están disponibles en los iconos de las barras de herramientas.

Barras de herramientas: Son varias y presentan las opciones para manejar los elementos del esquemático, tales como colocación de dispositivos, manejo de librerías, visualización, rotación de componentes, operaciones sobre bloques de dispositivos; así como las opciones de animación del diseño, entre otras.

Zona de trabajo: Es donde se colocará el diseño a realizar para posteriormente simularlo.

Ventana de vista completa/Zoom/Mapa del diseño: Esta ventana nos muestra una visión global del diseño, y mediante el puntero podemos seleccionar que zona del diseño estará visible en la ventana de edición, si no fuese posible visualizar todo sobre dicha ventana.

Lista de componentes: En esta ventana aparecerán todos los componentes, terminales, pines, generadores, etc. que se quieran introducir en el diagrama esquemático.

Capítulo 3. Desarrollo del prototipo

Durante las pruebas y búsqueda de soluciones para lograr la comunicación PC-SIMCAB, se emplearon 2 prototipos diferentes para sustituir la conexión original, que se tenía con la tarjeta de Nacional Instruments. El segundo prototipo optó por ser remplazado por una tarjeta de desarrollo Arduino Leonardo, y un circuito que cuenta con un arreglo similar al de las compuertas básicas del primer prototipo, este circuito se usa para la parte de los frenos, su función es enviar señales que van a las entradas de las compuertas lógicas del 7404, sus salidas se conectaron a una serie de resistencias y de ellas se juntaron en una sola señal, para ser enviadas a una de las entradas del Arduino. De forma separada se agregó una fuente de alimentación para el cofre de fallas, del cual es empleado con un cofre de simulación de fallas, que enviará voltaje a las luces para poder visualizar las diferentes fallas que puede tener el tren en el transcurso de su desplazamiento por la línea, como también un espacio para acoplar un Arduino nano que hará la función de un seleccionador de trenes que se podrá visualizar con un LCD 16x2.

3.1. Diagrama a bloques

A continuación, se presenta el diagrama de bloques (Figura 3.1), que muestra a los elementos más relevantes del sistema.

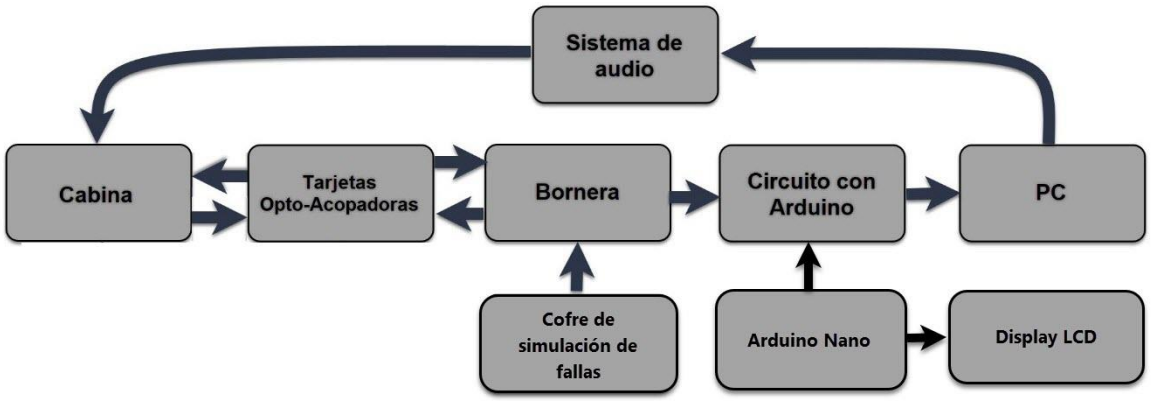


Figura 3.1 Diagrama de comunicación

Sistema de audio. - Conjunto de 2 bocinas con un subwoofer localizado en la cabina y conectada a la computadora.

Cabina. - Simulador de trenes SIMCAB basado al diseño de un tren de la compañía CAF.

Tarjetas Opto-acopladoras. - Tarjetas de acoplamiento de señales localizadas en la parte trasera de la cabina.

Bornera. - Tarjeta de conexiones de señales.

Cofre de simulación de fallas. - Circuito con botonera para enviar señales luminosas a la cabina.

Circuito con Arduino. - circuito configurado para recibir y enviar señales de la bornera conectada a la computadora.

Arduino Nano. - Tarjeta programada para la selección de trenes.

Display LCD. - Pantalla donde se mostrará la lista de trenes disponibles en la configuración dentro del circuito con Arduino.

PC. - Computadora con el programa OpenBVE para ser operada por el instructor.

3.2 Diagrama de flujo

La programación de las diversas rutinas que incorpora el sistema se representa mediante los diagramas a bloque mostrados en las figuras: 3.2, 3.3, 3.4 ,3.5 y 3.6.

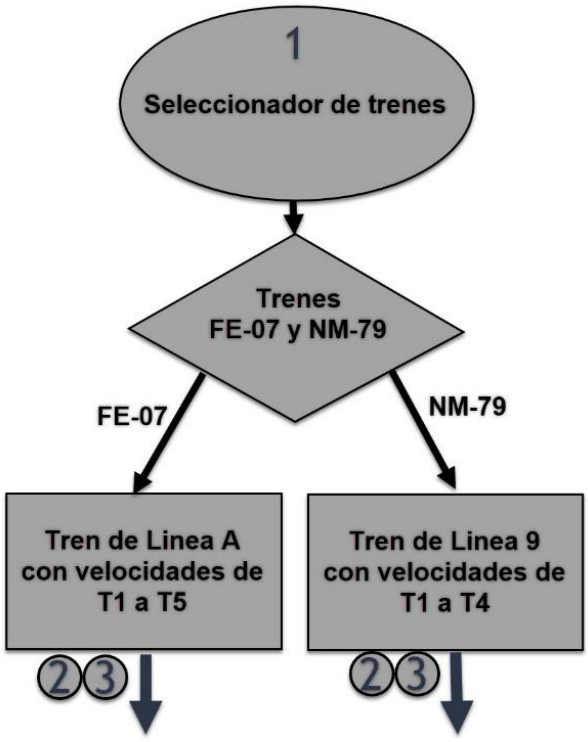


Figura 3.2 Parte 1 del seleccionador de trenes

Figura 3.3 Parte 2 del seleccionador de trenes

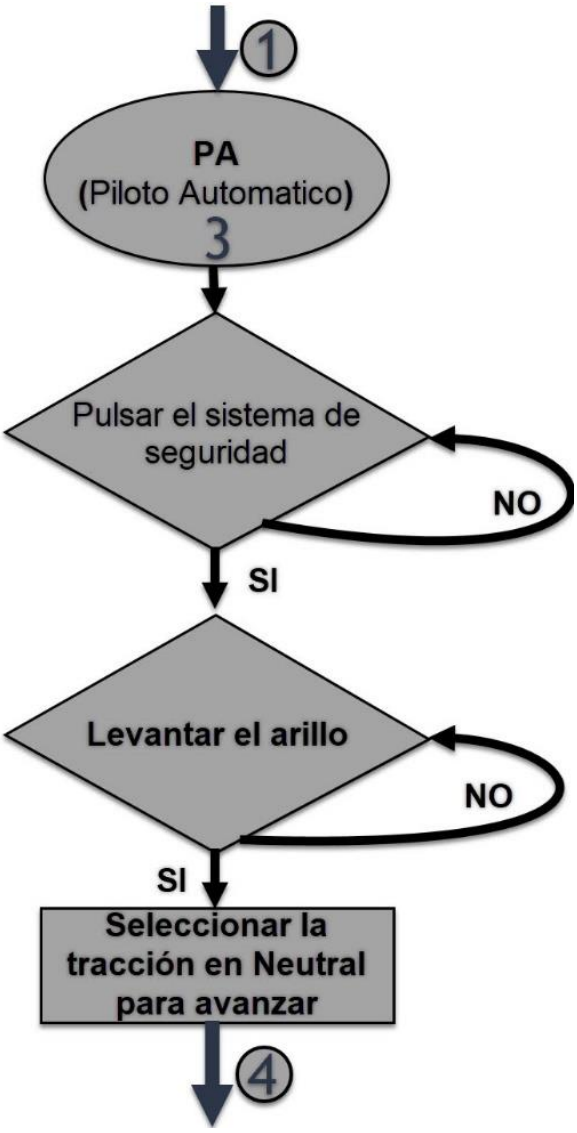


Figura 3.4 Parte 3 del seleccionador de trenes

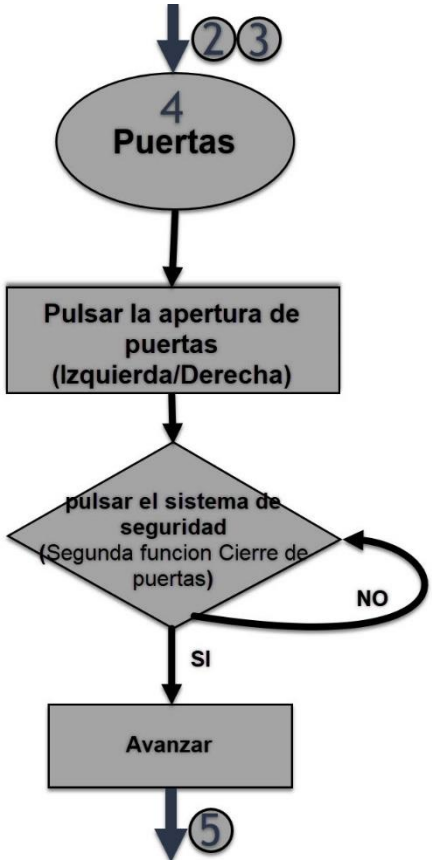


Figura 3.5 Parte 4 del seleccionador de trenes

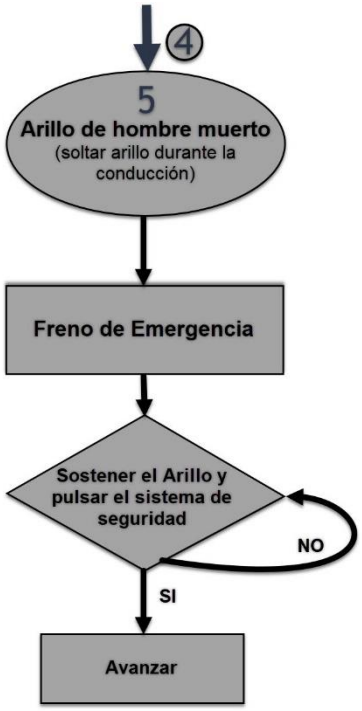


Figura 3.6 Parte 5 del seleccionador de trenes

3.3 Códigos

Código principal

En la primera parte del programa que se mostrará a continuación. Se incluye la biblioteca para la emulación del controlador de videojuegos, configuración del controlador, la distribución de las señales de la cabina en los diferentes pines del Arduino Leonardo y por último se tienen variables que nos servirán para el correcto funcionamiento del tren.

```
 #include "Joystick.h"
Joystick_ Joystick(JOYSTICK_DEFAULT_REPORT_ID, JOYSTICK_TYPE_GAMEPAD,
                   15, 0,                   // Button Count, Hat Switch
Count
                   false, false, false,     // X and Y and Z Axis
                   false, false, false,   // Rx, Ry, or Rz
                   false, false,          // rudder or throttle
                   false, false, false);  // accelerator, brake, or
steering
unsigned long MillisAnteriores = 0;
//entradas de control:
int IZQUIERDA = 3;
int ABRIR     = 2;
int CERRAR    = 5;
int DERECHA   = 4;
int val;
char lado;
int cont;
int cont1;
//entradas de velocidades:
int T1 = 6;
int T2 = 7;
int T3 = 8;
int T4 = 9;
int Fu = 10;//entrada freno emergencia
int atras = 11;
int adelante = 12;
int seguridad = 13;
int bocina = A1;
int arillo = A2;
int CM = A3;
int pa;
int fhm;
int FHM = A4;
int actual;
int actualf;
int cond;
int z;
unsigned long currentMillis;
void setup() {
  pinMode (FHM  , INPUT);
  pinMode (arillo  , INPUT);
  pinMode (CM  , INPUT);
  pinMode (IZQUIERDA , INPUT);
  pinMode (IZQUIERDA , INPUT);
  pinMode (ABRIR     , INPUT);
```

```
  pinMode (CERRAR    , INPUT);
  pinMode (DERECHA   , INPUT);
  pinMode (Tren1     , INPUT);
  cont = 1;
  cont1 = 1;
  pinMode (T1 , INPUT);
  pinMode (T2 , INPUT);
  pinMode (T3 , INPUT);
  pinMode (T4 , INPUT);
  actual = 0;
  actualf = 0;
  pa = 0;
  cond = 0;
  z = 0;
  Joystick.begin();
}
```

Para conseguir el funcionamiento adecuado de las puertas se escribió el siguiente código:

```
 //Puertas:
  if    (digitalRead(DERECHA) == HIGH && digitalRead(IZQUIERDA) == LOW)
  {
    lado = 'd';
  }
  if    (digitalRead(IZQUIERDA) == HIGH && digitalRead(DERECHA) == LOW)
  {
    lado = 'i';
  }
```

Las señales llamadas “Puertas derechas” y “Puertas izquierdas” nos sirven para seleccionar las puertas a abrir o cerrar (Llave T1) sin embargo al presionar el botón de cierre de puertas se podía observar que las señales llamadas “Cierre”, “Puertas derechas” y “Puertas izquierdas” se encontraban en un “1” lógico, por lo tanto al querer cerrar las puertas derechas igual cerraba las izquierdas y viceversa para resolver esto, al momento en que el Arduino recibe solo una señal de la selección se asigna una letra “d” o “i” para las puertas derecha e izquierda respectivamente. Cabe mencionar que cuando la posición de la llave T1 está en neutro los botones abrir y cerrar no responderán.

En la siguiente parte del código se lleva a cabo la apertura y cierre de puertas:

```
 // lado derecho
  if    (lado == 'd' && digitalRead(ABRIR) == HIGH && cont == 1) {
    delay(100);
    Joystick.pressButton (0);
    delay(100);
    Joystick.releaseButton (0);
    cont = 2;
  }
  if    (lado == 'd' && digitalRead(CERRAR) == HIGH && cont == 2) {
    delay(100);
    Joystick.pressButton (0);
    delay(100);
    Joystick.releaseButton (0);
    cont = 1;
```

```
}
// lado Izquierdo
if    (lado == 'i' && digitalRead(ABRIR) == HIGH && cont1 == 1)
{ delay(100);
  Joystick.pressButton (1);
  delay(100);
  Joystick.releaseButton (1);
  cont1 = 2;
}
if    (lado == 'i' && digitalRead(CERRAR) == HIGH && cont1 == 2) {
  delay(100);
  Joystick.pressButton (1);
  delay(100);
  Joystick.releaseButton (1);
  cont1 = 1;
}
```

A continuación, se ve la forma en la que se implementó el sistema de seguridad:

```
if (digitalRead (CERRAR) == HIGH && cont1 == 1 && cont == 1) {
  cond = 1;
  z = 0;
}
switch (cond) {
  case 1:
    conduccion();
    if (z == 0) {
      for (int x = 0; x <= 8; x++) {
        Joystick.pressButton (10);
        delay(100);
        Joystick.releaseButton (10);
        delay(100);
      }
    }
    z = 1;
    break;
}
```

El funcionamiento del código anterior es el siguiente, primero se comprueba que las puertas están cerradas esto con ayuda de los contadores “cont” y “cont1” (que se usaron anteriormente), si es así, tendremos acceso a la conducción, por otra parte, si durante la conducción se suelta el arillo más de tres segundos (se observara más adelante en el código), estaremos en una posición donde “FU” y “cond” serán igual a cero, así que para poder acceder a la conducción necesitamos tocar nuevamente el botón de cierre de puertas. El ciclo for nos sirve para poner al tren en una posición neutral.

La función llamada “conducción” tiene el objetivo de poder tomar control del metro ya sea con la conducción automática o manual, a continuación, se mostrará un poco del código acerca de esta función:

```
void conduccion () {
 if (digitalRead(CM) == HIGH && digitalRead(T1) == LOW
 && digitalRead(T2) == LOW && digitalRead(T3) == LOW
 && digitalRead(T4) == LOW && digitalRead(Fu) == LOW
 && digitalRead(arillo) == HIGH && val >= 1020) {
```

```
    if (actualf != 0 || actual != 0) {
      for (int x = 0; x <= 8; x++) {
        Joystick.pressButton (10);
        delay(100);
        Joystick.releaseButton (10);
        delay(100);
      }
      actual = 0;
      actualf = 0;
      fhm = 1;
      MillisAnteriores = millis();
    }
  }
```

La explicación del segmento anterior es: dado que no contamos con una señal que nos indique que el manipulador está en Neutro, nuestro indicador será cuando la llave de selección de modos de conducción está en CM y no hay ninguna velocidad o freno aplicado y el arillo de hombre muerto está activo.

Si durante la conducción el tren va a neutro se vuelven a verificar las variables “actualf” y “actual” que nos indican la posición de los controles de frenado y tracción respectivamente, y como alguno de estos será diferente de cero se activa el botón las veces necesarias para regresar a la posición de neutro.

Lo siguiente es la función del arillo de hombre muerto:

```
if (digitalRead(arillo) == LOW && fhm == 1) {
      currentMillis = millis();
      if (currentMillis - MillisAnteriores >= 5000) {
        if (digitalRead(arillo) == LOW && fhm == 1) {
          Joystick.pressButton (6);
          delay(100);
          Joystick.releaseButton (6);
          fhm = 0;
          cond = 0;
        }
      }
    }
```

La función del fragmento anterior es cuanto se está en CM y se suelta el arillo de hombre muerto se toma el tiempo y si este pasa de los 5 segundos el freno de emergencia será activado como medida de seguridad y no se podrá avanzar hasta que se vuelva a hacer el protocolo de seguridad antes descrito, en caso contrario si antes de los 5 segundos se vuelve a levantar el arillo de hombre muerto la conducción continuará de forma normal. La variable “millisAnteriores” es necesaria en diferentes partes del código para lograr esta medida de seguridad.

En la próxima sección de código se muestra como se asignan las señales a cada tracción:

```
    if (digitalRead(T1) == HIGH) {// Traccion 1
    if (digitalRead(T2) == HIGH && digitalRead(T4) == LOW) {// Traccion 2
    if (digitalRead(T2) == HIGH && digitalRead(T4) == HIGH) {// Traccion 3
    if (digitalRead(T3) == HIGH && digitalRead(T4) == LOW) {// Traccion 4
    if (digitalRead(T3) == HIGH && digitalRead(T4) == HIGH) {// Traccion 5
```

En el momento en el que desde la cabina se selecciona alguna tracción las señales enviadas son las siguientes:

- T1 para la tracción 1.
- T2 para la tracción 2.
- T2 y T4 para la tracción 3.
- T3 para la tracción 4.
- T3 y T4 para la tracción 5.

Con el código anterior asignamos las funciones creadas para que cada tracción según las señales obtenidas, se pueda activar de buena manera, igual es importante mencionar que si el arillo de hombre muerto no está activo, no se envía ninguna señal de tracción.

Cada if que vemos en el segmento de código anterior, contiene diversas funciones como se ve a continuación:

```
if (digitalRead(T1) == HIGH) {
      MillisAnteriores = millis();
      FaN();
      NaT1();
      T2aT1();
      T3aT2();
      T4aT3();
      T5aT4();
    }
```

Antes se tenía el problema de que al hacer cambios rápidos de tracción en el manipulador no se detectaban los cambios en las señales; debido a esto, cuando se pasaba de la tracción 4 a la 1 en la simulación no se mostraba la tracción elegida, con las funciones creadas, y que se pueden ver en el código anterior, se resuelve ese problema.

En la siguiente parte del código se puede ver como se conforman las funciones creadas:

```
void T1aT2() {
 if (actual == 1)  {
   Joystick.pressButton (2);
   dela(100);
   Joystick.releaseButton (2);
   delay(100);
   actual = 2;
  }
}
```

La función “T1aT2” nos ayudara para el traslado mencionado, su estructura es sencilla y todas las demás funciones para tracción y frenado son similares.

En las líneas siguientes se asignan valores obtenidos a cada freno:

```
    if (val >= 981 && val <= 1019) {// Freno 6
    if (val >= 961 && val <= 980) {// Freno 5
    if (val >= 856 && val <= 960) {// Freno 4
    if (val >= 576 && val <= 855) {// Freno 3
    if (val >= 501 && val <= 575) {// Freno 2
    if (val >= 200 && val <= 500) {// Freno 1
```

Como se observó en los diagramas electrónicos, los frenos pasan de ser varias señales a solo una señal; por lo tanto, para cada freno se le asignó un nivel de voltaje, como se observa en el código anterior; por lo tanto, para el F6 el valor leído debe estar entre 981 y 1019, para F5 entre 961 y 980, etc.

Dentro de cada asignación se encuentra lo siguiente:

```
if (val >= 981 && val <= 1019) { //Freno 6
     MillisAnteriores = millis();
     TaN();
     NaF1();
     F1aF2();
     F2aF3();
     F3aF4();
     F4aF5();
     F5aF6();
   }
```

En las líneas de código anteriores dentro de cada asignación como en el caso de las tracciones se en encuentran funciones creadas para el correcto funcionamiento.

La siguiente sección de código muestra el funcionamiento de la conducción manual:

```
if (digitalRead(CM) == LOW && digitalRead(Fu) == LOW) {
   if (pa == 0 && digitalRead(arillo) == HIGH ) {
     Joystick.pressButton (3);
     delay(100);
     Joystick.releaseButton (3);
     pa = 1;
   }
   MillisAnteriores = millis();
   while (digitalRead(arillo) == LOW && pa == 1) {
     currentMillis = millis();
     if (digitalRead(Fu) == HIGH && pa == 1) {
       Joystick.pressButton (3);
       delay(100);
       Joystick.releaseButton (3);
       pa = 0;
       cond = 0;
     }
     if (currentMillis - MillisAnteriores >= 5000) {
       Joystick.pressButton (3);
       delay(100);
```

```
        Joystick.releaseButton (3);
        delay(100);
        Joystick.pressButton (6);
        delay(100);
        Joystick.releaseButton (6);
        pa = 0;
        cond = 0;
      }
    }
  }
```

El código anterior funciona de la siguiente manera, debido a que no se cuenta con una señal de PA, nuestro indicador será cuando la señal de CM no se encuentre activa, para hacer uso del PA en la simulación, se debe realizar el protocolo de seguridad ya indicado y tener la llave de conducción en PA (CM inactivo) y el arillo alzado, de esta forma se manda la señal para activar el piloto automático.

En la simulación al igual que con el modo de conducción manual si el arillo permanece abajo más de 5 segundos al momento de estar activo el piloto automático, se debe detener el tren, para realizar esto se mandará una señal para que se desactive el piloto automático y la señal del freno de emergencia, si solo se enviara la señal del freno de emergencia se crea conflicto en la simulación.

En caso de activar el freno de emergencia de forma manual, se desactivará el piloto automático para que logre cumplir su propósito.

Por último, en el siguiente código, si se llega a cambiar entre PA a CM se manda la señal para que se desactive el piloto automático, y como la “cond =0”, se manda el tren a neutral.

```
if (digitalRead(Fu) == HIGH && pa == 1) {
   Joystick.pressButton (3);
   delay(100);
   Joystick.releaseButton (3);
   pa = 0;
   cond = 0;
 }
```

Código del seleccionador de trenes

En la parte del seleccionador de trenes, se utiliza la biblioteca de display para el LCD 16x2 con la conexión del módulo I^2C, para ocupar menos entradas del Arduino Nano y se configuro uno de los pines digitales para enviar la señal del tren elegido al Arduino Leonardo.

```
#include <LiquidCrystal_I2C.h>
#include<Wire.h>
LiquidCrystal_I2C lcd(0x3f, 16, 2);
int Tren1 = 2;
int Tren2 = 3;
int menu = 0;
int D5 = 5;
```

Para elegir uno de los dos trenes que ya están configurados en la simulación, se elaboró un menú, acompañado primero de una presentación del seleccionador, como se muestra a continuación.

```
void setup() {
  lcd.init();
  lcd.backlight();
  pinMode(5, OUTPUT);
  pinMode(Tren1, INPUT_PULLUP);
  pinMode (Tren2, INPUT_PULLUP);
  lcd.home();
  lcd.setCursor (3,0);
  lcd.print ("FES ARAGON");
  lcd.setCursor (6,1);
  lcd.print ("UNAM");
  delay (3000);
  lcd.clear ();
  lcd.setCursor (0,0);
  lcd.print ("S. P. Alejandro");
  lcd.setCursor (0,1);
  lcd.print ("O. L. Juan Luis");
  delay (3000);
  lcd.clear ();
  lcd.setCursor (1,0);
  lcd.print ("Seleccionador");
  lcd.setCursor (3,1);
  lcd.print ("de trenes");
  delay (3000);
  lcd.init ();
  updateMenu();
}
void loop() {
  if (!digitalRead(Tren1)) {
    menu = 1;
    executeAction();
    action1;
    updateMenu();
    delay(100);
  }
  if (!digitalRead(Tren2)) {
    menu = 2;
    executeAction();
    action2;
    updateMenu();
    delay(100);
  }
}
void updateMenu() {
  switch (menu) {
    case 0:
     menu = 0;
     lcd.clear();
      lcd.print(" FE-07 Linea A");
      lcd.setCursor(0, 1);
      lcd.print(" NM-79 Linea 9");
```

```
      break;
    case 1:
     menu = 1;
      lcd.clear();
      lcd.print(">FE-07 Linea A");
      lcd.setCursor(0, 1);
      lcd.print(" NM-79 Linea 9");
      break;
    case 2:
     menu = 2;
      lcd.clear();
      lcd.print(" FE-07 Linea A");
      lcd.setCursor(0, 1);
      lcd.print(">NM-79 Linea 9");
      break;
  }
```

Por último, se muestra el tren seleccionado en la LCD.

```
void action1() {
  lcd.clear();
  lcd.print(">FE-07 Elegido");
  delay(1000);
}
void action2() {
  lcd.clear();
  lcd.print(">NM-79 Elegido");
 delay(1000);
}
```

3.4 Diagrama esquemático

Los siguientes diagramas representan el funcionamiento electrónico de la cabina SIMCAB en cuanto a la tracción, puertas y así como el circuito creado para los frenos.

En la figura 3.7 se puede visualizar la representación del sistema electrónico en cuanto a la tracción.

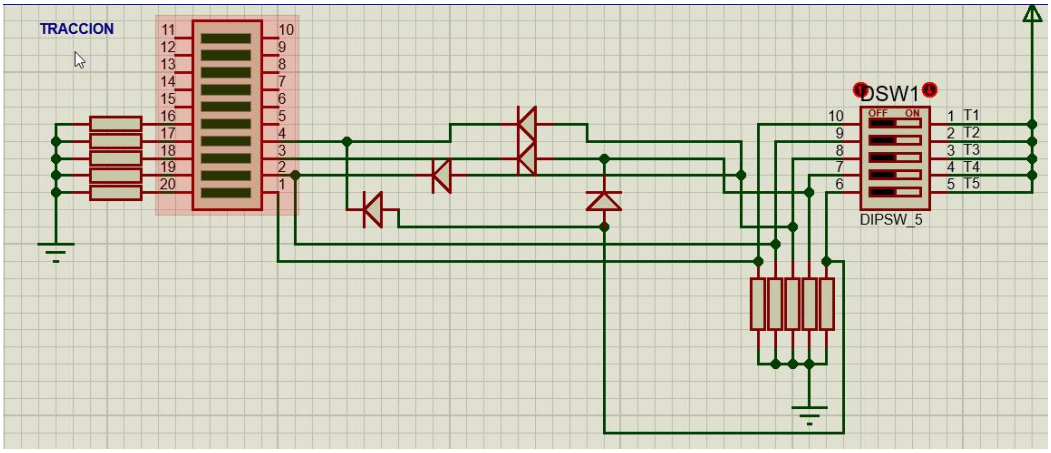


Figura 3.7 Diseño del sistema de Tracción

Las velocidades en la cabina son 5, mientras que las señales que envía son 4, con diferentes combinaciones, según la velocidad seleccionada. Para verificar las señales enviadas desde la cabina se colocaron LED a cada una de las salidas, y se tomó nota de los resultados.

En la figura 3.8 se puede visualizar la representación del sistema electrónico en cuanto a la apertura y cierre de puertas. En este caso, cuando en la cabina se acciona el cierre de puertas, se envían distintas señales; por último, como en el caso anterior, para verificar las señales enviadas, se colocaron LED a cada una de las salidas y se tomó nota de los resultados.

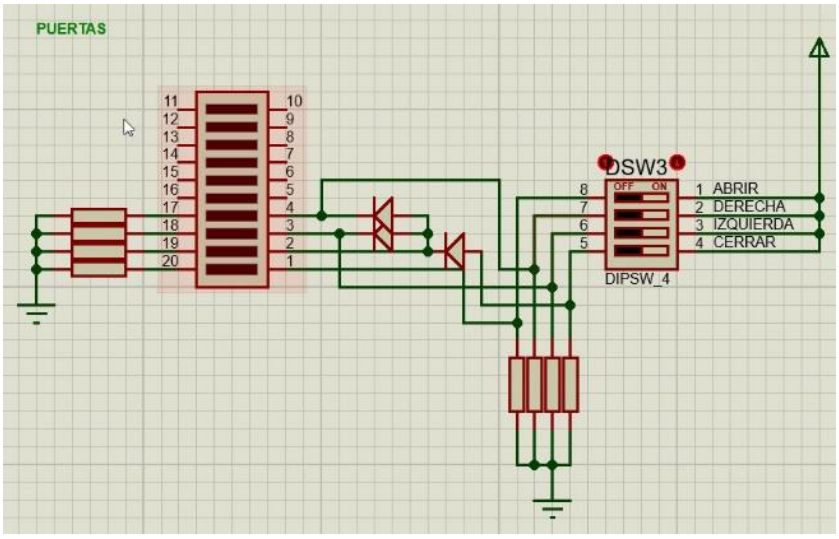


Figura 3.8 Diseño del sistema de Puertas

En la figura 3.9 se muestra el circuito creado para los frenos, estos pasan de ser varias señales, a solo una con diferentes niveles de voltaje según el freno accionado. Se conectó esta nueva señal al Arduino y se tomaron los valores arrojados para cada uno de los frenos.

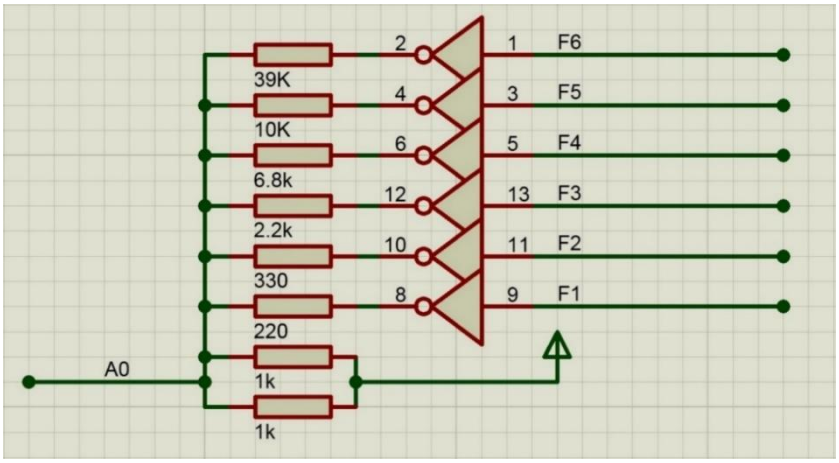


Figura 3.9 Diseño del sistema de Frenos

3.5 Mejoramiento visual

Originalmente, el diseño de las rutas tenía imágenes de poca resolución; con las cuales, dentro de la simulación, no se lograba notar lo que quería representar cada estación; además, se observó que a cada objeto le faltaba detalle para poder ser entendible; por ello, se agregaron o modificaron las imágenes con el programa de edición GIMP, para lo cual, se tomaron fotos de objetos, señales, paredes, escaleras que se encuentran dentro de cada estación, para añadirlas a la línea en desarrollo dentro de la simulación y así dar un toque de realismo dentro del recorrido. A continuación, se muestra el ejemplo de una señal de precaución (Figura 3.10) y una palanca de emergencia (Figura 3.11) con una resolución baja.

Figura 3.10 Señalamiento

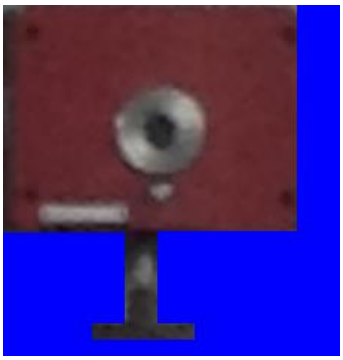


Figura 3.11 Palanca de emergencia

Para añadir las nuevas imágenes con retoque o mayor resolución dentro de la simulación, se modificaron y/o añadieron más de ciento veinte archivos de imágenes en formato PNG, que se localizan en la siguiente ruta en el explorador de archivos “C:\Users\Administrador\Desktop\BVE\Railway\Object\linea”; donde “línea” es aquella a la que se desea mejorar.

Con ello, dentro de la simulación se puede apreciar la diferencia de cada imagen. Tomando de nuevo como ejemplo la señal de precaución y la palanca de emergencia, ahora con una mejora visual (Figuras 3.12 y 3.13).

Figura 3.12 Señalamiento

Figura 3.13 Palanca de emergencia

Capítulo 4. Pruebas y resultados

4.1 Prototipo 1

El primer prototipo construido, está conformado por un Atmega328p, compuertas básicas, una tarjeta de un mando de PlayStation 2 (Figura 4.1), y un KITBON DUAL PlayStation (Figura 4.2).

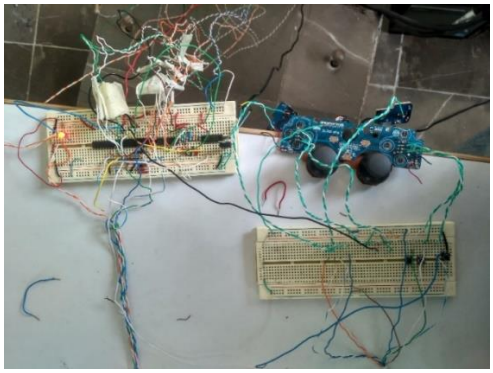


Figura 4.1 Conexión del prototipo 1

Figura 4.2 Adaptador Ps2-USB KITBON DUAL PlayStation

La comunicación del prototipo cumplía con la función requerida; sin embargo, las conexiones a la placa eran inestables y generaban falsos en las entradas de señales (Figura 4.3), y los voltajes de las señales de la cabina variaban de tal forma que terminaban provocando errores al enviar las señales al simulador.

Figura 4.3 Visualizador en entradas de señales

4.2 Prototipo 2

Debido a las fallas encontradas del primer prototipo, se tuvo que rediseñar, eliminando y reasignando los elementos a las entradas del Arduino (Figura 4.4), al código se le aplicaron cambios, y se agregaron algunas configuraciones al simulador para que reconociera cada señal en el Arduino y se efectuaron pruebas de funcionamiento.

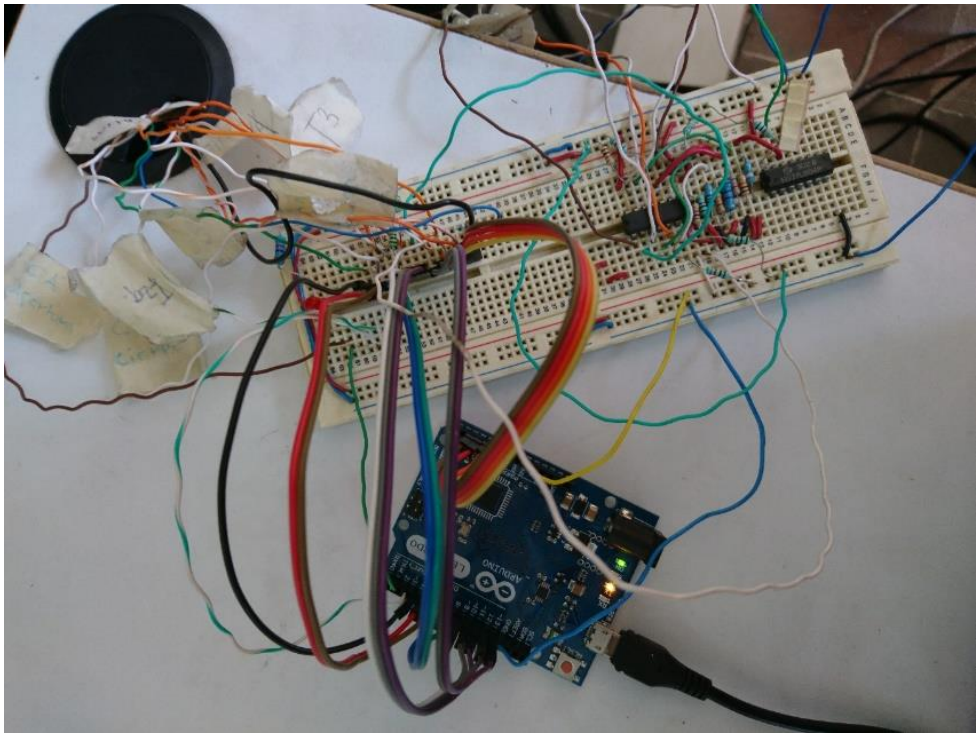


Figura 4.4 Conexión del prototipo 2

Durante las pruebas, el simulador no lograba acelerar de forma correcta, los frenos fallaban de forma constante, y las puertas se mantenían abiertas durante el recorrido. El sistema de frenos de emergencia no se activaba en el tiempo esperado (después de los tres segundos al soltar el arillo). Se verificaron las conexiones de las tarjetas opto-acopladoras a la bornera y al Arduino, encontrándose variaciones de voltaje en cada señal; por lo cual, a todas las señales se le agregaron resistores en configuración pull down, logrando así que la simulación no tuviera errores al ejecutar alguna palanca, velocidad o freno dentro de la cabina.

Después de solventar el problema de las señales, se realizaron nuevamente las pruebas con el simulador, logrando una mayor estabilidad en el funcionamiento de las señales con el Arduino Leonardo. Se comenzó con el diseño de un circuito en tarjeta perforada para eliminar la protoboard, al cual se le agregaron los resistores, la fuente de alimentación y el circuito para el frenado (Figura 4.5).

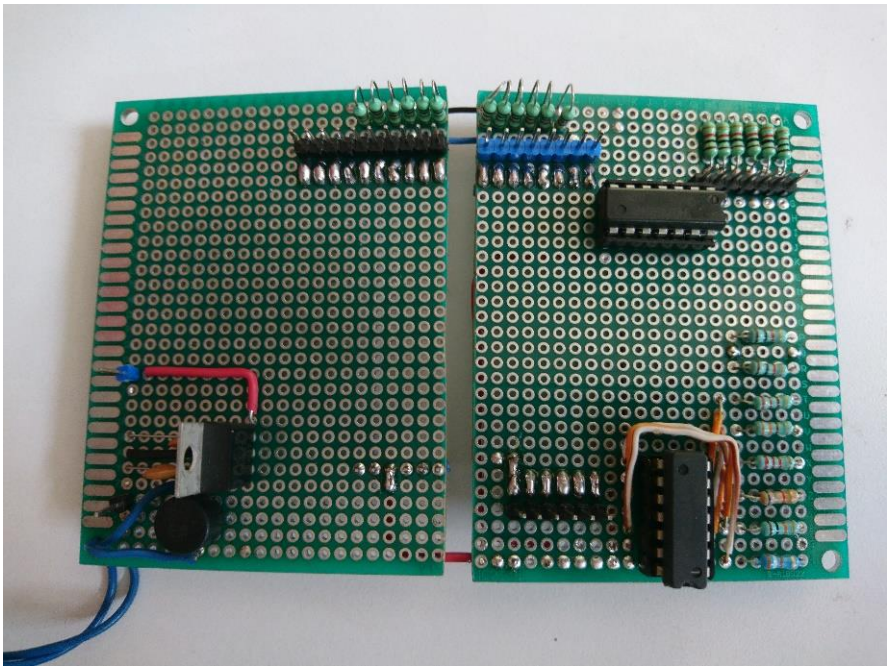


Figura 4.5 Circuito creado

Más adelante, se implementó la idea de añadir un seleccionador de trenes para poder utilizar más de un tipo de tren en el simulador, agregando los parámetros de un tren neumático de la línea 9 y un férreo de la línea A. Se puede visualizar el tipo de tren en una pantalla LCD de 16x2 (Figura 4.6), conectado a un Arduino Nano y enviando la selección al Arduino Leonardo por la entrada analógica A5.

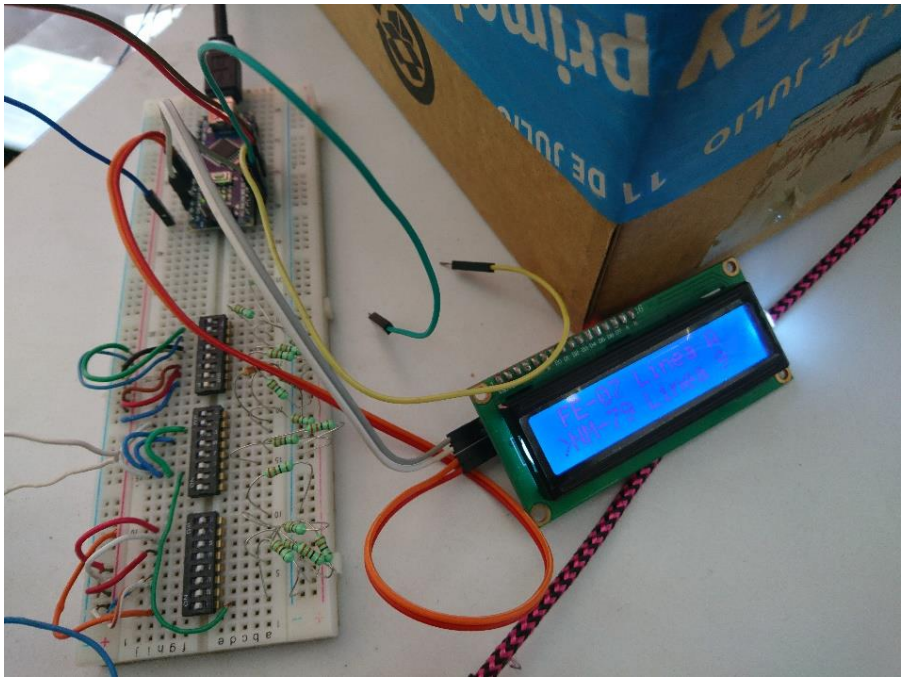


Figura 4.6 Conexión del seleccionador de trenes

A continuación, se muestran imágenes (Figuras 4.7 y 4.8) de la operación del simulador durante un recorrido en la línea 9.

Figura 4.7 Prueba de conducción en OpenBVE

Figura 4.8 Prueba de apertura de puertas en Open BVE

Cofre de simulación de fallas:

Para simular fallas en la SIMCAB se realizó un cofre (Figuras 4.9 y 4.10) para que el instructor pueda prender una o varias de las luces que requiera en el cofre de simulación de fallas (Figuras 4.11 y 4.12). Para su correcto funcionamiento se elaboró una fuente de 9 Volts que se encuentra en una esquina del circuito creado.

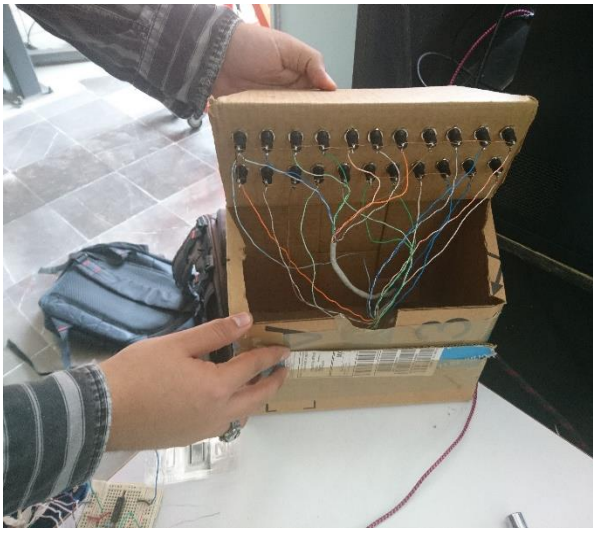


Figura 4.9 conexión del cofre de fallas

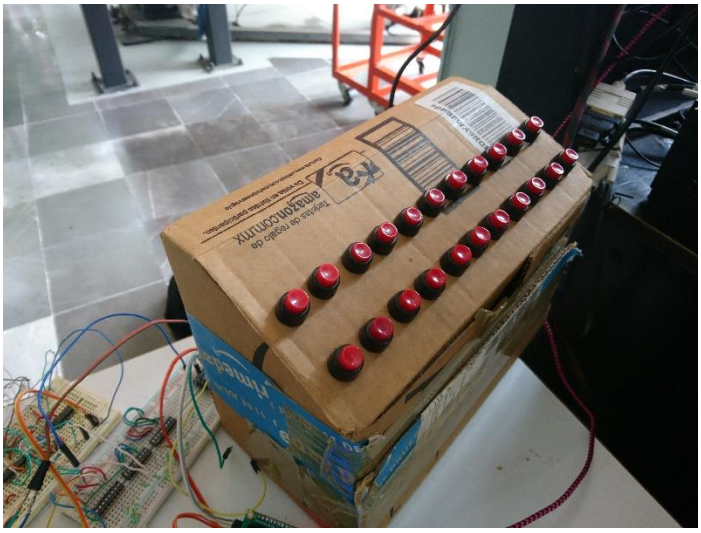


Figura 4.10 Cofre de fallas

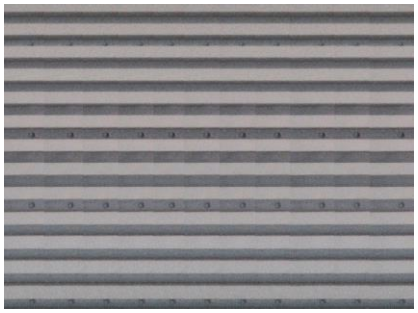
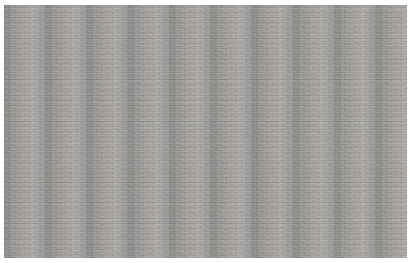
Figura 4.11 Avisos Luminosos en el Cofre de fallas

Figura 4.12 Avisos luminosos en el panel de fallas

4.3 Resultados Visuales

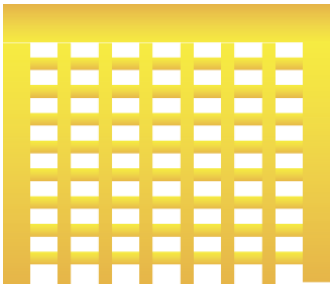
En la tabla 4.1 se puede observar un antes y un después del mejoramiento visual en los objetos que visualizan durante la simulación sobre las líneas A y 9 del metro.

Tabla 4-1 Resultados visuales

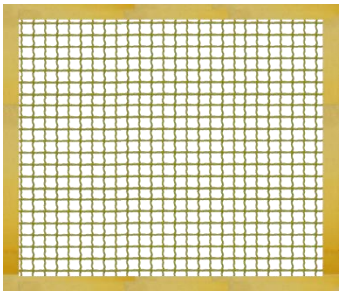
Inicio	Final
 Trafimuro	 Trafimuro
 Balasto	 Balasto
 Techo	 Techo
 Muro	 Muro

Señal

Señal



Reja



Reja

Extintor

Extintor

A continuación, se muestra el mejoramiento visual de las estaciones dentro del simulador, acompañado de una breve explicación de las mejoras realizadas.

Línea 9

En la versión inicial de la Línea 9, no se contaba con algunas de las imágenes del techo y publicidad, y las imágenes eran de poca resolución en paredes y señalamientos (Figuras 4.13, 4.15 y 4.17).

En su versión final de la Línea 9, se puede observar que fueron agregadas las imágenes de techo, se agregó publicidad y se modificaron señalamientos, paredes, etc. (Figuras 4.14, 4.16 y 4.18).

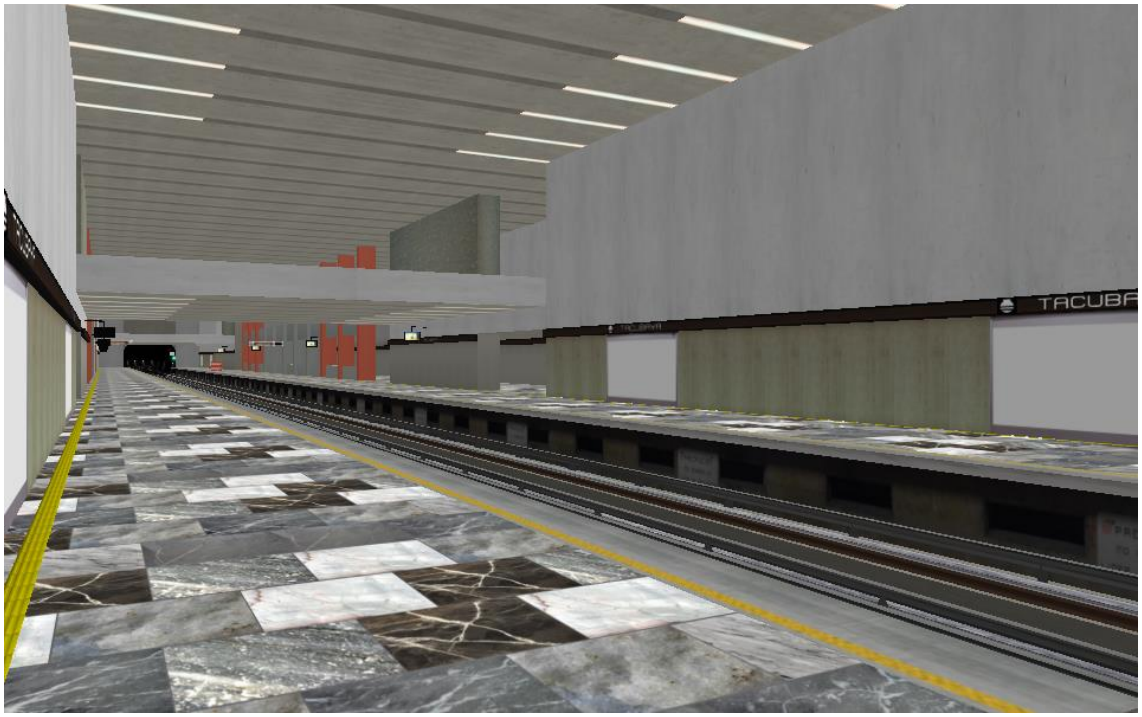


Figura 4.13 Estación Tacuba Línea 9 versión inicial

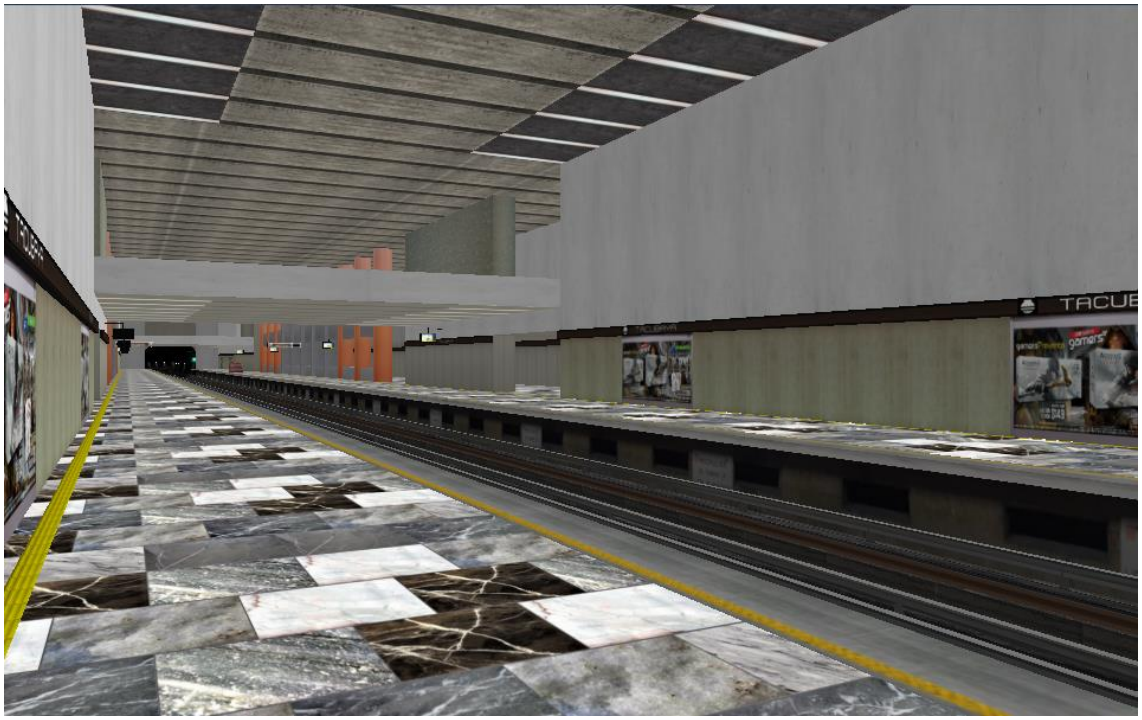


Figura 4.14 Estación Tacuba Línea 9 versión final

Figura 4.15 Estación Chilpancingo Línea 9 versión inicial

Figura 4.16 Estación Chilpancingo línea 9 versión final

Figura 4.17 Estación Centro Medico Línea 9 versión inicial

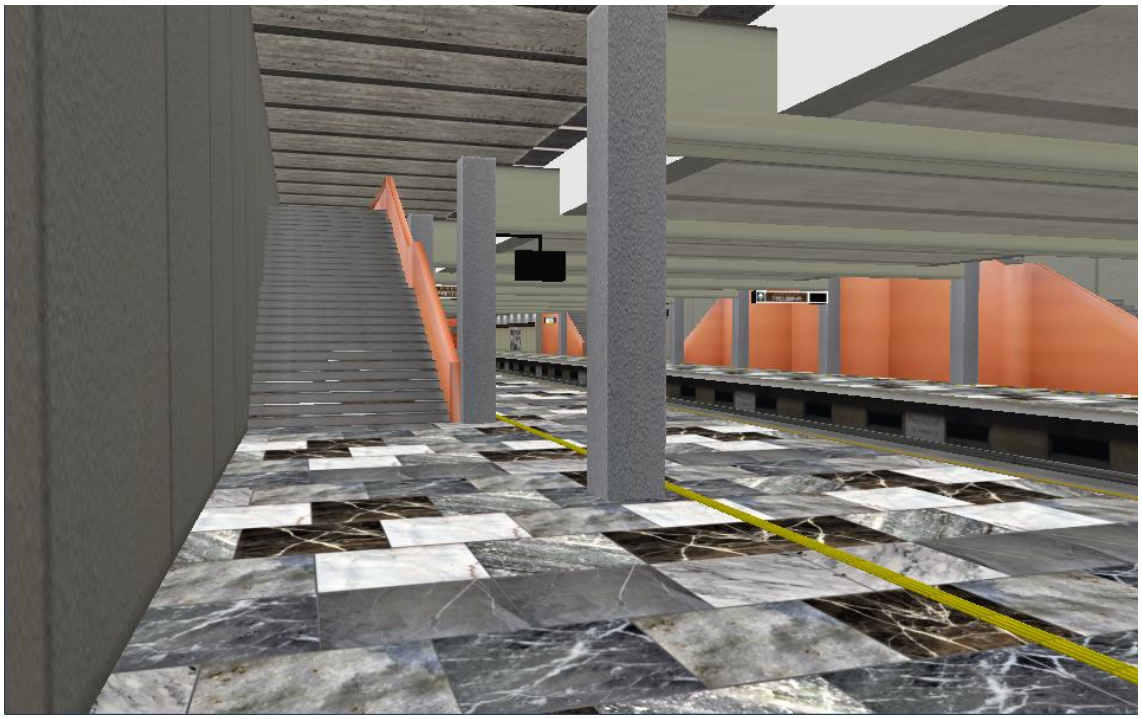


Figura 4.18 Estación Centro Medico Línea 9 versión final

Línea A

En la versión inicial de la línea A, las paredes de las estaciones estaban coloreadas de color azul, el cual no corresponde al color real de la estación; los barandales de las escaleras también presentaban inconsistencias, mismo caso con los rieles, escaleras y techos (Figuras 4.19, 4.21 y 4.23).

En su versión final de la Línea A, se puede observar que fueron agregadas nuevas imágenes de techo, edificios en la parte exterior de las estaciones como también, se modificó señalamientos, paredes, barandales, escaleras y terracería para una mejor aproximación de las estaciones reales (Figuras 4.20, 4.22 y 4.24).

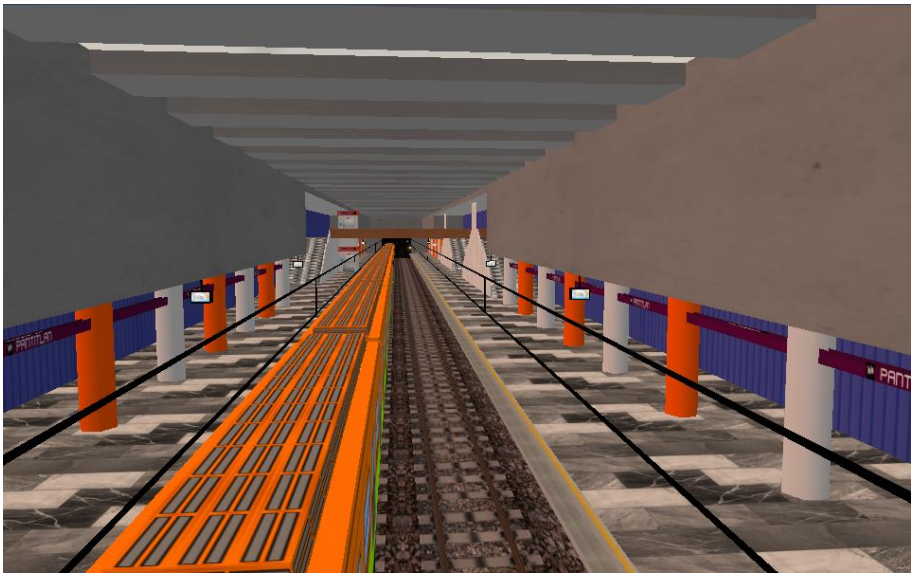


Figura 4.19 Estación Pantitlán Línea A versión inicial

Figura 4.20 Estación Pantitlán Línea A versión final

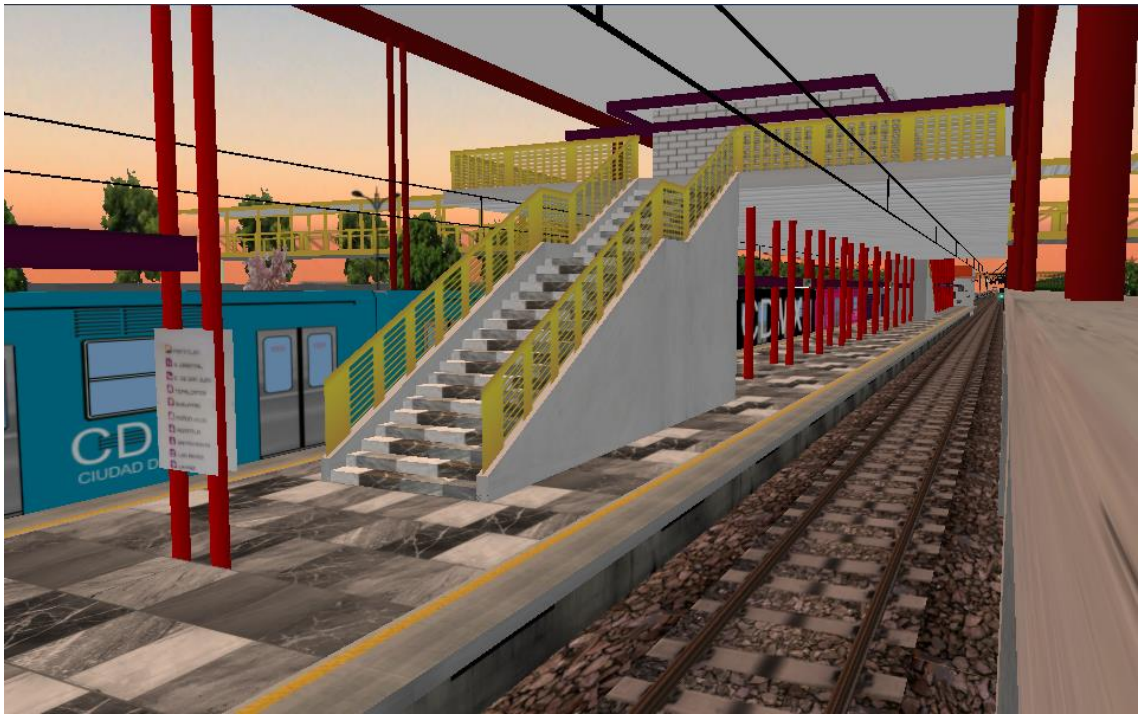


Figura 4.21 Estación Canal de San Juan Línea A versión inicial

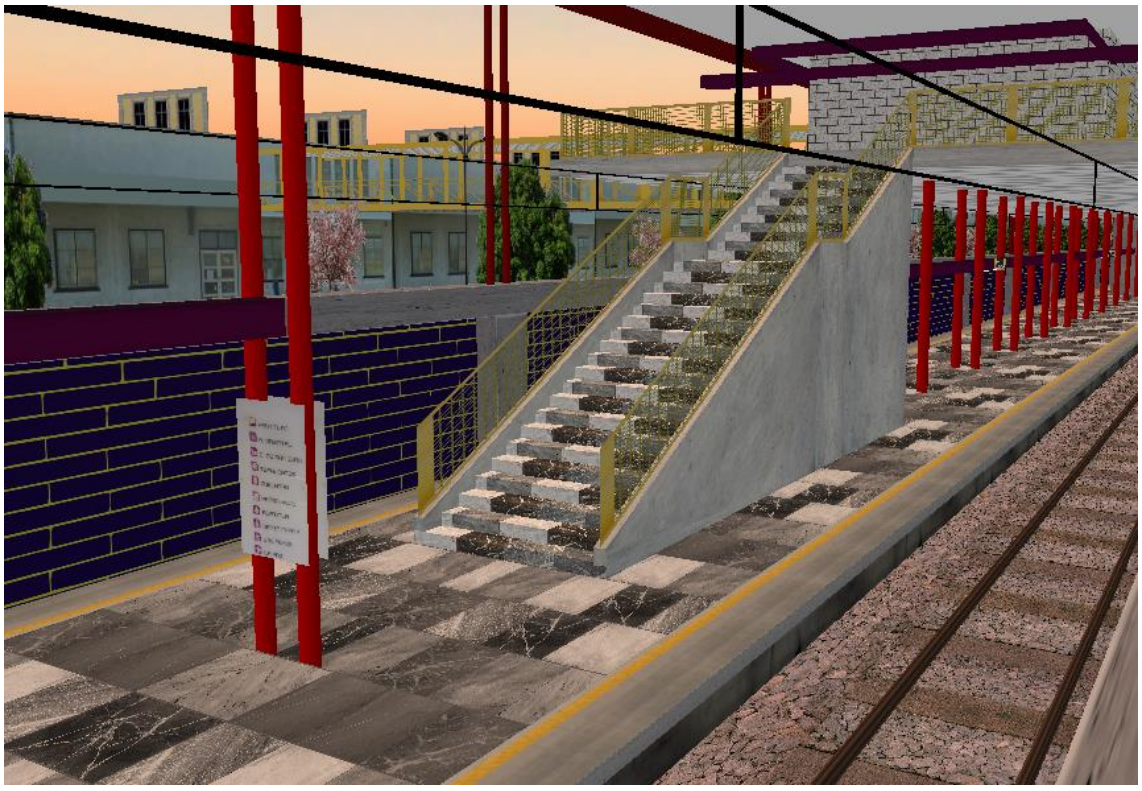


Figura 4.22 Estación Canal de San Juan Línea A versión final

Figura 4.23 Estación La Paz Línea A versión inicial

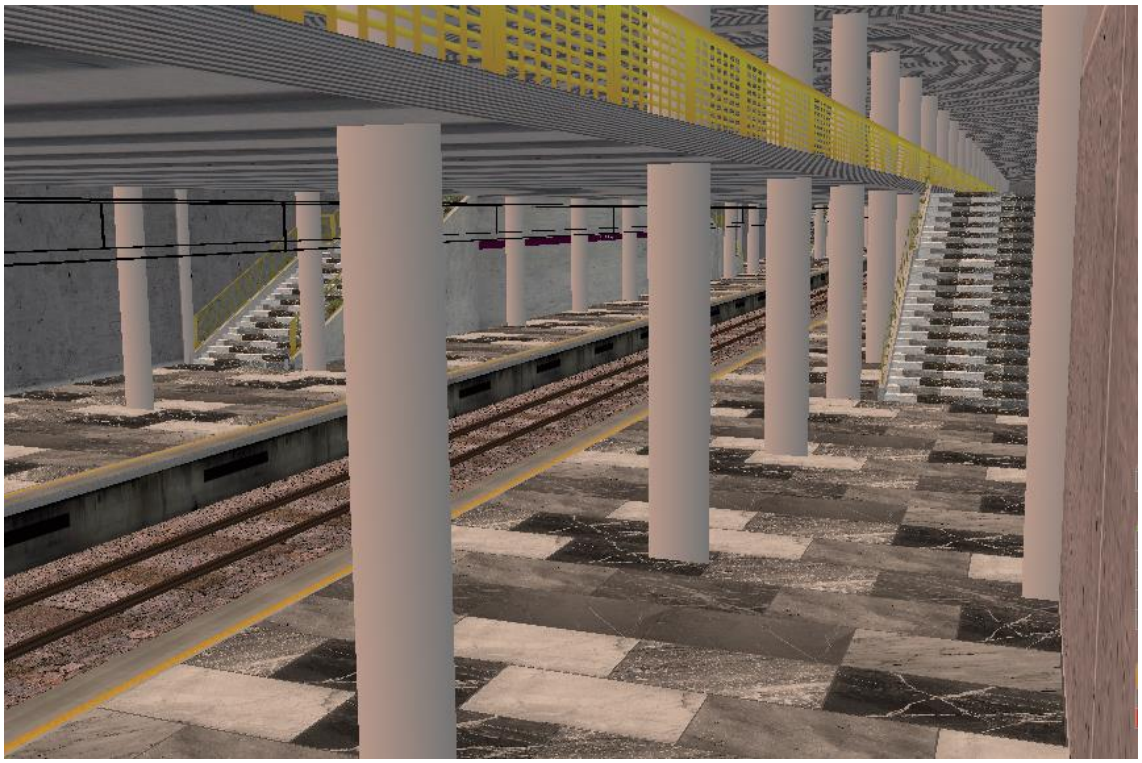


Figura 4.24 Estación La Paz Línea A versión final

Conclusiones y trabajo futuro

Conclusiones

Durante el desarrollo de la presente propuesta se pudo entender el funcionamiento a nivel electrónico de la cabina, de igual forma se logró adaptar sus señales para la correcta interacción con la tarjeta Arduino, y así la tarjeta pudo funcionar como un intérprete entre la cabina y el software de simulación, sin olvidar que igual cumple un papel fundamental para el protocolo de seguridad que se realiza en el recorrido de cualquier línea. Con la realización del segundo prototipo se pudo hacer uso de las tracciones, frenados, cierre y apertura de puertas, detección del modo de conducción seleccionado, y hacer un uso efectivo del modo manual o automático; así como también funciones secundarias tales como: bocinas y el botón de inicio de simulación.

Al revisar las tarjetas Opto-acopladoras y las diferentes conexiones de la cabina, se planteó la idea de reacondicionar los avisos luminosos que se encuentran en el tablero de control; con ello, se buscaron las señales de cada aviso y se conectaron a una caja que se elaboró con botones para poder mandar señales y visualizar los avisos de encendido en el tablero, ya que, al observar que si se energizaba con el Arduino, esta provocaba errores a las diferentes señales que manda la cabina dentro de la simulación, por lo tanto se agregó una fuente de poder que trabaja a 9 Volts, solo para el uso de los avisos luminosos.

Este cofre de simulación de fallas es fundamental para el instructor, ya que sirve como medio para mandar las señales de fallas, y así poder poner a prueba al próximo conductor del metro, ya que no solo es importante la conducción del metro, si no también saber enfrentarse a las diversas situaciones inesperadas que surgen en un día de trabajo.

En la parte del mejoramiento visual, se fue revisando las imágenes de cada estación, observándose que algunas eran poco legibles, inexistentes, y en otros casos imprecisas, por lo que se procedió a ser remplazadas o añadidas por otras, realizando capturas en las estaciones y usando programas de edición de imágenes. Todo ello se guardó en las carpetas correspondientes de cada línea y estación; para su lectura rápida se consideró una adecuada relación entre resolución y tamaño final de la imagen, para una mejor compatibilidad se trabajó con el formato PNG para todas las imágenes, gracias a esto se logró dar a la simulación una mejor presentación, fluidez y se visualización más cercana a la real de cada estación.

Con todo lo anterior, se recorrieron las líneas en la simulación, y revisando que el entorno visual fuera el correcto, pero se descubrió que el tren al llegar a una estación, no se detenía en el punto de paro estando en el modo automático, y se saltaba las estaciones hasta llegar a la terminal, revisando los códigos de los trenes, se encontró que los parámetros para activar al sensor que hace que el tren se

detenga, se estaban desactivados, por lo que se procedió a realizar las configuraciones adecuadas, resolviendo así dicha problemática.

Al hacer las pruebas de los trenes, se logró hacer un recorrido correcto de principio a fin de las líneas A y 9. Con ello se demostró que la comunicación entre la cabina y la PC cumple con los estándares y funciones mínimas requeridas por el instructor en turno, con ello se cumplió la meta de desarrollar un componente de bajo costo y funcional, con la finalidad de producir réplicas de simuladores en diferentes puntos del metro, para el aprendizaje de los nuevos conductores del STC.

Trabajo Futuro

Para dar un mejor funcionamiento en el área de capacitación, se busca lograr una interacción entre el cofre de simulación de fallas y la tarjeta de desarrollo, para que en un futuro el instructor no tenga que intervenir en el apagado de cualquier aviso luminoso, con ello, el aspirante a conductor estando en la cabina de simulación deba de ejecutar un procedimiento de activación de palancas dentro de la cabina para poder solventar la falla, y cuando esté procedimiento sea realizado correctamente, el aviso luminoso se apagará de forma automática.

Adicionalmente se busca añadir las líneas faltantes del metro dentro de la simulación, así como el mejoramiento visual de las existentes no trabajadas en este proyecto. De igual forma implementar los trenes faltantes para las nuevas líneas; como algunos trenes tienen diferencias en tracción y frenado con los usados en este proyecto se necesita hacer modificaciones menores en el código tanto como en el principal como en el del seleccionador de trenes.

Referencias

Barrios:, A. (21 de 5 de 2018). ¿Cómo funciona el Metro de la Ciudad de México? Obtenido de http://www.agustin.mx/posts.php?a=MetroMexico

Barrón, Z. N. (2010). Del naranja al dorado crecimiento del STC-Metro de la Ciudad de México. México, D.F. Universidad Nacional Autónoma de México.

CAF. (5 de 5 de 2018). Compañía CAF. Obtenido de https://www.caf.net/es/index.php

EasyEDA. (6 de 6 de 2018). EasyEDA. Obtenido de https://easyeda.com/es

Gonzáles, E. S. (2012). Desarrollo de un simulador 3D de manejo para la capacitación de conductores del sistema de transporte colectivo metropolitano. . México D.F. Instituto Politécnico Nacional.

López, C. M. (2002). Capacitación y formación de instructores incade un caso práctico (STC. Metro). México, D.F. Universidad Nacional Autónoma de México.: Gema.

López., J. R. (1999). El transporte urbano de pasajeros de la Ciudad de México en el siglo XX. México, D.F. Ciudad de México: Comité Editorial del Gobierno del Distrito Federal.

OpenBVE. (6 de 7 de 2018). ATS y ATC en OpenBVE. Obtenido de https://openbve-project.net/play-japanese/

playground.Arduino.cc. (12 de 11 de 2018). Referencia del lenguage C en Arduino. Obtenido de https://playground.Arduino.cc/ArduinoNotebookTraduccion/Structure

playground.Arduino.cc. (24 de 5 de 2018). Referencia del lenguage C en Arduino. Obtenido de https://www.prometec.net/intro-programacion/

vizcaíno, J. R. (2014). Sistemas integrados con Arduino. México Ciudad de México.: TEC MARCOMBO / ALFAOMEGA.

wikibooks. (5 de 6 de 2018). La caja de herramientas de GIMP. Obtenido de https://es.wikibooks.org/wiki/GIMP/La_caja_de_herramientas

wikipedia. (7 de 6 de 2018). EasyEDA. Obtenido de https://en.wikipedia.org/wiki/EasyEDA

wikipedia. (5 de 6 de 2018). GIMP. Obtenido de https://es.wikipedia.org/wiki/GIMP

wikipedia. (5 de 6 de 2018). Metro de la ciudad de México. Obtenido de https://es.wikipedia.org/wiki/Metro_de_la_Ciudad_de_M%C3%A9xico#Inicios

Referencias de imágenes:

Figura 2.33 Placa Arduino Leonardo, recuperada el 19 de Julio del 2018 desde: https://www.seeedstudio.com/Arduino-Leonardo-header-p-1233.html

Figura 2.34 Distribución de puertos en Arduino Leonardo, recuperada el 19 de Julio del 2018 desde: http://electrobist.com/product/arduino-leonardo-r3-cable/

Figura 2.35 Placa Arduino Nano, recuperada el 20 de Julio del 2018 desde: https://avelectronics.cc/producto/nano-v3-ch340/

Figura 2.36 Distribución de pines en Arduino Nano, recuperada el 19 de Julio del 2018 desde: https://www.avrfreaks.net/forum/sending-hex-codes-arduino-nano

Figura 2.37 Logo del programa GIMP, recuperada el 21 de agosto del 2018 desde: https://planetared.com/2017/11/redondear-esquinas-imagen-gimp/

Figura 2.38 Interfaz del programa GIMP, recuperada el 21 de agosto del 2018 desde: https://sites.google.com/site/tratamientodeimagenaudioanna/gimp-herramientas

Figura 2.39 Logo del programa EasyEDA, recuperada el 26 de agosto del 2018 desde: https://easyeda.com/eric/EasyEDA_LOGO-pOI0iAuM4

Figura 4.2 Adaptador Ps2-USB KITBON DUAL PlayStation, recuperada el 20 de agosto del 2018 desde: https://www.dx.com/p/kitbon-dual-ps1-ps2-psx-controller-game-console-joystick-to-pc-usb-converter-adapter-2018792#.XHm-4MBKiUk

Las demás Figuras utilizadas en la tesis son propias.

Anexo

Código Principal

```
#include "Joystick.h"
Joystick_ Joystick(JOYSTICK_DEFAULT_REPORT_ID, JOYSTICK_TYPE_GAMEPAD,
                   15, 0,                   // Button Count, Hat Switch
Count
                   false, false, false,     // X and Y and Z Axis
                   false, false, false,   // Rx, Ry, or Rz
                   false, false,          // rudder or throttle
                   false, false, false);  // accelerator, brake, or
steering
unsigned long MillisAnteriores = 0;
//entradas de control:
int IZQUIERDA = 3;
int ABRIR     = 2;
int CERRAR    = 5;
int DERECHA   = 4;
int val;
char lado;
int cont;
int cont1;
//entradas de velocidades:
int T1 = 6;
int T2 = 7;
int T3 = 8;
int T4 = 9;
int Fu = 10;//entrada freno emergencia
int atras = 11;
int adelante = 12;
int seguridad = 13;
int bocina = A1;
int arillo = A2;
int CM = A3;
int Tren1 = A4;
int pa;
int fhm;
int actual;
int actualf;
int cond;
int z;
unsigned long currentMillis;
void setup() {
  pinMode (arillo  , INPUT);
  pinMode (CM  , INPUT);
  pinMode (IZQUIERDA , INPUT);
  pinMode (IZQUIERDA , INPUT);
  pinMode (ABRIR     , INPUT);
  pinMode (CERRAR    , INPUT);
  pinMode (DERECHA   , INPUT);
  pinMode (Tren1 , INPUT);
  cont = 1;
  cont1 = 1;
  pinMode (T1 , INPUT);
  pinMode (T2 , INPUT);
```

```
  pinMode (T3 , INPUT);
  pinMode (T4 , INPUT);
  actual = 0;
  actualf = 0;
  pa = 0;
  cond = 0;
  z = 0;
  Joystick.begin();
}
void loop() {
  if (digitalRead(Fu) == HIGH) {
    Joystick.pressButton (6);
  }
  if (digitalRead(Fu) == LOW) {
    Joystick.releaseButton (6);
  }
  if (digitalRead(adelante) == HIGH) {
    Joystick.pressButton (5);
    Joystick.releaseButton (4);
  }
  if (digitalRead(atras) == HIGH) {
    Joystick.pressButton (4);
    Joystick.releaseButton (5);
  }
  if (digitalRead(bocina) == HIGH) {
    Joystick.pressButton (9);
  }
  Joystick.releaseButton (9);
  if (digitalRead(seguridad) == HIGH) {
    Joystick.pressButton (8);
  }
  Joystick.releaseButton (8);
  //Puertas:
  if    (digitalRead(DERECHA) == HIGH && digitalRead(IZQUIERDA) == LOW)
  {
    lado = 'd';
  }
  if    (digitalRead(IZQUIERDA) == HIGH && digitalRead(DERECHA) == LOW)
  {
    lado = 'i';
  }
  // lado derecho
  if    (lado == 'd' && digitalRead(ABRIR) == HIGH && cont == 1) {
    delay(100);
    Joystick.pressButton (0);
    delay(100);
    Joystick.releaseButton (0);
    cont = 2;
  }
  if    (lado == 'd' && digitalRead(CERRAR) == HIGH && cont == 2) {
    delay(100);
    Joystick.pressButton (0);
    delay(100);
    Joystick.releaseButton (0);
    cont = 1;
  }
  // lado Izquierdo
```

```
  if    (lado == 'i' && digitalRead(ABRIR) == HIGH && cont1 == 1)
  { delay(100);
    Joystick.pressButton (1);
    delay(100);
    Joystick.releaseButton (1);
    cont1 = 2;
  }
  if    (lado == 'i' && digitalRead(CERRAR) == HIGH && cont1 == 2) {
    delay(100);
    Joystick.pressButton (1);
    delay(100);
    Joystick.releaseButton (1);
    cont1 = 1;
  }
  if (digitalRead (CERRAR) == HIGH && cont1 == 1 && cont == 1) {
    cond = 1;
    z = 0;
  }
  switch (cond) {
    case 1:
      conduccion();
      if (z == 0) {
        for (int x = 0; x <= 8; x++) {
          Joystick.pressButton (10);
          delay(100);
          Joystick.releaseButton (10);
          delay(100);
        }
      }
      z = 1;
      break;
  }
}
void conduccion () {
  if (digitalRead(CM) == HIGH && digitalRead(T1) == LOW
  && digitalRead(T2) == LOW && digitalRead(T3) == LOW
  && digitalRead(T4) == LOW && digitalRead(Fu) == LOW
  && digitalRead(arillo) == HIGH && val >= 1020) {
    if (actualf != 0 || actual != 0) {
      for (int x = 0; x <= 8; x++) {
        Joystick.pressButton (10);
        delay(100);
        Joystick.releaseButton (10);
        delay(100);
      }
      actual = 0;
      actualf = 0;
      fhm = 1;
      MillisAnteriores = millis();
    }
  }
  if (digitalRead(CM) == HIGH) {
    if (digitalRead(CM) == HIGH && pa == 1) {
      Joystick.pressButton (3);
      delay(100);
      Joystick.releaseButton (3);
      pa = 0;
```

```
    for (int x = 0; x <= 8; x++) {
      Joystick.pressButton (10);
      delay(100);
      Joystick.releaseButton (10);
      delay(100);
    }
  }
  if (digitalRead(arillo) == LOW && fhm == 1) {
    currentMillis = millis();
    if (currentMillis - MillisAnteriores >= 5000) {
      if (digitalRead(arillo) == LOW && fhm == 1) {
        Joystick.pressButton (6);
        delay(100);
        Joystick.releaseButton (6);
        fhm = 0;
        cond = 0;
      }
    }
  }
  delay(100);
  if (digitalRead(T1) == HIGH) {// Traccion 1
    MillisAnteriores = millis();
    FaN();
    NaT1();
    T2aT1();
    T3aT2();
    T4aT3();
    T5aT4();
  }
  if (digitalRead(T2) == HIGH && digitalRead(T4) == LOW) {// Traccion 2
    MillisAnteriores = millis();
    FaN();
    NaT1();
    T1aT2();
    T3aT2();
    T4aT3();
    T5aT4();
  }
  if (digitalRead(T2) == HIGH && digitalRead(T4) == HIGH) {// Traccion
3
    MillisAnteriores = millis();
    FaN();
    NaT1();
    T1aT2();
    T2aT3();
    T4aT3();
    T5aT4();
  }
  if (digitalRead(T3) == HIGH && digitalRead(T4) == LOW) {// Traccion 4
    MillisAnteriores = millis();
    FaN();
    NaT1();
    T1aT2();
    T2aT3();
    T3aT4();
    T5aT4();
  }
```

```
    if (digitalRead(T3) == HIGH && digitalRead(T4) == HIGH) {// Traccion
5
      MillisAnteriores = millis();
      FaN();
      NaT1();
      T1aT2();
      T2aT3();
      T3aT4();
      if (digitalRead (Tren1) == HIGH){
      T4aT5();
      }
    }
    val = analogRead(0);
    delay(10);
    if (val >= 981 && val <= 1019) {// Freno 6
      MillisAnteriores = millis();
      TaN();
      NaF1();
      F1aF2();
      F2aF3();
      F3aF4();
      F4aF5();
      F5aF6();
    }
    if (val >= 961 && val <= 980) {// Freno 5
      MillisAnteriores = millis();
      TaN();
      NaF1();
      F1aF2();
      F2aF3();
      F3aF4();
      F4aF5();
      F6aF5();
    }
    if (val >= 856 && val <= 960) {// Freno 4
      MillisAnteriores = millis();
      TaN();
      NaF1();
      F1aF2();
      F2aF3();
      F3aF4();
      F5aF4();
      F6aF5();
    }
    if (val >= 576 && val <= 855) {// Freno 3
      MillisAnteriores = millis();
      TaN();
      NaF1();
      F1aF2();
      F2aF3();
      F4aF3();
      F5aF4();
      F6aF5();
    }
    if (val >= 501 && val <= 575) {// Freno 2
      MillisAnteriores = millis();
      TaN();
```

```
      NaF1();
      F1aF2();
      F3aF2();
      F4aF3();
      F5aF4();
      F6aF5();
    }
    if (val >= 200 && val <= 500) {// Freno 1
      MillisAnteriores = millis();
      TaN();
      NaF1();
      F2aF1();
      F3aF2();
      F4aF3();
      F5aF4();
      F6aF5();
    }
  }
  if (digitalRead(CM) == LOW && digitalRead(Fu) == LOW) {
    if (pa == 0 && digitalRead(arillo) == HIGH ) {
      Joystick.pressButton (3);
      delay(100);
      Joystick.releaseButton (3);
      pa = 1;
    }
    MillisAnteriores = millis();
    while (digitalRead(arillo) == LOW && pa == 1) {
      currentMillis = millis();
      if (digitalRead(Fu) == HIGH && pa == 1) {
        Joystick.pressButton (3);
        delay(100);
        Joystick.releaseButton (3);
        pa = 0;
        cond = 0;
      }
      if (currentMillis - MillisAnteriores >= 5000) {
        Joystick.pressButton (3);
        delay(100);
        Joystick.releaseButton (3);
        delay(100);
        Joystick.pressButton (6);
        delay(100);
        Joystick.releaseButton (6);
        pa = 0;
        cond = 0;
      }
    }
  }
  if (digitalRead(Fu) == HIGH && pa == 1) {
    Joystick.pressButton (3);
    delay(100);
    Joystick.releaseButton (3);
    pa = 0;
    cond = 0;
  }
}
void FaN() {
```

```
  F6aF5();
  F5aF4();
  F4aF3();
  F3aF2();
  F2aF1();
  if (actualf == 1)  {
    for (int x = 0; x <= 1; x++) {
      Joystick.pressButton (2);
      delay(100);
      Joystick.releaseButton (2);
      delay(100);
    }
    actualf = 0;
  }
}
void TaN() {
  T5aT4();
  T4aT3();
  T3aT2();
  T2aT1();
  if (actual == 1) {
    Joystick.pressButton (7);
    delay(100);
    Joystick.releaseButton (7);
    delay(100);
    actual = 0;
  }
}
void NaT1() {
  if (actual == 0) {
    Joystick.pressButton (2);
    delay(100);
    Joystick.releaseButton (2);
    delay(100);
    actual = 1;
  }
}
void T1aT2() {
  if (actual == 1)  {
    Joystick.pressButton (2);
    delay(100);
    Joystick.releaseButton (2);
    delay(100);
    actual = 2;
  }
}
void T2aT3() {
  if (actual == 2)  {
    Joystick.pressButton (2);
    delay(100);
    Joystick.releaseButton (2);
    delay(100);
    actual = 3;
  }
}
void T3aT4() {
  if (actual == 3)  {
```

```
    Joystick.pressButton (2);
    delay(100);
    Joystick.releaseButton (2);
    delay(100);
    actual = 4;
  }
}
void T4aT5() {
  if (actual == 4)  {
    Joystick.pressButton (2);
    delay(100);
    Joystick.releaseButton (2);
    delay(100);
    actual = 5;
  }
}
void T2aT1() {
  if (actual == 2) {
    Joystick.pressButton (7);
    delay(100);
    Joystick.releaseButton (7);
    delay(100);
    actual = 1;
  }
}
void T3aT2() {
  if (actual == 3) {
    Joystick.pressButton (7);
    delay(100);
    Joystick.releaseButton (7);
    delay(100);
    actual = 2;
  }
}
void T4aT3() {
  if (actual == 4) {
    Joystick.pressButton (7);
    delay(100);
    Joystick.releaseButton (7);
    delay(100);
    actual = 3;
  }
}
void T5aT4() {
  if (actual == 5) {
    Joystick.pressButton (7);
    delay(100);
    Joystick.releaseButton (7);
    delay(100);
    actual = 4;
  }
}
void NaF1() {
  if (actualf == 0)  {
    for (int x = 0; x <= 1; x++) {
      Joystick.pressButton (7);
      delay(100);
```

```
      Joystick.releaseButton (7);
      delay(100);
    }
    actualf = 1;
  }
}
void F1aF2() {
  if (actualf == 1)  {
    Joystick.pressButton (7);
    delay(100);
    Joystick.releaseButton (7);
    delay(100);
    actualf = 2;
  }
}
void F2aF3() {
  if (actualf == 2)  {
    Joystick.pressButton (7);
    delay(100);
    Joystick.releaseButton (7);
    delay(100);
    actualf = 3;
  }
}
void F3aF4() {
  if (actualf == 3)  {
    Joystick.pressButton (7);
    delay(100);
    Joystick.releaseButton (7);
    delay(100);
    actualf = 4;
  }
}
void F4aF5() {
  if (actualf == 4)  {
    Joystick.pressButton (7);
    delay(100);
    Joystick.releaseButton (7);
    delay(100);
    actualf = 5;
  }
}
void F5aF6() {
  if (actualf == 5)  {
    Joystick.pressButton (7);
    delay(100);
    Joystick.releaseButton (7);
    delay(100);
    actualf = 6;
  }
}
void F2aF1() {
  if (actualf == 2)  {
    Joystick.pressButton (2);
    delay(100);
    Joystick.releaseButton (2);
    delay(100);
```

```
    actualf = 1;
  }
}
void F3aF2() {
  if (actualf == 3)  {
    Joystick.pressButton (2);
    delay(100);
    Joystick.releaseButton (2);
    delay(100);
    actualf = 2;
  }
}
void F4aF3() {
  if (actualf == 4)  {
    Joystick.pressButton (2);
    delay(100);
    Joystick.releaseButton (2);
    delay(100);
    actualf = 3;
  }
}
void F5aF4() {
  if (actualf == 5)  {
    Joystick.pressButton (2);
    delay(100);
    Joystick.releaseButton (2);
    delay(100);
    actualf = 4;
  }
}
void F6aF5() {
  if (actualf == 6)  {
    Joystick.pressButton (2);
    delay(100);
    Joystick.releaseButton (2);
    delay(100);
    actualf = 5;
  }
}
```

Código del seleccionador de trenes

```
#include <LiquidCrystal_I2C.h> // Debe descargar la Librería que controla
el I2C
#include<Wire.h>

LiquidCrystal_I2C lcd(0x3f, 16, 2);
int Tren1 = 2;
int Tren2 = 3;
int menu = 0;
int D5 = 5;

void setup() {
  lcd.init();
  lcd.backlight();
  pinMode(5, OUTPUT);
  pinMode(Tren1, INPUT_PULLUP);
  pinMode (Tren2, INPUT_PULLUP);
  lcd.home();
  lcd.setCursor (3,0);
  lcd.print ("FES ARAGON");
  lcd.setCursor (6,1);
  lcd.print ("UNAM");
  delay (3000);
  lcd.clear ();
  lcd.setCursor (0,0);
  lcd.print ("S. P. Alejandro");
  lcd.setCursor (0,1);
  lcd.print ("O. L. Juan Luis");
  delay (3000);
  lcd.clear ();
  lcd.setCursor (1,0);
  lcd.print ("Seleccionador");
  lcd.setCursor (3,1);
  lcd.print ("de trenes");
  delay (3000);
  lcd.init ();
  updateMenu();
}
void loop() {
  if (!digitalRead(Tren1)) {
    menu = 1;
    executeAction();
    action1;
    updateMenu();
    delay(100);
  }
  if (!digitalRead(Tren2)) {
    menu = 2;
    executeAction();
    action2;
    updateMenu();
    delay(100);
  }
}
void updateMenu() {
  switch (menu) {
```

```
    case 0:
     menu = 0;
     lcd.clear();
      lcd.print(" FE-07 Linea A");
      lcd.setCursor(0, 1);
      lcd.print(" NM-79 Linea 9");
      break;
    case 1:
     menu = 1;
      lcd.clear();
      lcd.print(">FE-07 Linea A");
      lcd.setCursor(0, 1);
      lcd.print(" NM-79 Linea 9");
      break;
    case 2:
     menu = 2;
      lcd.clear();
      lcd.print(" FE-07 Linea A");
      lcd.setCursor(0, 1);
      lcd.print(">NM-79 Linea 9");
      break;
  }
}
void executeAction() {
  switch (menu) {
    case 1:
      action1();
      digitalWrite(D5, HIGH);
      break;
    case 2:
      action2();
      digitalWrite(D5, LOW);
      break;
  }
}
void action1() {
  lcd.clear();
  lcd.print(">FE-07 Elegido");
  delay(1000);
}
void action2() {
  lcd.clear();
  lcd.print(">NM-79 Elegido");
 delay(1000);
}
```