ESCUELA DE
INGENIERÍA EN CIENCIAS Y SISTEMAS
FACULTAD DE INGENIERÍA
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

Documentación
Proyecto 1

Grupo 8

Laboratorio Arquitectura de Computadoras y Ensambladores 1

Sección N

Integrantes:

Nombre	Carnet
Rodrigo Alejandro Hernández de León	201900042
José Manuel Solis Ortiz	201800517
Andrea María Cabrera Rosito	202010918
Ana Belén Contreras Orozco	201901604
Allen Giankarlo Román Vásquez	202004745
Carlos Roberto Quixtán Pérez	201901159
Pedro Antonio Castro Calderón	201900612

Introducción

En la presente documentación, se describe el funcionamiento del proyecto realizado - en este se utilizó Proteus y Arduino para la esquematización de la simulación de estacionamientos. Este estacionamiento, conformado por 16 parqueos, se conectó a una aplicación para dispositivos Android para que este pudiera ser gestionado desde allí. Todo esto se logró gracias a conexiones Bluetooth y la utilización de una API REST para el consumo de datos enviados y recibidos por los diferentes canales de comunicación.

Puede verse a detalle el código utilizado, al igual que las diferentes técnicas empleadas para la solución de los problemas y el llevado del control de dicho proyecto.

Desarrollo del Proyecto

Herramientas utilizadas:

- Proteus 8.10
- Arduino IDE 2.0.3
- Arduino Simulino Mega
- Microsoft Visual Studio Code(User)
- Node.js 16.18.0.
- MIT App Inventor
- Repositorio de Github:

  https://github.com/rodrialeh01/ACE1_Proyecto1_Grupo8

Código de Arduino

Dicho código es el que ayudó con el funcionamiento del Arduino MEGA, se realizó de la siguiente manera:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <Stepper.h>
```

Se importaron las siguientes librerías, utilizadas para la implementación de conexiones, el uso de la pantalla LCD con I2C y funcionamiento de los motores Stepper, respectivamente.

La pantalla LCD utilizada tiene las dimensiones de 16x4 en la cual se muestra un menú en el cual las personas podrán seleccionar y ver que el número de parqueos disponibles, apartados y no disponibles además de unas animaciones representadas por diferentes arreglos de chars, los cuales forman carros moviéndose en toda la pantalla.

Para esta pantalla se necesita usar la comunicación I2C, esto para que dicha pantalla no utilice una gran cantidad de pines en el arduino y se reduzca.

```
void Menu() {
  switch (seleccion) {
    case 0:
      seleccion = 1;
      break;
    case 1:
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Menu Informacion");
      lcd.setCursor(0, 1);
      lcd.print("> Disponibles");
      lcd.setCursor(0, 2);
      lcd.print("Reservados");
      lcd.setCursor(0, 3);
      lcd.print("Ocupados");
      break;
```

```
byte image02[8] = {B11111, B11100, B11100, B11111, B11111, B11111, B10001, B10001};
byte image03[8] = {B00000, B10000, B11110, B11101, B11111, B11111, B01000, B10000};
byte image01[8] = {B00000, B00001, B00111, B00111, B01111, B01111, B00010, B00001};
byte image42[8] = {B00000,
                   B00000,
                   B01110,
                   B11001,
                   B10101,
                   B10011,
                   B01110,
                   B00000
                  };
byte image43[8] = {
  B01110,
  B11001,
  B10101,
  B10011,
  B01110,
  B00000,
  B00000,
  B00000
};
```

Para la conexión con la aplicación se utilizaron la conexiones seriales, una para la conectarse al puerto de la interfaz y otra para conectarse al Bluetooth que se utilizará para mover las talanqueras (motores Stepper).

```
Serial.begin(9600); //BLUETOOTH
Serial1.begin(9600); //API
```

```
if (Serial.available() > 0) {
  estado = char(Serial.read());

  if (estado == '1') {
    abrirSalida();
  }
  else if (estado == '0') {
    abrirEntrada();
  }
}
```

Para el movimiento del motor Stepper, el cual será activado por medio de una señal enviada por vía Bluetooth del teléfono conectado a la aplicación, hacia un método de entrada y otro de salida, hará únicamente un movimiento de 90°, por lo que la revolución se hará únicamente 1 / 4. Y esperará para regresar al inicio del giro.

```
void abrirEntrada() {
  stepperB.setSpeed(40);
  stepperB.step(stepsPerRevolution / 4);
  delay(4000);
  stepperB.step(-stepsPerRevolution / 4);
}
```

Para validar que el lugar está libre u ocupado se utilizará la función “Estacionar()”, en donde se utilizaron fotorresistencias que se están validando para cuando una de ellas manda una señal diferente a la que tenía anteriormente se restará un espacio a la

```
void Estacionar() {
  Disponible = 16;
  Ocupado = 0;
  for (int i = 22; i <= 37; i++) {
    if (digitalRead(i) == HIGH) {
      Disponible--;
      Ocupado++;
    }
  }
}
```

variable “Disponible” y sumará a la variable “Ocupado”, para saber cuantos espacios quedan disponibles y ocupados, para ser mostrado en la pantalla LCD y el mismo usuario podrá verlo.

Los siguientes métodos son activados por medio de la conexión de la API:

- Reservado(): se validará que espacios están reservados y su luz led amarilla se encenderá y luego de 5 segundos dicha luz se apagará.
- EstadoParqueo(): se podrá validar si el parqueo está ocupado o libre, gracias a la señal que envía la fotoresistencia.
- verif(): el cual es utilizado para actualizar los estado de los estacionamientos en la API.

Descripción de la API

La API es el servicio que conecta el circuito en proteus y la aplicación android, fue programada en NodeJS y en ella se realizan las consultas y peticiones sobre el estado de cada posición del parqueo y la activación de las alarmas. Contiene las siguientes rutas:

/marcador

Esta ruta devuelve la distribución de los estados de las posiciones del parqueo en disponibles, reservados y ocupados, está ruta es llamada constantemente para actualizar dichos valores.

```
app.get('/marcador', (req,res) => {
    cont_disp = 0
    cont_ocup = 0
    cont_reserv = 0

    for(var i = 0; i < parqueo.length; i++) {
        for(var j = 0; j < parqueo[i].length; j++) {
            if (parqueo[i][j] == 1){
                cont_disp++
            }else if(parqueo[i][j] == 2){
                cont_reserv++
            }else if(parqueo[i][j] == 3){
                cont_ocup++
            }
        }
    }

    res.json({
        "Espacios disponibles": cont_disp,
        "Espacios ocupados": cont_ocup,
        "Espacios reservados": cont_reserv
    })

})
```

/estadoBarrera

Esta ruta se utiliza para consultar si el parqueo está completamente lleno, de tal forma que si lo está, la barrera de acceso no podrá ser activada, esto debido a que ya no habrían espacios disponibles.

```
app.get('/estadoBarrera', (req, res) => {
    if(cont_disp == 0){
        res.json({
            "Mensaje": "La barrera no se abrirá porque ya no hay espacios disponibles",
            "Estado": 0
        })
    }
    else{
        res.json({
            "Mensaje": "La barrera se abrirá porque aún hay espacios disponibles",
            "Estado": 1
        })
    }
})
```

/abrirBarreraSalida

Esta ruta siempre aceptará la petición, puesto que la barrera de salida siempre se activará sin tomar en cuenta el estado de cada posición del parqueo.

/reservarEspacio

Esta ruta recibirá un json con el número de nivel y el número de posición en el parqueo, el servicio buscará en su registro si la posición que el usuario

desea reservar está disponible, de ser así, la petición será aceptada y procederá a cambiar el estado de esa posición, de lo contrario, será rechazada.

```
app.post('/reservarEspacio',(req,res) => {
    data = req.body;
    response = "";
    estado = parqueo[data.nivel-1][data.posicion-1];
    if(estado == 1){
        parqueo[data.nivel-1][data.posicion-1] = 2;
        if(parqueo[0][0] == 2){
            arduinoSerialPort.write('A')
            parqueo[0][0] = 1
        }
        if(parqueo[0][1] == 2){
            arduinoSerialPort.write('B')
            parqueo[0][1] =1
        }
```

/ocuparEspacio

Esta ruta recibirá un json con el número de nivel y el número de posición en el parqueo, el servicio buscará en su registro si la posición que el usuario desea ocupar está disponible, de ser así, la petición será aceptada y procederá a cambiar el estado de esa posición, de lo contrario, será rechazada.

```
app.post('/ocuparEspacio',(req,res) => {
    data = req.body;
    response = "";
    estado = parqueo[data.nivel-1][data.posicion-1];
    if(estado == 1){
        parqueo[data.nivel-1][data.posicion-1] = 3;
        console.log(parqueo[data.nivel-1][data.posicion-1])
        response = {
            "Mensaje": "Se ocupó con éxito el espacio "+data.posicion.toString()+" del nivel "+data.nivel.toString(),
            "Estado": 1
        }
```

/desocuparEspacio

Esta ruta recibirá un json con el número de nivel y el número de posición del parqueo que el usuario procederá a desocupar, procediendo a cambiar el estado de esa posición a disponible para otro usuario.

```
app.post('/desocuparEspacio',(req,res) => {
    data = req.body;
    response = "";
    estado = parqueo[data.nivel-1][data.posicion-1];
    if(estado == 1){
        response = {
            "Mensaje": "El espacio "+data.posicion.toString()+" del nivel "+data.nivel.toString()+" ya está desocupado",
            "Estado": 0
        }
```

/alarmaAntirrobos

Recibe un json con el número de nivel y posición del parqueo, si la alarma actualmente está desactivada, procederá a activarla, de lo contrario, hará el proceso inverso.

```
app.post('/alarmaAntirrobos',(req,res) => {
    data = req.body;
    response = "";
    estado = alarmas[data.nivel-1][data.posicion-1];
    if(estado == 0){
        alarmas[data.nivel-1][data.posicion-1] = 1;
        response = {
            "Mensaje": "Se activó la alarma del espacio "+data.posicion.toString()+" del nivel "+data.nivel.toString(),
            "Estado": 1
        }
    }else if(estado == 1){
```

Aplicación Android

La aplicación Android denominada Parqueo es la cara del proyecto. Se conecta en su mayoría por medio de la Api Rest con algunos segmentos que se conectan por medio de bluetooth directamente al proteus. En la misma se llevarán a cabo las acciones efectuadas por los usuarios que desean efectuar alguna funcionalidad del parqueo. Entre sus funciones se encuentran:

Boton Bluetooth

Este botón te proporcionará una lista de dispositivos los cuales se es posible conectar por medio de bluetooth y cuando la conexión se haya efectuado el texto en la barra amarilla te mostrará un mensaje “CONECTADO”.

Textbox Ip

Esta función te permite agregar una dirección ip y enviarla al sistema.

Visualizador de espacios

La aplicacion es capaz de mostrar en tiempo real el estado de cada uno de los parqueos mostrando el nombre del parqueo en un espacio que cambia de color dependiendo su estado. Si el parqueo esta vacio el espacio se colorea a verde; si esta ocupado se colorea a rojo y si esta reservado se colorea ha amarillo ”.

Boton Reservar

Cada parqueo cuenta con un botón de reservar que como su nombre lo indica va a reservar dicho espacio. El espacio se mantendrá reservado por un tiempo límite al menos que el usuario desee cancelar la reservación o haya llegado al parqueo; en ese caso debe volver a presionar el botón.

Botón Alarma

Cada espacio de estacionamiento también cuenta con un botón de alarma antirrobos la cual solo se puede activar cuando el espacio ya se encuentra ocupado. Una vez activada el boton se volvera de color celeste y si se detecta un cambio en el estado del estacionamiento, comenzara a sonar. Para desactivar la alarma hay que volver a presionar el botón celeste lo cual la va a desactivar y volver a colorear el botón a rojo

Barreras (Talanqueras)

En la parte de abajo se encuentra el espacio de Talanqueras. Esta funcionalidad te permite abrir las barreras de entrada y salida respectivamente presionando un botón. Esta función está conectada directamente por bluetooth.

Vistas

En la parte de baja se encuentra un botón que te envía a otra screen denominada vistas. Vistas se encarga de proporcionarte un dato numérico de los estados del estacionamiento. Para volver a la screen principal también cuenta con un botón de “Hacia Mapeo”.

.

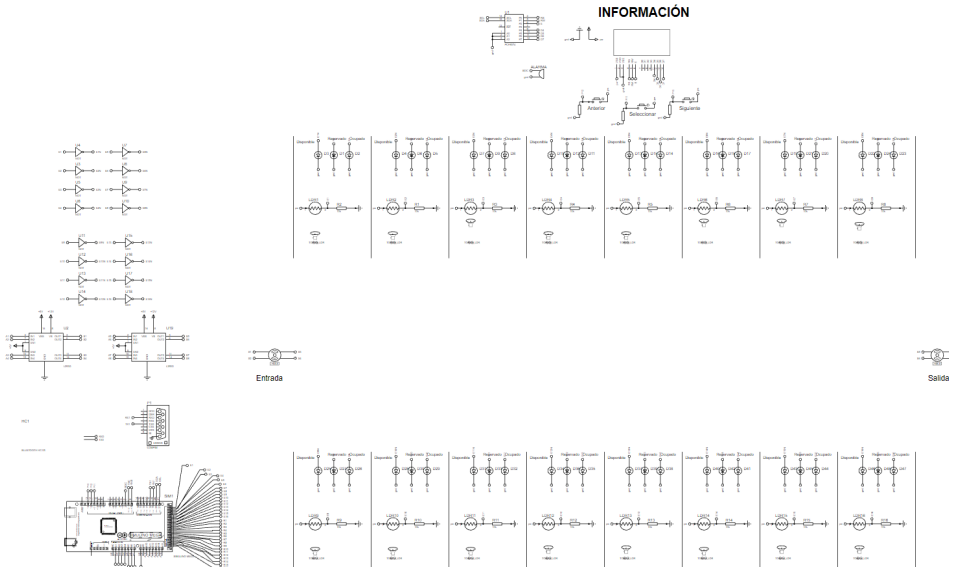
Diagramas de circuitos

Para el desarrollo del circuito eléctrico y la simulación del arduino, se utilizó la plataforma de simulación Proteus en su versión 8.10.

Para la elaboración del circuito fueron necesarios los siguientes componentes:

- SIMULINO MEGA
- COMPIM (Utilizada para la comunicación serial)
- BLUETOOTH HC-05
- L293D
- COMPUERTA NOT 7404
- MOTOR STEPPER
- LED (VERDE, ROJO, AMARILLO)
- FOTORESISTENCIA
- RESISTENCIA
- PANTALLA LCD (16X4)
- I2C BUS
- BOCINA
- BOTONES

El puerto Bluetooth HC-05 manda datos a la aplicación Android por medio de Bluetooth y el COMPIM se comunica con la API con la aplicación para pedir servicios que serán desplegados en la pantalla del teléfono Android.



Repositorio en Github

https://github.com/rodrialeh01/ACE1_Proyecto1_Grupo8