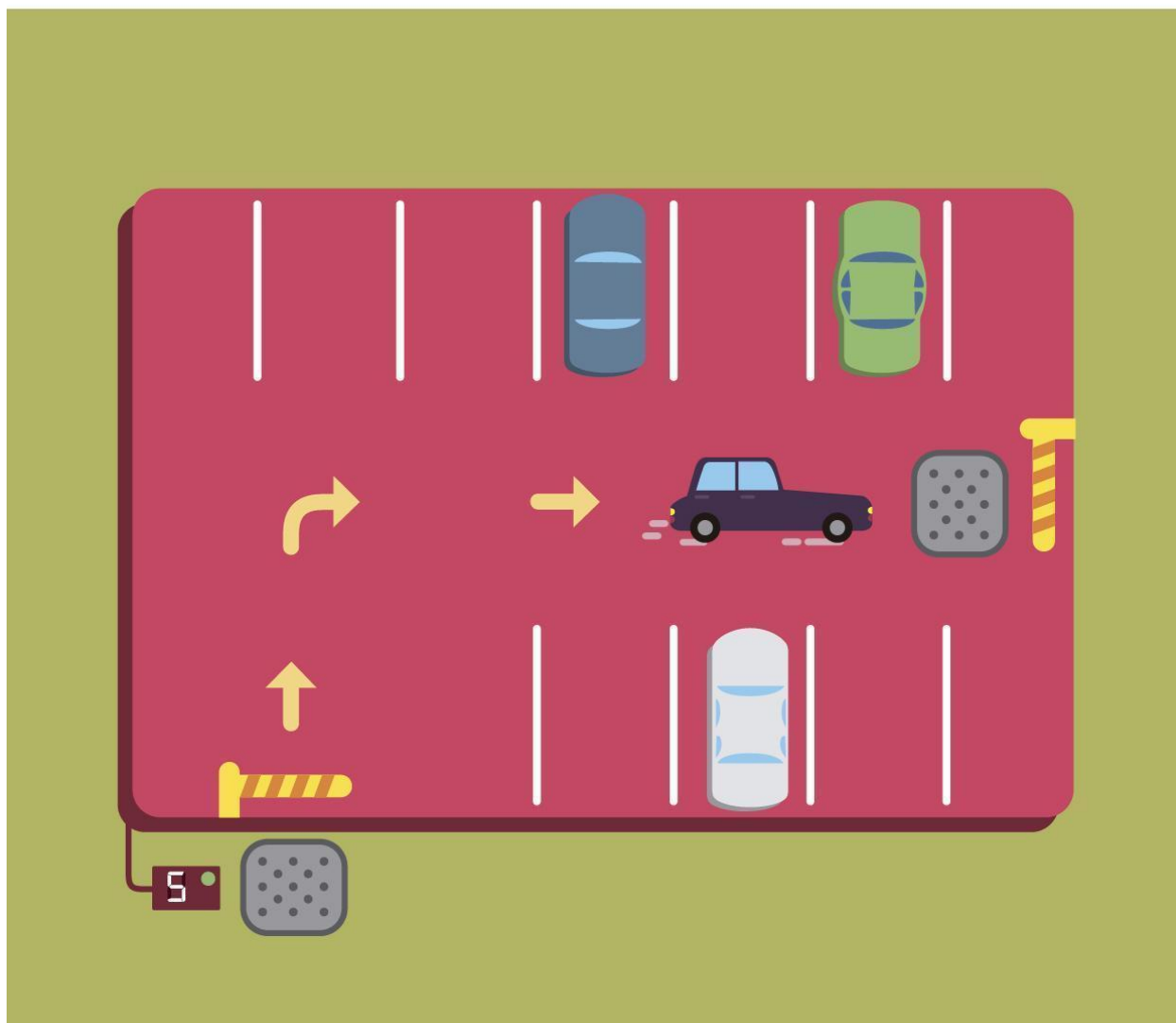


SABERES DIGITALES



ESTACIONAMIENTO AUTOMATIZADO

AUTORIDADES

Presidente de la Nación

Mauricio Macri

Vicepresidenta de la Nación

Marta Gabriela Michetti

Jefe de Gabinete de Ministros

Marcos Peña

Ministro de Educación, Cultura, Ciencia y Tecnología

Alejandro Finocchiaro

Titular de la Unidad de Coordinación General del Ministerio de Educación, Cultura, Ciencia y Tecnología

Manuel Vidal

Subsecretario de Coordinación Administrativa

Javier Mezzamico

Director Ejecutivo INET

Leandro Goroyesky

Gerenta General de EDUCAR Sociedad del Estado

Liliana Casaleggio

Directora Nacional de Asuntos Federales

María José Licio Rinaldi

Director Nacional de Educación Técnico - Profesional

Fabián Prieto

Coordinador de Secundaria Técnica

Alejandro Anchava

Responsable de Formación Docente Inicial y Continua INET

Judit Schneider

Coordinador General En FoCo

Pablo Trangone

| | |
|--|-------------------------------|
| AUTORIDADES | ¡Error! Marcador no definido. |
| ESTACIONAMIENTO AUTOMATIZADO | 3 |
| Ficha técnica | 3 |
| Presentación | 4 |
| Desarrollo | 5 |
| Nivel Inicial | 5 |
| Paso 1 - Realizar el montaje del servomotor en una Protoboard | 5 |
| Paso 2 - Programar el movimiento del servomotor | 6 |
| Paso 3 - Subir el código a la placa Arduino | 10 |
| Paso 4 - Mover el servo al presionar un pulsador | 10 |
| Paso 5 - Agregar LED de estado | 13 |
| Paso 6 - Armado de maqueta | 18 |
| Nivel Intermedio | 20 |
| Paso 1 - Subir el código a través de Arduino IDE | 20 |
| Paso 2 - Conectar el <i>display</i> de 7 segmentos. | 22 |
| Paso 2 - Conectar el <i>display</i> de 7 segmentos. | 22 |
| Fig.14 | 23 |
| Paso 3 - Programar el <i>display</i> de 7 segmentos. | 23 |
| Paso 4 - Incorporar el <i>display</i> de 7 segmentos al programa del estacionamiento | 29 |
| Nivel Avanzado | 32 |
| Paso 1 - Introducción a Internet de las Cosas (IoT) | 32 |
| Paso 2 - Crear un Panel de Control | 33 |
| Paso 3 - Conectar Módulo OBloq | 39 |
| Paso 4 - Arduino IDE | 40 |
| Paso 5 – Código bloqueante y IoT | 40 |
| Paso 6 - Programación IoT | 41 |
| Cierre | 49 |
| Reconocimientos | ¡Error! Marcador no definido. |

ESTACIONAMIENTO AUTOMATIZADO

Ficha técnica

| | |
|-------------------------------|--|
| Nivel educativo | Secundario. Ciclo Básico. |
| Descripción general | Los estudiantes elaborarán un dispositivo electrónico para la automatización de la gestión de un estacionamiento. |
| Niveles de complejidad | <p>Nivel inicial: instalar con una Placa Arduino un sistema de gestión que permita contabilizar si hay espacios libres en el estacionamiento y que abra las barreras de entrada y salida automáticamente, utilizando pulsadores y un servomotor.</p> <p>Nivel intermedio: agregar un display de 7 segmentos que permita visualizar la cantidad de espacios libres.</p> <p>Nivel avanzado: informar los datos obtenidos a un dispositivo móvil a través de internet de las cosas (IoT).</p> |
| Insumos | <ul style="list-style-type: none">• 2 x Servomotor SG-5010• 2 x Pulsadores• 10 x Resistencias 220 ohm• 2 x Resistencias 10K ohm• 1 x LED rojo• 1 x LED verde• 1 x Display 7 Segmentos• 20 x Cables dupont macho hembra• 20 x Cables dupont hembra hembra• 1 x Arduino UNO R3• 1 x Módulo IoT OBLOQ• 1 x Protoboard• 1 x Cable USB tipo B• 1 x Fuente de 9v 1 A (plug centro positivo, 5.5 x 2.1 mm) |

| | |
|-------------------------|--|
| Equipamiento | <ul style="list-style-type: none"> • Computadora • Soldador • Estaño • Alicates • Pinza de punta • Brusela |
| Otros requisitos | <ul style="list-style-type: none"> • Conexión a internet • Descargar el programa “mblock3” http://www.mblock.cc/software-1/mblock/mblock3/ |

Presentación

Descripción ampliada del proyecto

Se propone armar una maqueta/prototipo a escala de un estacionamiento con un sistema de gestión automatizado. Este permitirá abrir las barreras automáticamente cuando ingrese un vehículo y contabilizar, mediante el uso de pulsadores, si hay o no lugares libres en el estacionamiento.

En el nivel intermedio, se propone el uso de un *display* de 7 segmentos para visualizar el número exacto de lugares libres. En el nivel más avanzado, se plantea incorporar Internet de la Cosas (IoT) para poder realizar a distancia el monitoreo en tiempo real de la cantidad de espacios disponibles.

Al final de esta guía se puede encontrar un glosario donde se provee la información técnica necesaria para poder poner el proyecto en funcionamiento. El mismo cuenta con aclaraciones sobre los diversos elementos electrónicos involucrados así como también conceptos claves.

Objetivos

- Aproximarse al conocimiento y al manejo de distintos componentes electrónicos, mediante la construcción de un sistema de gestión para una maqueta de un estacionamiento que contabilice la cantidad de lugares disponibles y levante automáticamente las barreras de ingreso y salida.
- Introducirse en el armado de circuitos utilizando placas Arduino, servomotores, LED y un *display* de 7 segmentos.
- Analizar y desarrollar la programación de una estructura secuencial para un programa que permita contabilizar y visualizar la disponibilidad de espacios en el estacionamiento y que habilite el acceso de vehículos cuando haya espacios libres.
- Analizar y desarrollar la programación de un *display* de 7 segmentos.
- Utilizar Internet de la Cosas (IoT) para monitorear la cantidad de espacios disponibles en el estacionamiento.

Desarrollo

Nivel Inicial

Juana trabaja en el estacionamiento de autos de su familia. El estacionamiento tiene espacios disponibles para 8 autos. El terreno queda en una esquina, tiene la entrada por una calle y la salida por la otra. Como el estacionamiento queda en el centro de la ciudad, hay mucha circulación de vehículos y entradas y salidas permanentes.

Por esta razón, muchas veces, Juana termina perdiendo la cuenta de cuántos vehículos hay estacionados, y deja ingresar autos aunque ya no quede ningún lugar disponible. Como esto le genera problemas con los clientes, decidió armar un sistema de gestión que le permita visualizar fácilmente el nivel de ocupación del estacionamiento. Este sistema también regulará la apertura de la barrera de entrada para que se abra de forma automática cuando los autos ingresen al estacionamiento y haya lugares disponibles.

En esta instancia, la propuesta para el grupo será realizar una maqueta a escala con una placa Arduino que lleve conectados dos pulsadores (uno para el ingreso de autos y otro para la salida) que permitan la contabilización de los autos que hay dentro y dos LED que indiquen si hay lugares disponibles (led verde) o si el estacionamiento está completo (led rojo).

Paso 1 - Realizar el montaje del servomotor en una Protoboard

El proceso se inicia con el armado del circuito que permitirá controlar el movimiento de la barrera que habilita el ingreso y egreso de automóviles al estacionamiento. Para comenzar, se conectará un servomotor a la placa Arduino. Cuando un auto se encuentre en la entrada, se abrirá la barrera con un movimiento del servomotor en un ángulo de 90°.

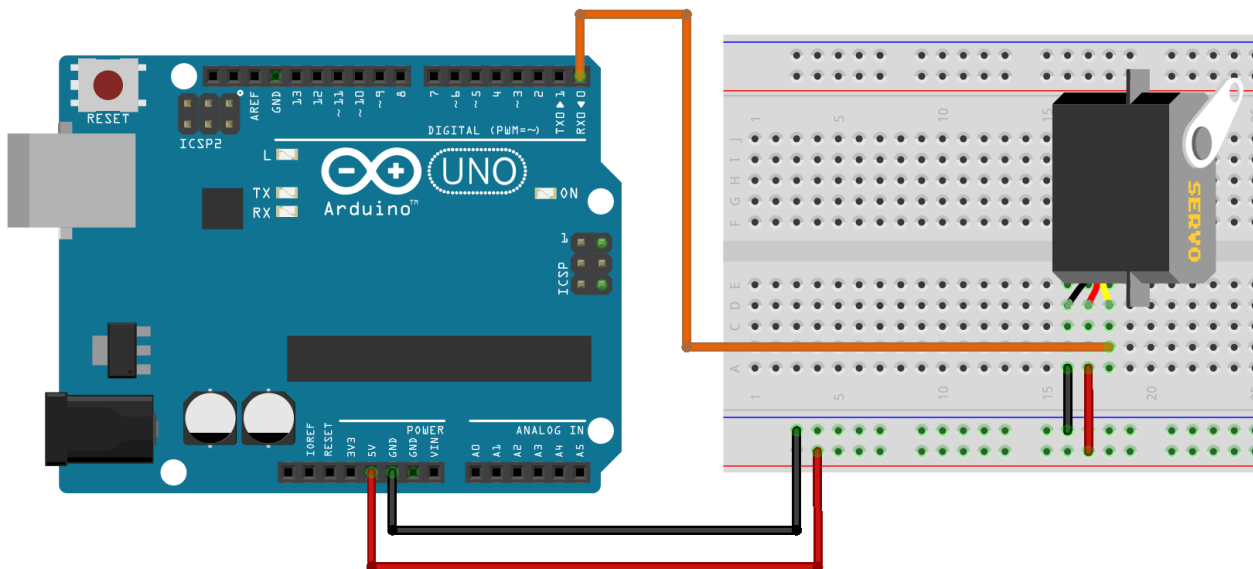
A continuación se explicará cómo armar el circuito y programar el sistema de gestión para el estacionamiento. Agregaremos el modelo de barrera a escala en el momento en que construyamos la maqueta del estacionamiento.



Un servo es un dispositivo que se compone de un motor y un sistema de control que le permite ubicarse en una posición específica. Los servos más comunes pueden moverse en un rango de 0° a 180°, sin poder girar de forma continua. Se suelen utilizar en aplicaciones tipo barreras o brazos mecánicos.

Fig.1

Conectamos el servomotor a la Protoboard como indica la figura:



Esquema 1

Paso 2 - Programar el movimiento del servomotor

La programación la realizaremos con mBlock3, un entorno de programación basado en Scratch2 que permite programar proyectos de Arduino utilizando bloques. Se puede descargar siguiendo este enlace: <http://www.mblock.cc/software-1/mblock/mblock3/>

Cuando abrimos mBlock3, vemos una pantalla como la siguiente:

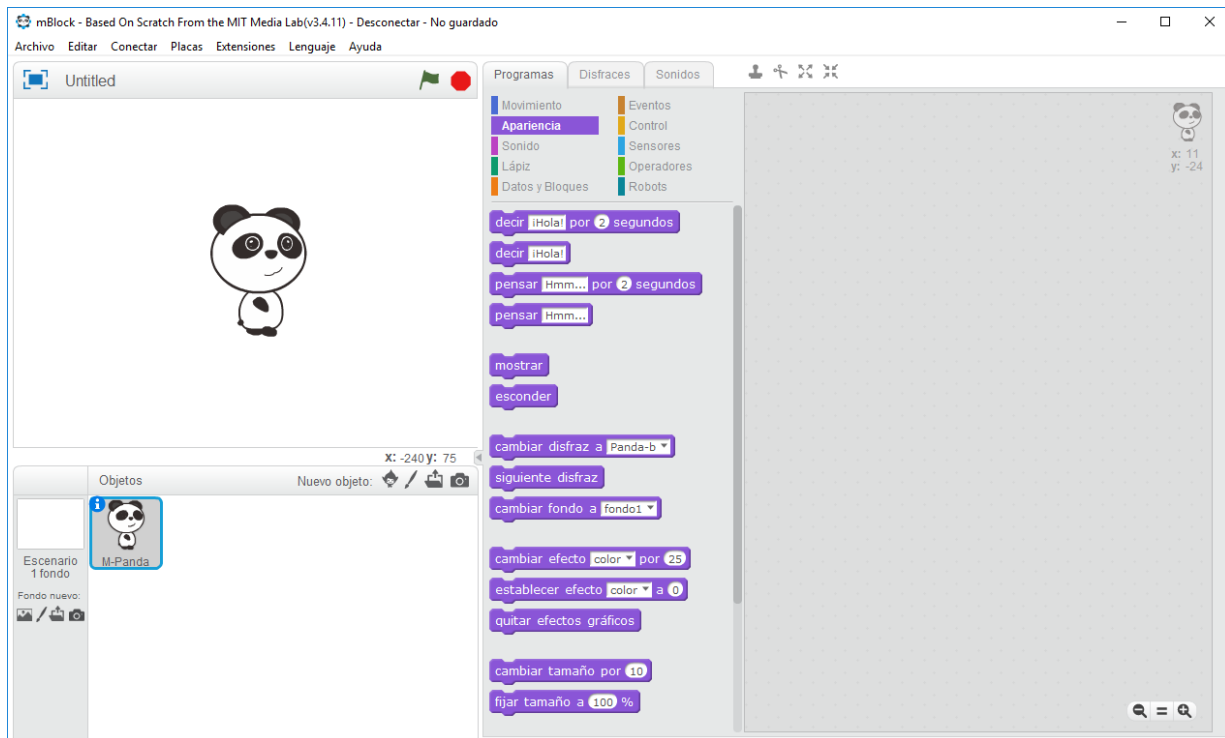


Fig. 2

Para programar un proyecto de Arduino con mBlock3 debemos seleccionar el “Modo Arduino” desde el menú.

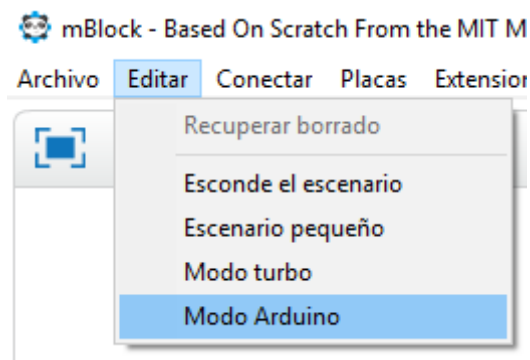


Fig. 3

Al seleccionar este modo, el programa cambiará de aspecto. Se verá un área en el centro que es la que utilizaremos para programar con bloques. A la derecha, se verá un campo donde aparecerá el código escrito que le corresponde a los bloques que están en el centro. Este código se irá escribiendo automáticamente a medida que se vaya armando el programa con los bloques.

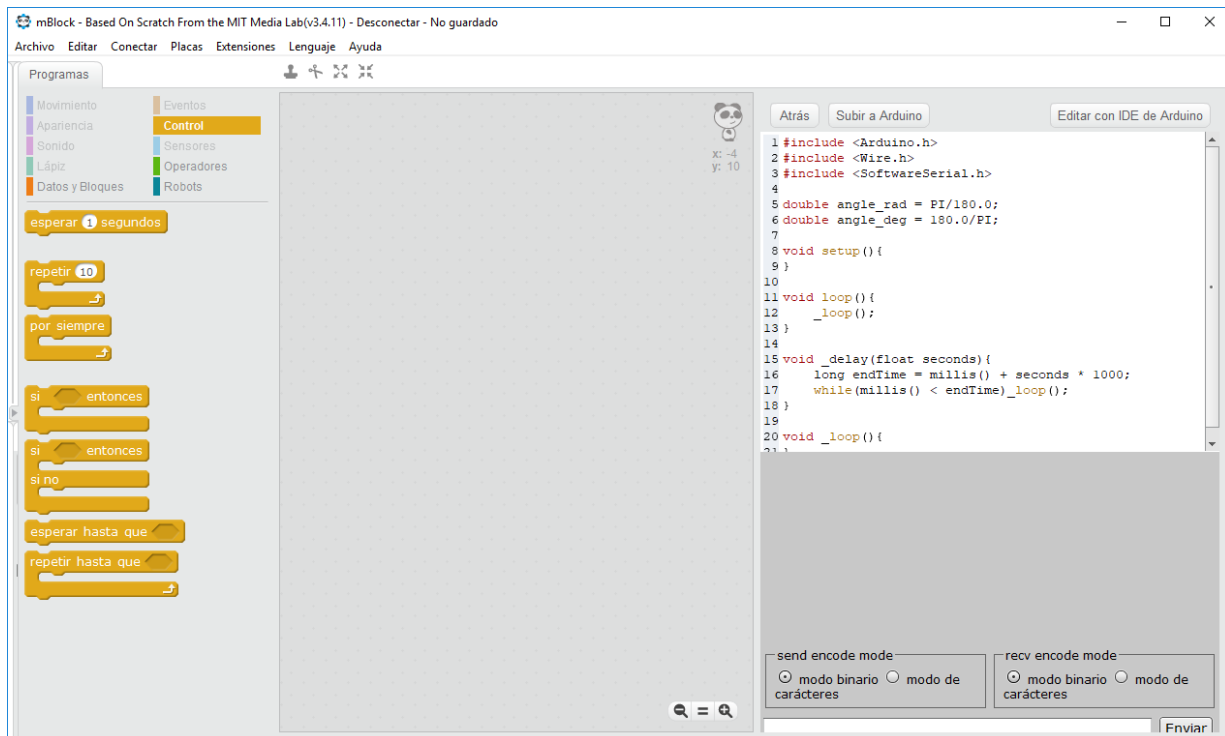


Fig. 4

Los bloques están agrupados por categorías. En este caso, se usarán las siguientes: “**Robots**”, “**Control**”, “**Operadores**”, y “**Datos y Bloques**”. Cuando seleccionamos una de estas categorías, se pueden visualizar todos los bloques que pertenecen a ese grupo.



Fig. 5

Primero, se programará el movimiento del servo de manera que se repita la siguiente secuencia:

- Se moverá el eje del servo hasta completar un ángulo de 90° .
- Se detendrá 2 segundos.
- Volverá a la posición inicial (0°).
- Se detendrá luego de 2 segundos más.

El programa nos quedará así:

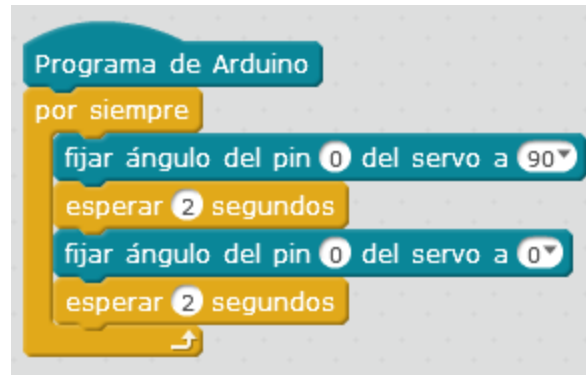


Fig.6

Como se verá a la derecha de la pantalla, el código escrito quedará así:

```
#include <Arduino.h>
#include <Wire.h>
#include <SoftwareSerial.h>

#include <Servo.h>

double angle_rad = PI/180.0;
double angle_deg = 180.0/PI;
Servo servo_0;

void setup(){
    servo_0.attach(0); // init pin
}

void loop(){
    servo_0.write(90); // write to servo
    _delay(2);
    servo_0.write(0); // write to servo
    _delay(2);
    _loop();
}

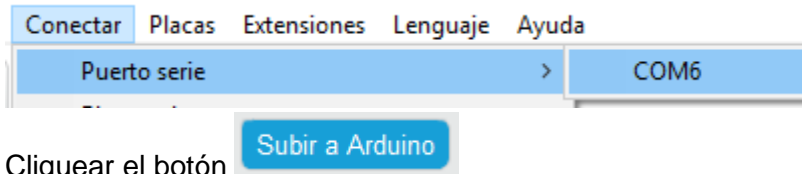
void _delay(float seconds){
    long endTime = millis() + seconds * 1000;
    while(millis() < endTime)_loop();
}

void _loop(){
}
```

Paso 3 - Subir el código a la placa Arduino

Para subir el código de nuestro programa a la placa Arduino, necesitamos:

1. Conectar la placa Arduino a la entrada USB.
2. Chequear que, en el menú "Placas", esté seleccionado "**Arduino Uno**".
3. Seleccionar el puerto serie al que está conectada la placa.



4. Clickear el botón

Al terminar de subir nuestro código, veremos este mensaje:

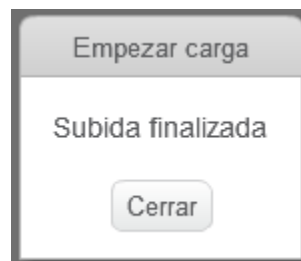
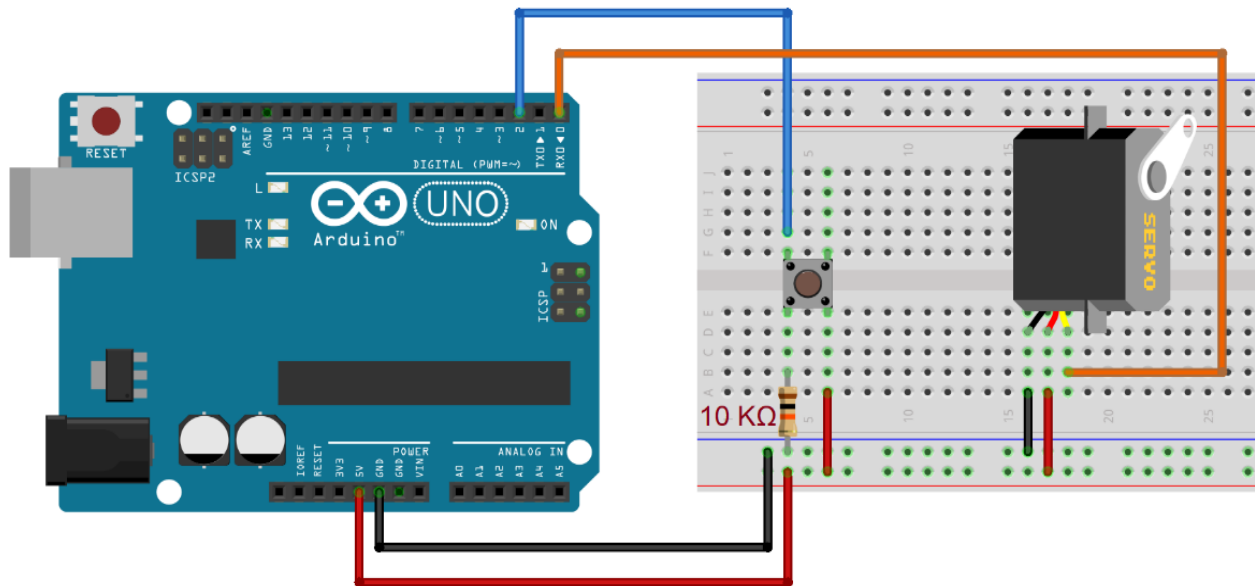


Fig.7

Paso 4 - Mover el servo al presionar un pulsador

En este momento, agregaremos a nuestro sistema un pulsador táctil para que accione el movimiento del servo. El pulsador táctil y la resistencia de 10 K Ω se incluyen como indica el siguiente esquema:



Esquema 2

Ahora modificaremos la programación que realizamos previamente para que el servo se accione únicamente cuando se presione el pulsador.

Más adelante, podremos colocar el pulsador debajo de una plataforma para automóviles que se ubique en la entrada del estacionamiento, de modo que se presione y active la barrera cada vez que ingresa un vehículo. Programaremos nuestro sistema para que:

- La barrera se abra en un ángulo de 90° cada vez que se presione el pulsador, por medio del movimiento del servomotor.
- Una vez que el pulsador haya sido liberado, el programa espere tres segundos y devuelva al servomotor y la barrera a su posición original (0°).

Para llevar a cabo la programación, le asignaremos un valor numérico a cada estado del pulsador. Al estado de reposo (es decir, cuando no está siendo pulsado) le asignaremos el valor "0", y al estado activo (es decir, cuando está siendo pulsado) le asignaremos el valor "1".

El programa debe ser similar al siguiente modelo:

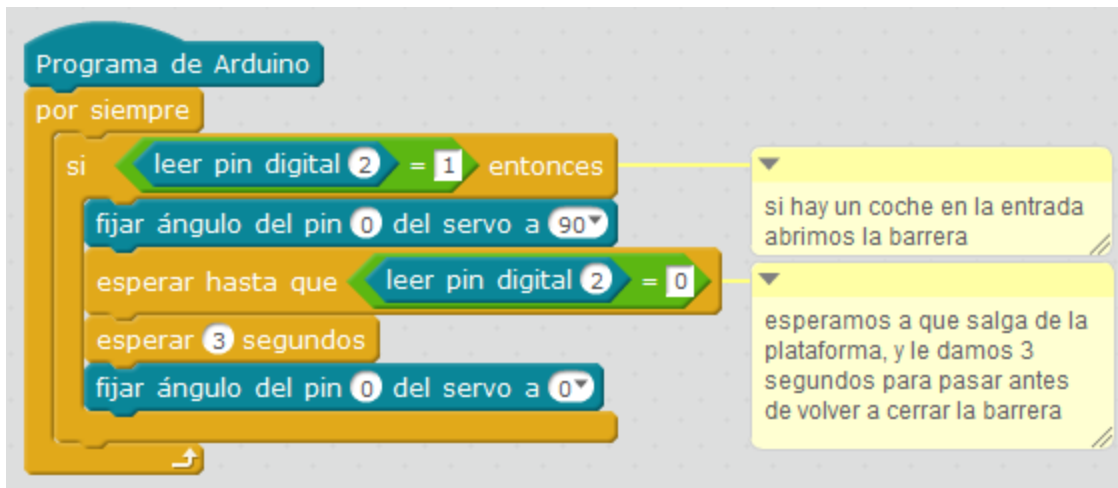


Fig.8

Y el código escrito que se puede ver a la derecha, debería ser similar al siguiente:

```
#include <Arduino.h>
#include <Wire.h>
#include <SoftwareSerial.h>

#include <Servo.h>

double angle_rad = PI/180.0;
double angle_deg = 180.0/PI;
Servo servo_0;

void setup(){
  pinMode(2,INPUT);
  servo_0.attach(0); // init pin
}

void loop(){
  if(((digitalRead(2))==1)){
    servo_0.write(90); // write to servo
    while(!(((digitalRead(2))==0)))
    {
      _loop();
    }
    _delay(3);
    servo_0.write(0); // write to servo
  }
}
```

```

    _loop();
}

void _delay(float seconds){
    long endTime = millis() + seconds * 1000;
    while(millis() < endTime)_loop();
}

void _loop(){
}

```

Paso 5 - Agregar LED de estado

En este momento, se construirá el sistema que indicará si hay lugares disponibles en el estacionamiento o no.

Consideraremos que el estacionamiento cuenta con ocho lugares como capacidad máxima. Cada vez que un vehículo ingrese al estacionamiento, sabremos que hay un lugar menos disponible (es decir, se resta un lugar de la disponibilidad total). Por el contrario, cada vez que un vehículo salga del estacionamiento, sabremos que hay un lugar más disponible (es decir, se suma un lugar a la disponibilidad total).

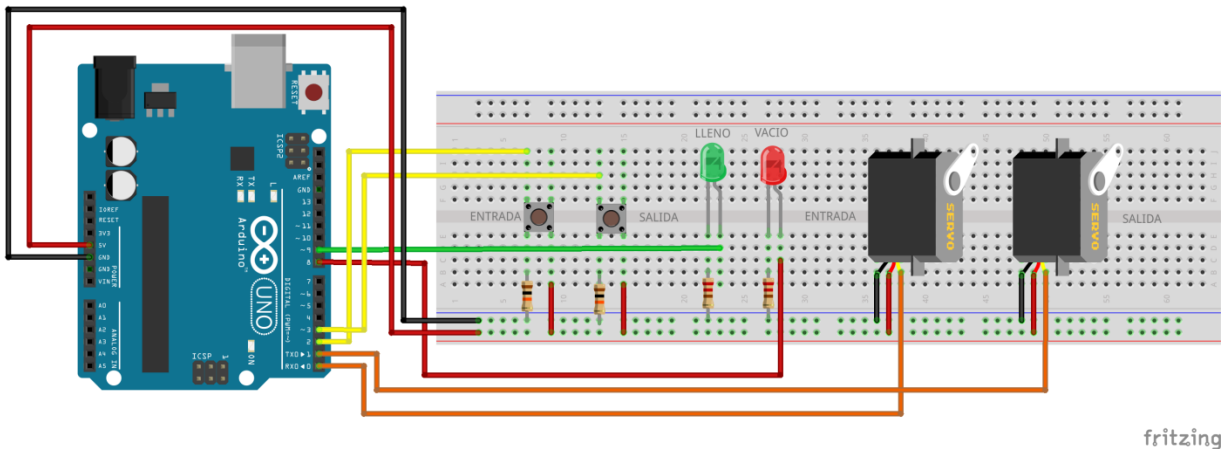
La barrera de salida se construirá con el mismo mecanismo que la barrera de entrada utilizando otro servomotor y otro pulsador. El sistema estará armado de modo que se encienda un LED verde cuando haya lugares disponibles en el estacionamiento y un led rojo cuando estén todos ocupados.

En un LED, la pata larga siempre es la “pata positiva” y es donde se conecta a nuestro *pin*.

- Pata larga positiva
- Pata corta negativa



Comenzamos conectando los LED como indica el siguiente esquema:



Esquema 3

Para que nuestro programa pueda llevar la cuenta de los lugares disponibles, tenemos que crear una variable que almacene ese valor. Esto lo hacemos ingresando en la categoría **“Datos y Bloques”** y clickeando la opción **“Crear una variable”**.



Fig. 9

Le ponemos el nombre *“lugares”* a la variable creada y hacemos click en **“OK”**. Para que funcione correctamente, es importante no utilizar espacios, ni tildes, ni eñes para los nombres de las variables.

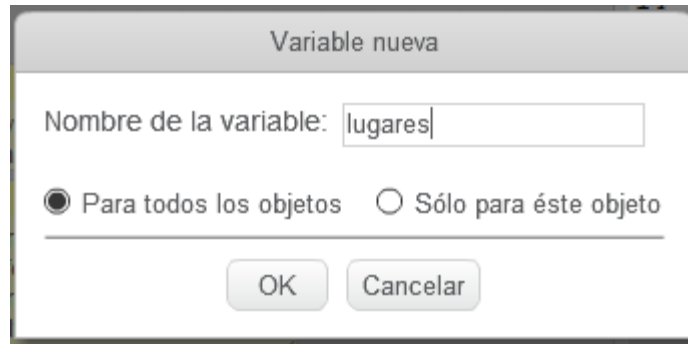


Fig.10

Una vez creada la variable, veremos que contamos con nuevos bloques disponibles que podremos utilizar en nuestro código.



Fig.11

- **lugares** nos permite consultar el valor de la variable.
- **fijar lugares a 0** nos permite asignarle a la variable un valor determinado (en este caso le asignamos el valor "0")
- **cambiar lugares por 1** nos permite incrementar (o disminuir) el valor de la variable en determinada medida (en este caso, en "1")
- **mostrar variable lugares** y **esconder variable lugares** no vamos a utilizarlas porque no corresponden a programas de Arduino.

Ahora que ya conocemos estos nuevos bloques, podemos escribir el programa que nos permitirá llevar la cuenta de los lugares disponibles. Este debe ser similar al siguiente modelo:



Fig.12

Y el código escrito que se puede ver a la derecha, debería ser similar al siguiente:

```
#include <Arduino.h>
#include <Wire.h>
#include <SoftwareSerial.h>

#include <Servo.h>

double angle_rad = PI/180.0;
```

```

double angle_deg = 180.0/PI;
double lugares;
Servo servo_0;
Servo servo_1;

void setup(){
    lugares = 8;
    pinMode(2,INPUT);
    servo_0.attach(0); // init pin
    pinMode(3,INPUT);
    servo_1.attach(1); // init pin
    pinMode(8,OUTPUT);
    pinMode(9,OUTPUT);
}

void loop(){
    if(((digitalRead(2))==1)){
        servo_0.write(90); // write to servo
        while(!(((digitalRead(2))==0)))
        {
            _loop();
        }
        _delay(3);
        servo_0.write(0); // write to servo
        lugares += -1;
    }
    if(((digitalRead(3))==1)){
        servo_1.write(90); // write to servo
        while(!(((digitalRead(3))==0)))
        {
            _loop();
        }
        _delay(3);
        servo_1.write(0); // write to servo
        lugares += 1;
    }
    if((lugares) > (0)){
        digitalWrite(8,0);
        digitalWrite(9,1);
    }else{
        digitalWrite(8,1);
        digitalWrite(9,0);
    }
}

```

```

    _loop();
}

void _delay(float seconds){
    long endTime = millis() + seconds * 1000;
    while(millis() < endTime)_loop();
}

void _loop(){
}

```

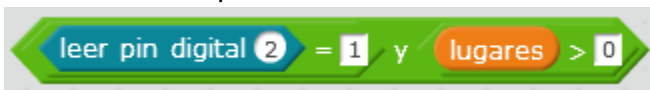
Todavía nos falta agregar una última funcionalidad a nuestro sistema. Necesitamos que el programa impida el ingreso de los vehículos cuando no haya plazas disponibles. Entonces, para abrir la barrera de entrada debemos comprobar tanto que el pulsador de entrada esté presionado como que haya al menos una plaza disponible.

Esto se realiza modificando los bloques de la siguiente manera:

Donde dice:



Lo modificamos por:

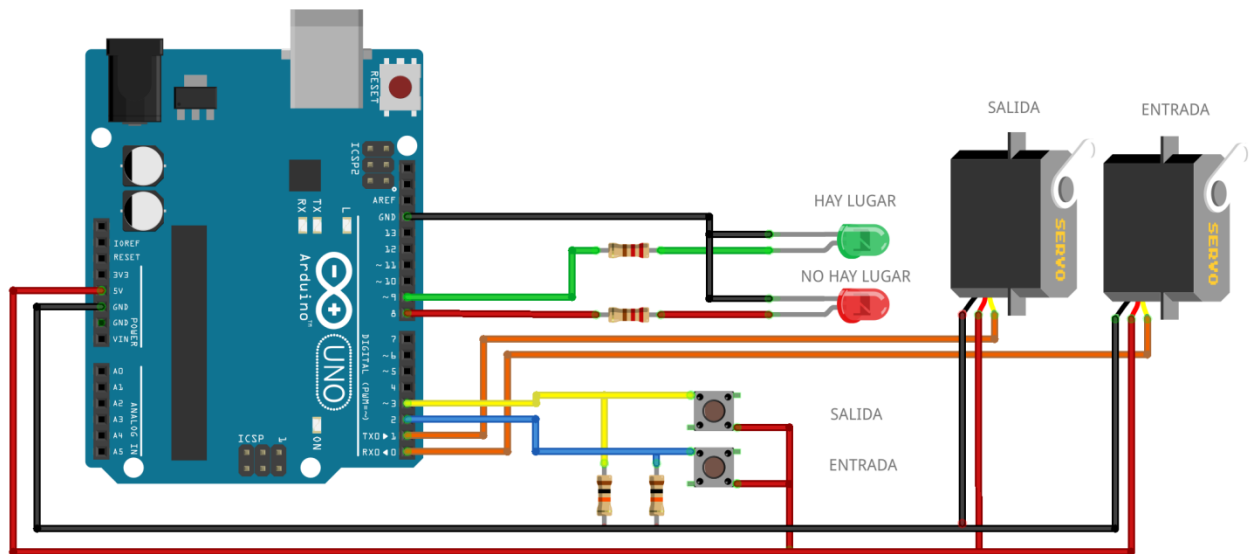


Paso 6 - Armado de maqueta

En este momento, debemos decidir si queremos avanzar al nivel de complejidad intermedio o no. Si decidimos cerrar el proyecto en este nivel, continuamos en este punto para armar la maqueta de nuestro sistema. Si, por el contrario, queremos avanzar con el proyecto, pasamos al próximo nivel directamente. Cabe aclarar en este punto que en el nivel intermedio debido a la complejidad que tiene su conexión proponemos no armar una maqueta sino cerrar el proyecto con las conexiones a la Protoboard.

Si decidimos avanzar con la maqueta, una vez que las conexiones han sido probadas y el sistema funciona correctamente, podemos pasar a armar los circuitos prescindiendo de la Protoboard. Para hacerlo, soldamos cada componente con un cable a la placa Arduino. Luego

colocamos cada pieza en su ubicación en la maqueta, como puede verse en la imagen siguiente:



Esquema 4

Para diseñar la planta del estacionamiento con los estudiantes, se deberá tener en cuenta la ubicación de los servomotores con sus barreras, la ubicación de los pulsadores en la entrada y en la salida, la ubicación de los LED y la distribución de los 8 lugares para los vehículos.

Nivel Intermedio

En algunas ocasiones sucede que muchos autos forman fila para ingresar al estacionamiento. Si bien el sistema indica fácilmente si hay al menos un espacio disponible, no informa cuántos lugares disponibles existen. Por lo tanto, Juana no puede saber si hay espacios suficientes para todos los vehículos que desean entrar. Por eso, decidió mejorar el sistema de su estacionamiento para poder ver la cantidad exacta de lugares libres que quedan.

A partir de esta situación proponemos al grupo agregar a la maqueta un display de 7 segmentos que permita visualizar la cantidad exacta de lugares disponibles en el estacionamiento.

Paso 1 - Subir el código a través de Arduino IDE

La programación por bloques tiene sus ventajas desde un punto de vista didáctico, pero tiene sus límites. Cuando el programa que queremos realizar comienza a ser más complejo podemos encontrarnos con el hecho de que ciertas operaciones no pueden resolverse utilizando bloques, o que hacerlo con este método resulta en algo mucho más engorroso y difícil de leer que si se utilizara el código escrito.

Hasta ahora hemos visto cómo al realizar nuestra programación en bloques se generaba simultáneamente un código escrito en el área lateral derecha. Para esta sección de la actividad se propone trabajar directamente sobre el código, para ello vamos a recurrir a el entorno nativo de Arduino que llamamos Arduino IDE (entorno de desarrollo integrado).

Para ello descarga el Arduino IDE desde el siguiente enlace y luego procede con la instalación del mismo: www.enfoco.net.ar/sd

Veremos que se nos presenta la siguiente interfaz:

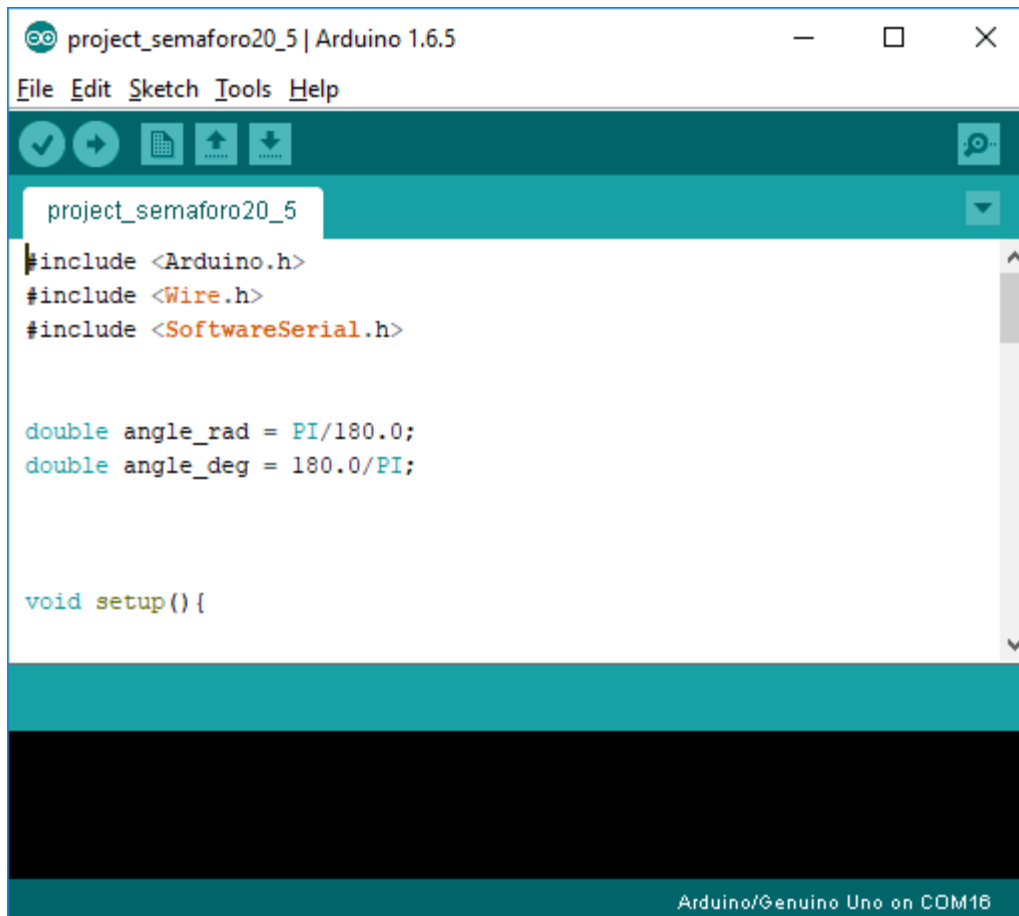


Fig.13

A continuación, se presenta una estructura mínima de un *sketch* (un programa) de Arduino:

```
void setup() {
  // Código de inicialización. Se ejecuta una sola vez.
}

void loop() {
  // Código principal. Se ejecuta repetidamente.
}
```

En líneas generales, un programa de Arduino es:

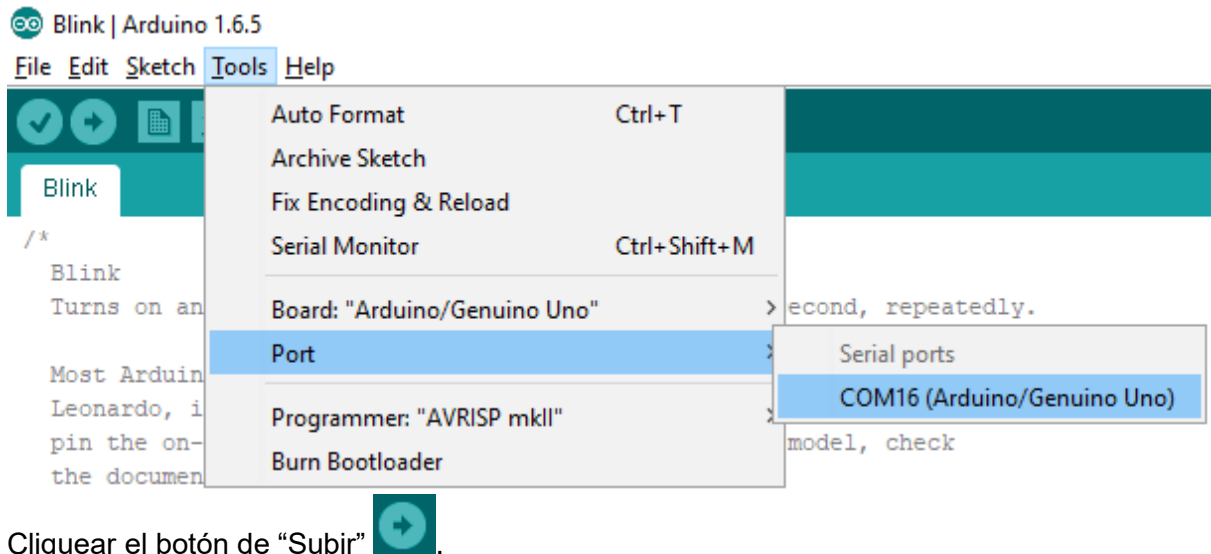
1. Un bloque de código que se ejecuta por única vez al inicializarse el dispositivo. Este bloque de código está contenido dentro de la función “setup” (se coloca dentro de `void setup() { y }`).

2. Un bloque de código que se ejecuta repetidamente luego de la función “setup”. Este bloque de código está contenido dentro de la función “loop” (se coloca dentro de `void loop() { y }`).

Después de `//` se incluyen comentarios para el lector que no tienen ningún efecto en el programa. Estos comentarios sirven para clarificar el código y que sea más fácil de interpretar para otras personas.

Los pasos para subir el código a través del Arduino IDE son similares a los que hemos visto para mBlock3:

1. Conectar la placa a la entrada USB.
2. Chequear que estén seleccionados la placa “Arduino/Genuino Uno” y el puerto serie al que está conectada la placa.



3. Clickear el botón de “Subir” .

Sabremos que nuestro código subió correctamente si en la barra de estado se escribe “Subido”.

Paso 2 - Conectar el *display* de 7 segmentos.

Paso 2 - Conectar el *display* de 7 segmentos.



El *display* de 7 segmentos es un componente que se utiliza para la representación de números en muchos dispositivos electrónicos. Está compuesto de siete led con forma de línea que se pueden encender o apagar individualmente

El *display* de 7 segmentos nos permitirá saber cuántos lugares disponibles tiene el estacionamiento.

Cada segmento está identificado con una letra, como muestra el siguiente dibujo. Así, cuando hablemos del segmento “g” sabremos que nos referimos al segmento horizontal del centro.

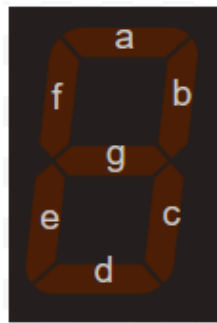
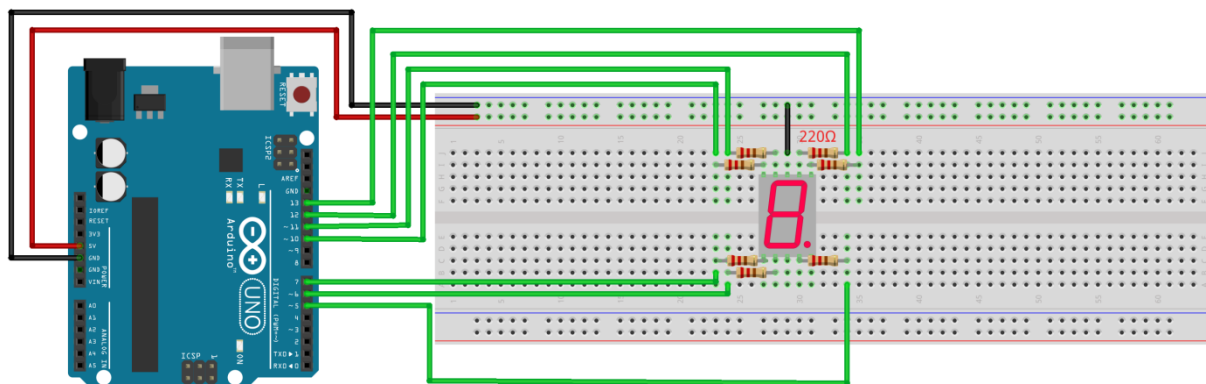


Fig.14

Agregamos el *display* de 7 segmentos a el sistema que ya hemos armado (la figura indica únicamente las conexiones que se agregan para manejar este componente).



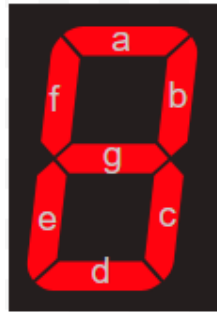
fritzing

Esquema 5

Paso 3 - Programar el *display* de 7 segmentos.

Un *display* de 7 segmentos, en definitiva, son siete LED organizados para representar caracteres. La forma de programarlo es igual al modo de programar el encendido y apagado de los LED en general. Si queremos representar un 8 por ejemplo, debemos encender todos los LED.

Escribimos, entonces, el código que permite encender todos los LED del *display*:



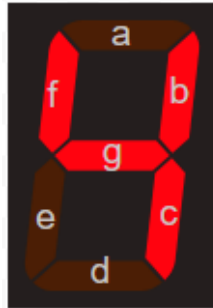
```
// guardamos los números de los pines en variables
// para identificarlos más fácilmente
int a=12;
int b=13;
int c=5;
int d=6;
int e=7;
int f=11;
int g=10;

void setup() {
  // configuramos los pines como salidas
  pinMode(a, OUTPUT);
  pinMode(b, OUTPUT);
  pinMode(c, OUTPUT);
  pinMode(d, OUTPUT);
  pinMode(e, OUTPUT);
  pinMode(f, OUTPUT);
  pinMode(g, OUTPUT);

  // encendemos todos los LED para escribir un 8
  digitalWrite(a, 1);
  digitalWrite(b, 1);
  digitalWrite(c, 1);
  digitalWrite(d, 1);
  digitalWrite(e, 1);
  digitalWrite(f, 1);
  digitalWrite(g, 1);
}
```

```
void loop() {
  // no hay nada que hacer
}
```

Si, en cambio, queremos representar un 4, deberemos encender solamente los segmentos **b**, **c**, **f** y **g**, y apagar los segmentos **a**, **d** y **e**



El código que permitiría escribir un 4 sería el siguiente:

```
//...
digitalWrite(a, 0);
digitalWrite(b, 1);
digitalWrite(c, 1);
digitalWrite(d, 0);
digitalWrite(e, 0);
digitalWrite(f, 1);
digitalWrite(g, 1);
//...
```

Como vamos viendo, cada carácter tiene su particular configuración de segmentos prendidos y apagados, que podemos seguir en la siguiente tabla:

| | a | b | c | d | e | f | g | |
|---|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--|
| 0 | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | | |
| 1 | | <input type="checkbox"/> | <input type="checkbox"/> | | | | | |
| 2 | <input type="checkbox"/> | <input type="checkbox"/> | | <input type="checkbox"/> | <input type="checkbox"/> | | <input type="checkbox"/> | |

| | | | | | | | | |
|---|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--|
| 3 | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | |
| 4 | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | |
| 5 | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | |
| 6 | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | |
| 7 | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | |
| 8 | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | |
| 9 | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | |

Teniendo en cuenta esto, sería muy conveniente para nuestro sistema poder definir en código de programación una tabla que indique qué segmentos se deben prender o apagar para representar un carácter determinado.

Al escribir esta tabla en código veremos que se parece bastante a la que está arriba:

```
// vector de 10x7 con la configuración de
// los segmentos {a,b,c,d,e,f,g} de cada carácter numérico
int segmentos[10][7] = { { 1,1,1,1,1,1,0 }, // 0
                        { 0,1,1,0,0,0,0 }, // 1
                        { 1,1,0,1,1,0,1 }, // 2
                        { 1,1,1,1,0,0,1 }, // 3
                        { 0,1,1,0,0,1,1 }, // 4
                        { 1,0,1,1,0,1,1 }, // 5
                        { 1,0,1,1,1,1,1 }, // 6
                        { 1,1,1,0,0,0,0 }, // 7
                        { 1,1,1,1,1,1,1 }, // 8
                        { 1,1,1,1,0,1,1 } }; // 9
```

Ahora necesitamos una función que “escriba” un carácter numérico determinado en base a la información contenida en esta tabla. Tomando de ejemplo lo que ya hemos visto para escribir un 8 o un 4, la función podría ser similar al modelo siguiente:

```
// escribir_numero recibe un valor numérico (num)
// y enciende/apaga los segmentos correspondientes para representarlo
```

```
// en base a lo que está configurado en la tabla "segmentos"
void escribir_numero(int num){
    digitalWrite(a, segmentos[num][0]);
    digitalWrite(b, segmentos[num][1]);
    digitalWrite(c, segmentos[num][2]);
    digitalWrite(d, segmentos[num][3]);
    digitalWrite(e, segmentos[num][4]);
    digitalWrite(f, segmentos[num][5]);
    digitalWrite(g, segmentos[num][6]);
}
```

Veamos cómo funciona esto. Primero “llamamos” a la función `escribir_numero()` y le asignamos a `num` un valor numérico. En este caso, tomaremos el 2 como ejemplo. Al asignarle ese valor, la función reemplazará `num` por 2 en cada lugar donde se encuentre. De esta manera, donde dice `segmentos[num][0]` dirá `segmentos[2][0]`, lo que representa la posición `[2,0]` del vector `segmentos`. Debemos tener en cuenta en esta parte que el índice (o posición) del primer elemento de un vector es 0 y no 1. De esta manera, `segmentos[2][0]` es el valor del segmento **a** del carácter **2**, `segmentos[2][1]` es el valor del segmento **b** del carácter **2**, y así sucesivamente. De esta manera, al tomar el valor que le asignamos, el programa hará que el *display* encienda o apague sus LED de acuerdo al patrón correspondiente a esa posición.

En las líneas de código las palabras no pueden llevar tilde ni usar la letra Ñ.

Probemos esta función escribiendo los números de 0 a 9 con una demora de 1 segundo entre cada uno. El código completo nos queda como se ve a continuación:

```
// Guardamos los números de los pines en variables
// para identificarlos más fácilmente.
int a=12;
int b=13;
int c=5;
int d=6;
int e=7;
int f=11;
int g=10;
// Vector de 10x7 con la configuración de
// los segmentos {a,b,c,d,e,f,g} de cada carácter numérico.
```

```

int segmentos[10][7] = { { 1,1,1,1,1,1,0 }, // 0
                          { 0,1,1,0,0,0,0 }, // 1
                          { 1,1,0,1,1,0,1 }, // 2
                          { 1,1,1,1,0,0,1 }, // 3
                          { 0,1,1,0,0,1,1 }, // 4
                          { 1,0,1,1,0,1,1 }, // 5
                          { 1,0,1,1,1,1,1 }, // 6
                          { 1,1,1,0,0,0,0 }, // 7
                          { 1,1,1,1,1,1,1 }, // 8
                          { 1,1,1,1,0,1,1 } }; // 9

// escribir_numero recibe un valor numérico (num)
// y enciende/apaga los segmentos correspondientes para representarlo
// en base a lo que está configurado en la tabla "segmentos".
void escribir_numero(int num){
    digitalWrite(a, segmentos[num][0]);
    digitalWrite(b, segmentos[num][1]);
    digitalWrite(c, segmentos[num][2]);
    digitalWrite(d, segmentos[num][3]);
    digitalWrite(e, segmentos[num][4]);
    digitalWrite(f, segmentos[num][5]);
    digitalWrite(g, segmentos[num][6]);
}

void setup() {
    // Configuramos los pines como salidas.
    pinMode(a, OUTPUT);
    pinMode(b, OUTPUT);
    pinMode(c, OUTPUT);
    pinMode(d, OUTPUT);
    pinMode(e, OUTPUT);
    pinMode(f, OUTPUT);
    pinMode(g, OUTPUT);
}

void loop()
{
    // de n=0 a n=9
    for (int n = 0; n < 10; n++)
    {
        // Escribir el valor de n en el display.
        escribir_numero(n);
        // Esperar 1 segundo (1000 ms).
    }
}

```

```

    delay(1000);
  }
}

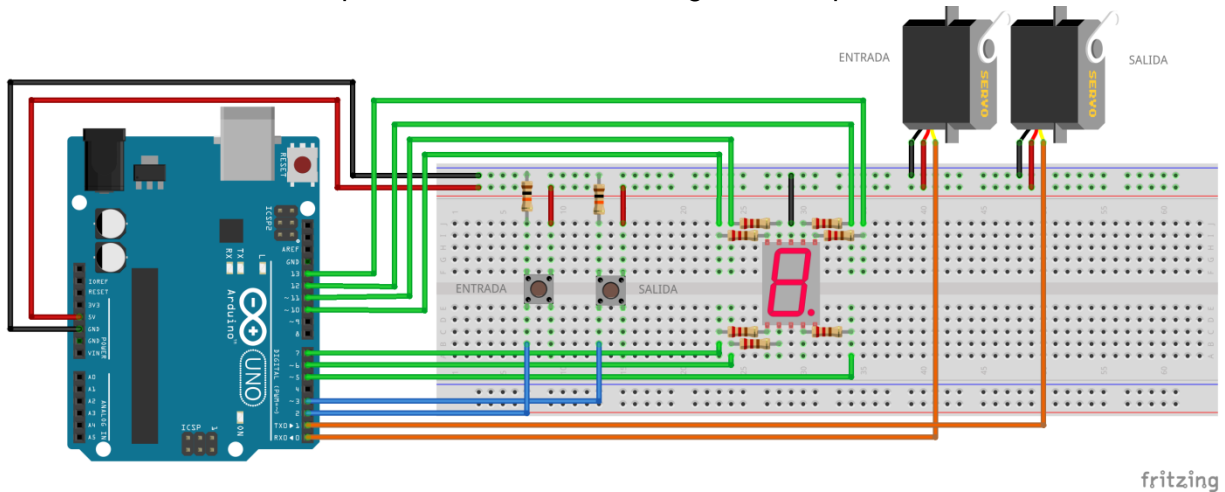
```

Luego de subir el código anterior a la placa Arduino, deberíamos ver cómo el *display* va mostrando los números de 0 a 9 y, al finalizar la secuencia, vuelve a comenzar.

Paso 4 - Incorporar el *display* de 7 segmentos al programa del estacionamiento

Ahora que tenemos este *display* y sabemos cómo programarlo, podemos conectarlo a nuestro sistema para que nos indique cuántas plazas libres quedan en el estacionamiento y reemplazar los LED rojo y verde que habíamos puesto antes.

Conectamos todos los componentes como indica el siguiente esquema:



Esquema 6

Incorporando la visualización de las plazas disponibles, nuestro código queda como se ve a continuación:

```

#include <Servo.h>
// Declaramos nuestro objeto servomotor
Servo servoEntrada;
Servo servoSalida;
// En esta variable llevamos la cuenta de los lugares disponibles.
int lugares = 8;

```

```

// Guardamos los números de los pines en variables
// para identificarlos más fácilmente.
int a=12;
int b=13;
int c=5;
int d=6;
int e=7;
int f=11;
int g=10;
// Vector de 10x7 con la configuración de
// los segmentos {a,b,c,d,e,f,g} de cada carácter numérico.
int segmentos[10][7] = { { 1,1,1,1,1,1,0 }, // 0
                        { 0,1,1,0,0,0,0 }, // 1
                        { 1,1,0,1,1,0,1 }, // 2
                        { 1,1,1,1,0,0,1 }, // 3
                        { 0,1,1,0,0,1,1 }, // 4
                        { 1,0,1,1,0,1,1 }, // 5
                        { 1,0,1,1,1,1,1 }, // 6
                        { 1,1,1,0,0,0,0 }, // 7
                        { 1,1,1,1,1,1,1 }, // 8
                        { 1,1,1,1,0,1,1 }}; // 9

// escribir_numero recibe un valor numérico (num)
// y enciende/apaga los segmentos correspondientes para representarlo
// en base a lo que está configurado en la tabla "segmentos".
void escribir_numero(int num){
    digitalWrite(a, segmentos[num][0]);
    digitalWrite(b, segmentos[num][1]);
    digitalWrite(c, segmentos[num][2]);
    digitalWrite(d, segmentos[num][3]);
    digitalWrite(e, segmentos[num][4]);
    digitalWrite(f, segmentos[num][5]);
    digitalWrite(g, segmentos[num][6]);
}

void setup() {
    // Inicializamos el servomotor de entrada conectado al pin digital 2
    servoEntrada.attach(0);
    // y lo llevamos a la posición de 0 grados.
    servoEntrada.write(0);

    // Inicializamos el servomotor de salida conectado al pin digital 3
    servoSalida.attach(1);
}

```

```

// y lo llevamos a la posición de 0 grados.
servoSalida.write(0);

// Configuramos los pines del display como salidas.
pinMode(a, OUTPUT);
pinMode(b, OUTPUT);
pinMode(c, OUTPUT);
pinMode(d, OUTPUT);
pinMode(e, OUTPUT);
pinMode(f, OUTPUT);
pinMode(g, OUTPUT);

}

void loop() {

// Si hay un coche en la entrada
// es decir, si el pulsador de entrada está siendo presionado
if(lugares>=1 && digitalRead(2)==HIGH){
    // subimos la barrera
    // moviendo el servomotor a la posición de 90 grados.
    servoEntrada.write(90);
    // Mientras el coche sigue en la plataforma, esperamos.
    while(digitalRead(2)==HIGH){}
    // Esperamos 3 segundos a que haya terminado de pasar.
    delay(3000);
    // Volvemos a bajar la barrera, llevando el servomotor a 0 grados.
    servoEntrada.write(0);
    // Restamos un lugar al total de lugares disponibles.
    lugares = lugares - 1;
    // Escribir el valor de la cantidad de lugares disponibles en el
display.
    escribir_numero(lugares);
}

// Si hay un coche a la salida
// es decir, si el pulsador de salida está presionado
if(digitalRead(3)==HIGH){
    // subimos la barrera
    // moviendo el servomotor a la posición de 90 grados.
    servoSalida.write(90);
    // Mientras el coche sigue en la plataforma, esperamos.
    while(digitalRead(3)==HIGH){}

```



```
// Esperamos 3 segundos a que haya terminado de pasar.  
delay(3000);  
// Volvemos a bajar la barrera, llevando el servomotor a 0 grados.  
servoSalida.write(0);  
// Incrementamos en 1 el valor total de lugares disponibles.  
lugares = lugares + 1;  
// Escribir el valor de la cantidad lugares disponibles en el display.  
escribir_numero(lugares);  
}  
  
}
```

Nivel Avanzado

Los primos de Juana tienen otro estacionamiento en el mismo barrio. Cuando no tienen lugares disponibles, les sugieren a los conductores que se acercan que vayan al estacionamiento de Juana. En esos momentos la llaman a Juana para consultarle si tiene lugares disponibles. Ella hace lo mismo cada vez que su estacionamiento está completo.

Para agilizar el proceso cada vez que esto sucede y evitarse la necesidad de llamar por teléfono, Juana decidió modificar el sistema de control que tiene instalado de manera que permita visualizar la cantidad de espacios disponibles desde un dispositivo móvil. Así sus primos podrían acceder a esa información desde su estacionamiento.

En esta situación se propone agregar la conectividad IoT que permite visualizar la información a través de una web o un dispositivo móvil.

Paso 1 - Introducción a Internet de las Cosas (IoT)

Internet de las Cosas (en inglés *Internet of Things*, abreviado IoT) es un concepto que refiere a la interconexión digital de objetos cotidianos con internet. Esta interconexión puede tener diversas funciones. Por ejemplo, puede utilizarse para monitorear la temperatura de un ambiente, enviando los datos obtenidos por un sensor a una central donde se recopile la información. De esta manera podría visualizarse en un dispositivo móvil la temperatura de un laboratorio, de un invernadero o de una sala de un hospital.

Para poder incorporar **IoT** a nuestro proyecto es necesario:

1. Un dispositivo capaz de conectarse a internet.

2. Un servidor que reciba y aloje los datos.

Existen diversas formas de lograr el cometido de registrar y almacenar los datos del sistema de tanques construido. En este caso, se detallará cómo hacerlo con un módulo OBloq de DFRobot, y con los servidores de Adafruit.

El módulo UART OBLOQ es un dispositivo WiFi a serie pensado para desarrolladores no profesionales. Permite enviar y recibir datos mediante los protocolos HTTP y MQTT.

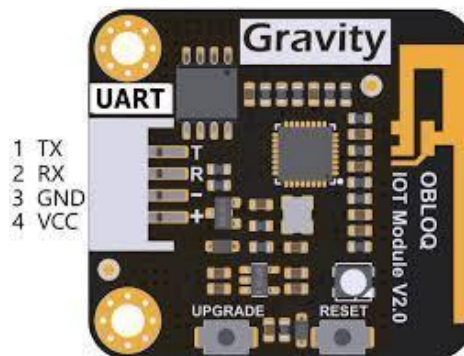


Fig.15

Paso 2 - Crear un Panel de Control

En primer lugar, se explicará cómo crear un Panel de Control en Adafruit. Luego, se verá cómo vincular los controles del Panel con los datos que se intercambian con el dispositivo. Primero debemos crear una cuenta de usuario en io.adafruit.com.

Una vez que ingresamos con nuestro usuario, creamos un nuevo panel haciendo click en el botón "Actions" y seleccionamos "Create a New Dashboard".

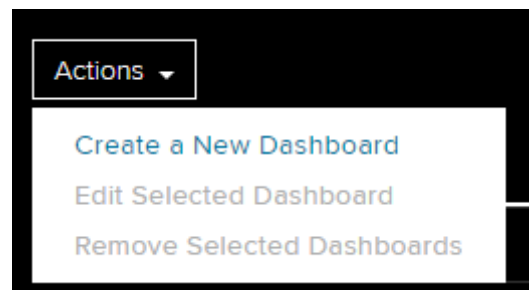


Fig.16

Luego designamos un nombre y una descripción. Y al finalizar presionamos el botón "Create".

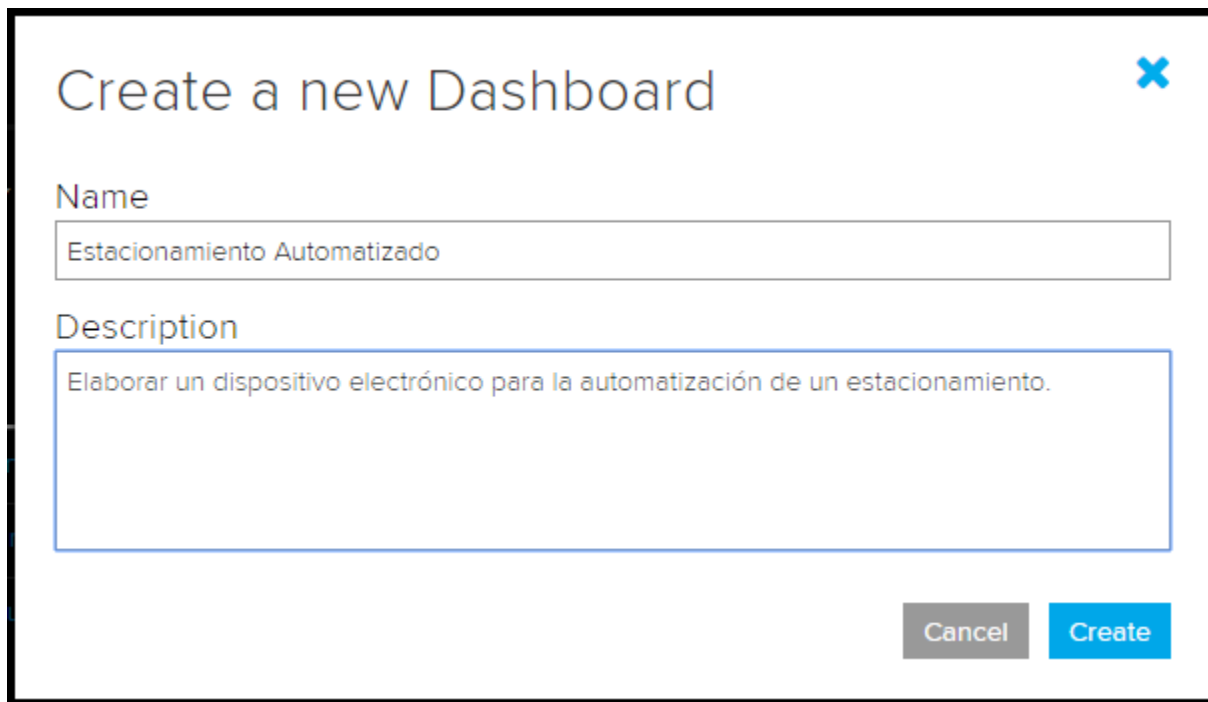
A screenshot of a web application dialog box titled "Create a new Dashboard" with a blue 'X' close button in the top right corner. The dialog contains two input fields: "Name" with the text "Estacionamiento Automatizado" and "Description" with the text "Elaborar un dispositivo electrónico para la automatización de un estacionamiento." At the bottom right, there are two buttons: a grey "Cancel" button and a blue "Create" button.

Fig.17

Seguidamente hacemos click en el nuevo panel creado y veremos una pantalla vacía.

Podemos comenzar a agregar bloques haciendo click en



Fig.18

Veremos una serie de controles posibles como en la siguiente imagen:

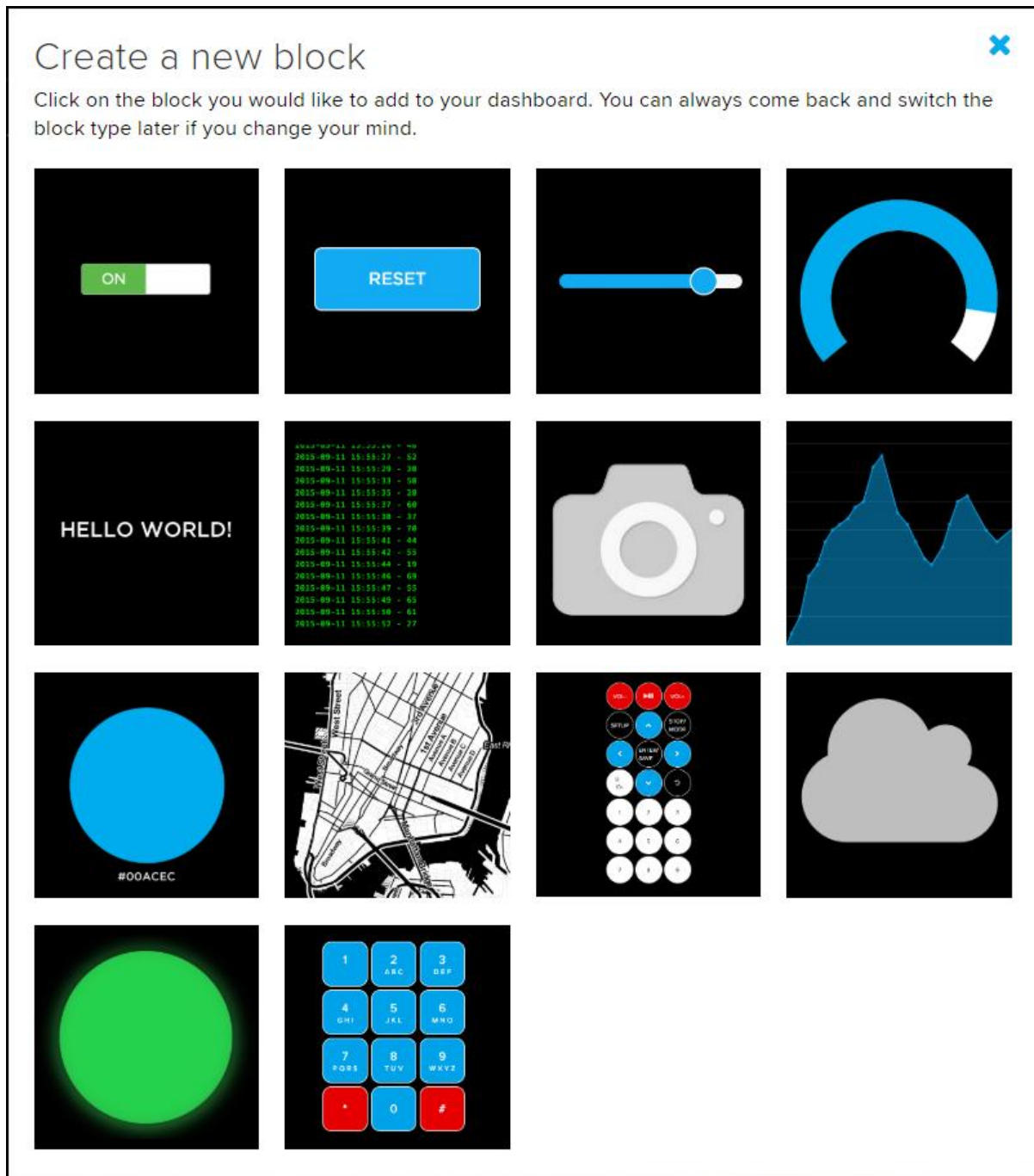


Fig.19

Para nuestro sistema del estacionamiento podríamos seleccionar un *Gauge* para mostrar las plazas que se encuentran ocupadas en nuestro estacionamiento.



Fig.20

Cuando agregamos un control al panel debemos asociarlo a un “feed”.

Un **feed** es una fuente de datos en la que uno puede publicar así como también se puede suscribir para recibir los datos de cierto feed.

Llamamos al primer *feed* “estacionamiento1” y le damos al botón “Create”. En él publicaremos, desde nuestro dispositivo, la información sobre el estado del primer semáforo.

Choose feed

Indicator: A simple on/off indicator lamp. Feed values are compared using the given conditions. If the conditions are true, then "On Color" is used, if false, "Off Color". All values are assumed to be numeric for comparison. If the current feed value can't be converted to a number, it will be treated as a string.

If you have lot of feeds, you may want to use the search field. You can also create a feed quickly below.

Create

| Group / Feed | Last value | Recorded |
|---------------------------------------|------------|----------|
| <input type="checkbox"/> Welcome Feed | | 2 days |

< Previous step Next step >

Fig. 21

Luego de crearlo, seleccionamos el feed creado y hacemos click en “Next step” (paso siguiente) para configurar nuestro control.

Choose feed ✕

Indicator: A simple on/off indicator lamp. Feed values are compared using the given conditions. If the conditions are true, then "On Color" is used, if false, "Off Color". All values are assumed to be numeric for comparison. If the current feed value can't be converted to a number, it will be treated as a string.

If you have lot of feeds, you may want to use the search field. You can also create a feed quickly below.

Create

| Group / Feed | Last value | Recorded |
|--|------------|-------------|
| <input checked="" type="checkbox"/> estacionamiento1 | | 3 hours ago |

< Previous step Next step >

Fig. 22

Este *feed* tendrá el estado del estacionamiento expresado con un indicador numérico y una lectura analógica que irá variando de acuerdo a la cantidad de plazas ocupadas.

Podemos incluir un título, y colores que indiquen cuan lleno o vacío está nuestro estacionamiento.

Block settings

In this final step, you can give your block a title and see a preview of how it will look. Customize the look and feel of your block with the remaining settings. When you are ready, click the "Create Block" button to send it to your dashboard.

Block Title (optional)

Estacionamiento 1

Gauge Min Value

0

Gauge Max Value

8

Gauge Width

25px

Gauge Label

plazas ocupadas

Low Warning Value

Optional. If no low warning value is given, the gauge will only change color when the value is out of bounds.

High Warning Value

Optional. If no high warning value is given, the gauge will only change color when the value is out of bounds.

Block Preview

Estacionamiento 1

5

plazas ocupadas

0

8

Gauge

A gauge is a read only block type that shows a fixed range of values.

Test Value

5

Published Value

0 bytes

Previous step

Create block

Fig.23

Al finalizar la configuración, hacemos click en “Create block” (crear bloque) para completar la operación.

Si se quiere, podemos modificar el tamaño y la ubicación de los bloques haciendo click en la “rueda de configuración”.

38

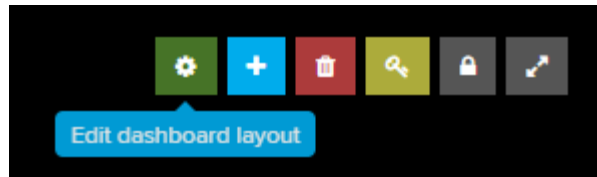


Fig.24

En este caso, registraremos en nuestro *dashboard* el estado de dos estacionamientos. El panel permite publicar y llevar adelante el registro del estado de tantas maquetas como deseemos armar (esto es útil si trabajamos en equipo, para que todos puedan ver el estado de todas las maquetas). Para sumar un nuevo estacionamiento al *dashboard* repetimos el procedimiento que llevamos adelante para “estacionamiento1”, pero ahora lo nombramos “estacionamiento2”. Se debería ver algo similar a lo siguiente:

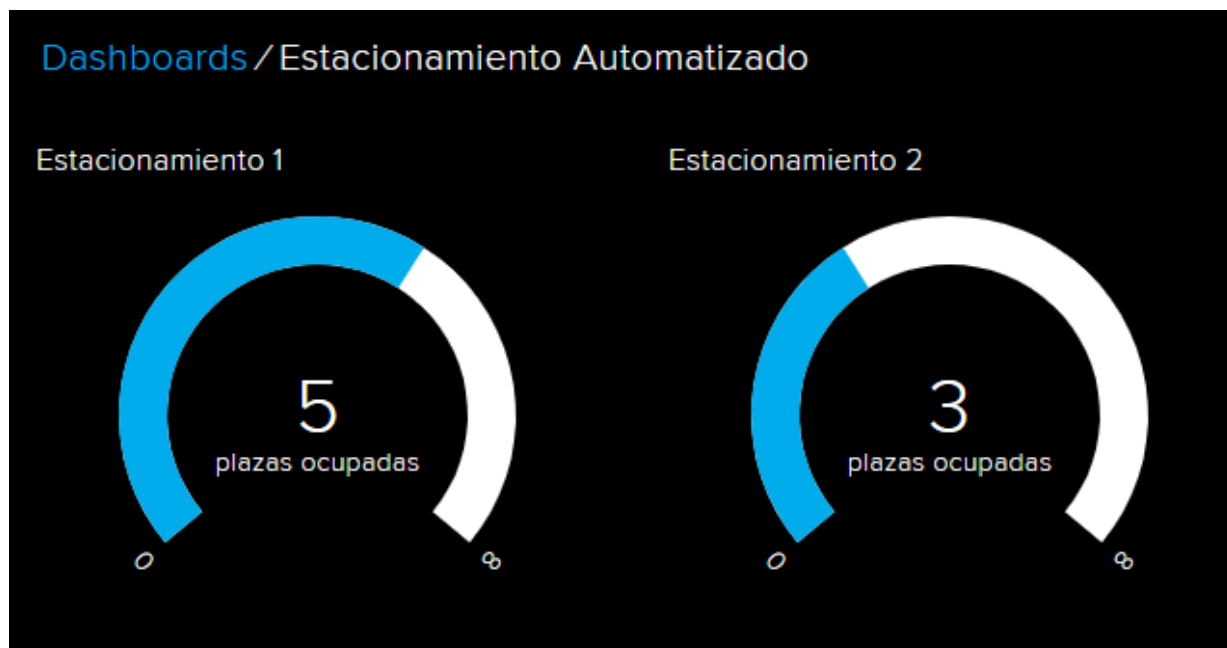
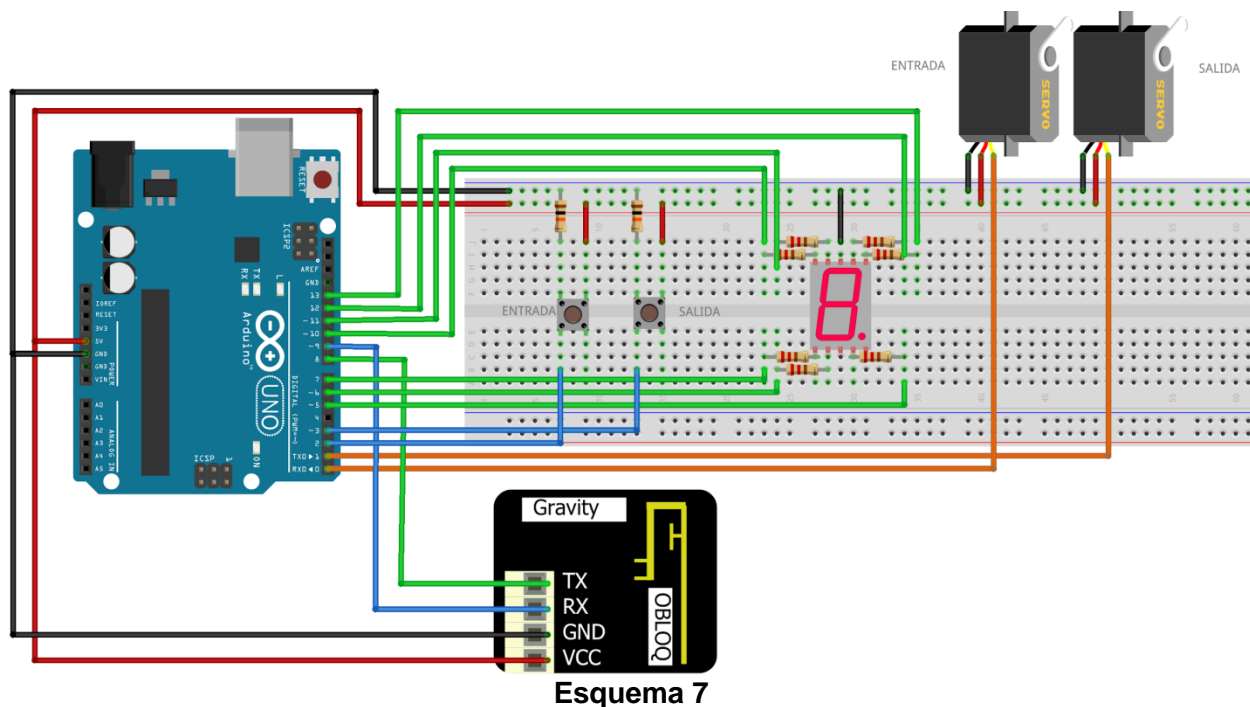


Fig.25

Una vez realizado el panel, publicaremos en el *feed* los estados de nuestros estacionamientos para poder monitorearlos de manera remota. Minimice la pantalla, luego sacaremos información de ella.

Paso 3 - Conectar Módulo OBloq

Conectamos el módulo OBloq como se muestra en el esquema siguiente:



Paso 4 - Arduino IDE

La programación por bloques tiene sus ventajas desde un punto de vista didáctico pero cuando el programa crece en complejidad puede resultar poco práctico. A menudo podemos encontrarnos con el hecho de que ciertas operaciones no pueden resolverse utilizando bloques o que hacerlo con este método resulta más engorroso y difícil de interpretar que si se utilizara el código escrito.

Hasta ahora hemos visto cómo al realizar nuestra programación en bloques se generaba simultáneamente un código escrito en el área lateral derecha. Para esta sección de la actividad se propone trabajar directamente sobre el código, para ello vamos a recurrir al entorno nativo de Arduino que llamamos Arduino IDE (entorno de desarrollo integrado).

Paso 5 – Código bloqueante y IoT

Antes de comenzar a utilizar IoT debemos hacer una aclaración con respecto a la función `_delay()` que figura en el código que usamos hasta ahora. Esta función brinda un tiempo de espera al sistema que puede utilizarse con varios fines. Suele utilizarse bastante en las primeras aproximaciones a la programación, ya que su comportamiento resulta fácil de comprender y su programación no requiere más que una línea de código. Sin embargo, esta función tiene una complicación, dado que genera un “código bloqueante”. Esto

significa que, cuando el programa entra en esa función, se detiene todo el procesamiento hasta que se cumpla el tiempo indicado. En otras palabras, cuando el programa entra al *delay* queda “colgado” por el período de tiempo establecido.

Al utilizar IoT, es conflictivo utilizar código “bloqueante”, ya que al detenerse el procesamiento se impide también que el sistema realice otras operaciones que funcionan en simultáneo. Por ejemplo, las tareas de publicación y el mantenimiento constante de la conexión a internet.

Para evitar estos problemas, se puede utilizar una alternativa de código “no bloqueante”, como la función `millis()`. Esta función arroja un valor sobre un conteo de tiempo, que se realiza desde el momento en que se inicia el sistema. Es decir, funciona como un cronómetro (en milisegundos) que, cada vez que es consultada desde el código, “devuelve” el valor en el que se encuentra. De esta manera podemos pedirle al sistema que informe cuánto tiempo transcurrió desde el inicio de las operaciones para dar indicaciones temporales sobre una tarea, sin detener todas las demás.

En este proyecto en particular, el código bloqueante se utiliza muy poco y aparece en pocas situaciones. Por lo tanto, para no complejizar la programación, podremos mantener el código como lo veníamos programando. En otros proyectos de esta serie encontrarán que se propone el reemplazo del código por otras funciones no bloqueantes.

Paso 6 - Programación IoT

Utilizaremos la librería `ObloqAdafruit` para informar a Adafruit cada vez que cambie el estado del semáforo. Podremos monitorear este estado desde el Panel de Control que hemos creado.

En primer lugar debemos instalar la librería en el Arduino IDE. Para esto debemos ingresar al menú Programa > Incluir Librería > Gestionar Librerías.

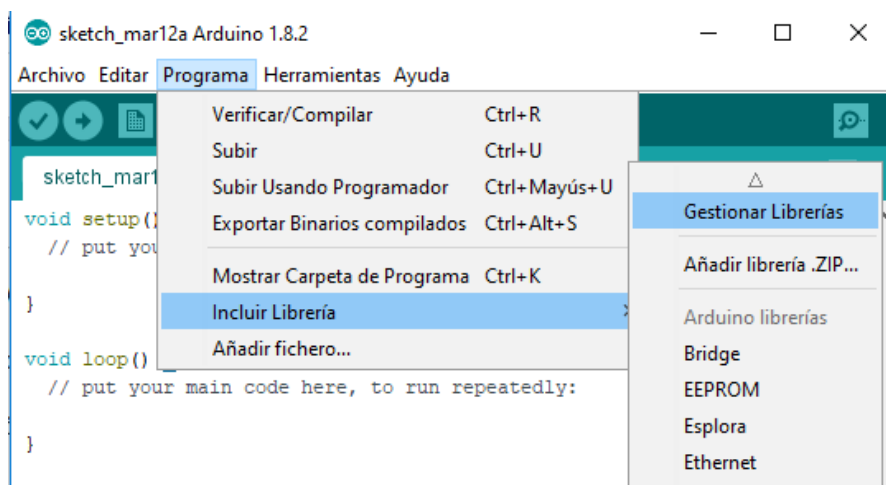


Fig. 26

Se abrirá una ventana con un buscador en margen superior. Debemos escribir Obloq, seleccionar la librería ObloqAdafruit y apretar el botón Instalar.

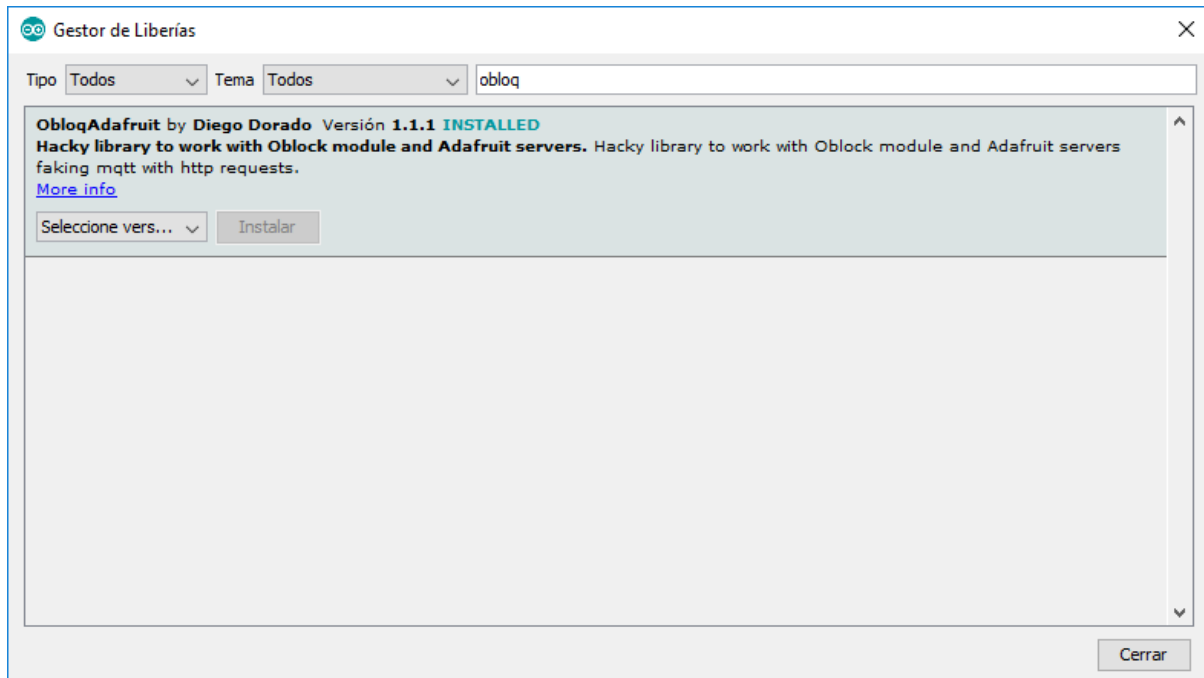


Fig. 27

En general las librerías traen códigos de ejemplo como referencia. Abrimos el ejemplo “Publicar” ubicado en Archivo > Ejemplos > ObloqAdafruit > Publicar.

Debemos reemplazar el SSID de la WiFi, su password, el IO_USERNAME e IO_KEY que son nuestras credenciales que copiaremos de Adafruit. Para ello maximizamos la pantalla de Adafruit y hacemos click en el ícono de la “llave”.

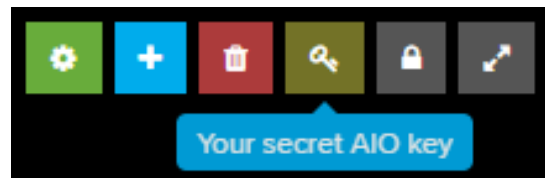



Fig. 28

Copiamos el código que nos ofrece para Arduino, con nuestro usuario y key.

YOUR AIO KEY

Your Adafruit IO key should be kept in a safe place and treated with the same care as your Adafruit username and password. People who have access to your AIO key can view all of your data, create new feeds for your account, and manipulate your active feeds.

If you need to regenerate a new AIO key, all of your existing programs and scripts will need to be manually changed to the new key.



Username

Active Key
REGENERATE AIO KEY

[Hide Code Samples](#)

Arduino

```
#define IO_USERNAME "usuario_aio"
#define IO_KEY      "1234cfdd29a244b6b049abb07727c117"
```

Fig. 29

Estos datos aparecen en el código de la siguiente manera:

```
#define IO_USERNAME "usuario_adafruit"
#define IO_KEY      "key_adafruit"
```

Se deberán reemplazar en esas dos líneas el usuario y key por los que se hayan obtenido en Adafruit. Por ejemplo:

```
#define IO_USERNAME "usuario_aio"
#define IO_KEY      "1234cfdd29a244b6b049abb07727c117"
```

También modificaremos `softSerial(10,11)` por `softSerial(8,9)` ya que así es como lo conectamos en nuestra placa, quedándonos el siguiente código:

```
#include "SoftwareSerial.h"
#include "ObloqAdafruit.h"

// Indicamos conexión de wifi.
#define WIFI_SSID      "SSID_de_Wifi"
#define WIFI_PASSWORD  "PWD_de_WIFI"

// Copiamos las credenciales obtenidas anteriormente en Adafruit.
```

```

#define IO_USERNAME    "usuario_adafruit"
#define IO_KEY         "key_adafruit"

SoftwareSerial softSerial(8,9);
ObloqAdafruit olq(&softSerial,WIFI_SSID,WIFI_PASSWORD,IO_USERNAME,IO_KEY);

```

El *setup* debe incluir la línea de inicialización del softwareSerial:

```

void setup()
{
    softSerial.begin(9600);
}

```

Se debe agregar también la función de actualización o “update”: `olq.update()`. Por esto es importante que nuestro código no sea bloqueante.

```

void loop()
{
    olq.update();
    // ..
    // ..
}

```

En esta instancia deberemos modificar el programa para que publique en Adafruit en nuestros *feeds*.

Se debe reemplazar el código que se venía utilizando por el que se presenta a continuación, que incluye tanto el funcionamiento del dispositivo como la publicación de los valores en *Adafruit*.

Finalmente, nuestro programa de estacionamiento con IoT queda como sigue:

```

// Incluimos las librerías que usaremos.
#include <Servo.h>
#include "SoftwareSerial.h"
#include "ObloqAdafruit.h"

// Indicamos conexión de wifi.

```

```

#define WIFI_SSID      "SSID_de_Wifi"
#define WIFI_PASSWORD  "PWD_de_WIFI"

// Copiamos las credenciales obtenidas anteriormente en Adafruit.
#define IO_USERNAME    "usuario_adafruit"
#define IO_KEY         "key_adafruit"

// Definimos el nombre del feed al que publicaremos
// (cambiar por estacionamiento2 para el segundo estacionamiento).
#define IO_FEED        "estacionamiento1"

// Indicamos a qué pines se conecta el módulo OBloq.
SoftwareSerial softSerial(8,9);
// Declaramos nuestro objeto IoT.
ObloqAdafruit olq(&softSerial,WIFI_SSID,WIFI_PASSWORD,IO_USERNAME,IO_KEY);

// Declaramos nuestro objeto servomotor
Servo servoEntrada;
Servo servoSalida;
// En esta variable llevamos la cuenta de los lugares disponibles.
int lugares = 8;
// Guardamos los números de los pines en variables
// para identificarlos más fácilmente.
int a=12;
int b=13;
int c=5;
int d=6;
int e=7;
int f=11;
int g=10;
// Vector de 10x7 con la configuración de
// los segmentos {a,b,c,d,e,f,g} de cada carácter numérico.
int segmentos[10][7] = { { 1,1,1,1,1,1,0 }, // 0
                          { 0,1,1,0,0,0,0 }, // 1
                          { 1,1,0,1,1,0,1 }, // 2
                          { 1,1,1,1,0,0,1 }, // 3
                          { 0,1,1,0,0,1,1 }, // 4
                          { 1,0,1,1,0,1,1 }, // 5
                          { 1,0,1,1,1,1,1 }, // 6
                          { 1,1,1,0,0,0,0 }, // 7
                          { 1,1,1,1,1,1,1 }, // 8
                          { 1,1,1,1,0,1,1 } }; // 9

```

```

// escribir_numero recibe un valor numérico (num)
// y enciende/apaga los segmentos correspondientes para representarlo
// en base a lo que está configurado en la tabla "segmentos".
void escribir_numero(int num){
    digitalWrite(a, segmentos[num][0]);
    digitalWrite(b, segmentos[num][1]);
    digitalWrite(c, segmentos[num][2]);
    digitalWrite(d, segmentos[num][3]);
    digitalWrite(e, segmentos[num][4]);
    digitalWrite(f, segmentos[num][5]);
    digitalWrite(g, segmentos[num][6]);
}

void setup() {
    // Inicializamos el servo de entrada conectado al pin digital 0
    servoEntrada.attach(0);
    // y lo llevamos a la posición de 0 grados.
    servoEntrada.write(0);

    // Inicializamos el servo de salida conectado al pin digital 1
    servoSalida.attach(1);
    // y lo llevamos a la posición de 0 grados.
    servoSalida.write(0);

    // Configuramos los pines del display como salidas.
    pinMode(a, OUTPUT);
    pinMode(b, OUTPUT);
    pinMode(c, OUTPUT);
    pinMode(d, OUTPUT);
    pinMode(e, OUTPUT);
    pinMode(f, OUTPUT);
    pinMode(g, OUTPUT);

    // Escribir el valor de lugares inicial en el display.
    escribir_numero(lugares);

    // Inicializamos la comunicación con el módulo OBloq.
    softSerial.begin(9600);
}

void loop() {

    // Si hay un coche en la entrada y hay lugares disponibles

```

```

if(lugares>=1 && digitalRead(2)==HIGH){
    // subimos la barrera
    // moviendo el servomotor a la posición de 90 grados.
    servoEntrada.write(90);
    // Mientras el coche sigue en la plataforma, esperamos.
    while(digitalRead(2)==HIGH){}
    // Esperamos 3 segundos a que haya terminado de pasar.
    delay(3000);
    // Volvemos a bajar la barrera, llevando el servomotor a 0 grados.
    servoEntrada.write(0);
    // Restamos un lugar al total de lugares disponibles.
    lugares = lugares - 1;
    // Escribir el valor de lugares en el display.
    escribir_numero(lugares);
    // Publicar en Adafruit la cantidad de lugares ocupados.
    olq.publish(IO_FEED, ( 8 - lugares) );
}

// Si hay un coche a la salida
// es decir, si el pulsador de salida está presionado
if(digitalRead(3)==HIGH){
    // subimos la barrera
    // moviendo el servomotor a la posición de 90 grados.
    servoSalida.write(90);
    // Mientras el coche sigue en la plataforma, esperamos.
    while(digitalRead(3)==HIGH){}
    // Esperamos 3 segundos a que haya terminado de pasar.
    delay(3000);
    // Volvemos a bajar la barrera, llevando el servomotor a 0 grados.
    servoSalida.write(0);
    // Incrementamos en 1 el valor total de lugares disponibles.
    lugares = lugares + 1;
    // Escribir el valor de lugares en el display.
    escribir_numero(lugares);
    // Publicar en Adafruit la cantidad de lugares ocupados.
    olq.publish(IO_FEED, ( 8 - lugares) );
}

// Llamamos a que la librería actualice lo que necesite.
olq.update();
}

```


Es importante destacar que el código que subimos para el primer estacionamiento es ligeramente diferente al que subimos para el segundo estacionamiento. Para el segundo cambiamos la línea:

```
#define IO_FEED      "estacionamiento1"
```

...por:

```
#define IO_FEED      "estacionamiento2"
```

De otro modo ambos publicarían al mismo feed y en el panel se nos mezclaría la información de ambos estacionamientos.

Cierre

Una vez finalizado este proyecto, si se quiere continuar, es posible extenderlo. Estas son algunas opciones sugeridas:

- Instalar sensores ultrasónicos y LED en cada espacio del estacionamiento que detecten la presencia de un vehículo, para visualizar rápidamente si los espacios están disponibles.
- Utilizar sensores ultrasónicos para detectar la presencia de un vehículo en la entrada y en la salida del estacionamiento, reemplazando los pulsadores.
- Conectar una alarma (un *buzzer*) para que suene cada vez que un vehículo sale o entra del estacionamiento y advierta a los peatones que circulan por la calle.

El proceso de resolución de problemas como los que se han planteado aquí permite la movilización y la integración de distintos saberes en la búsqueda de soluciones posibles a una situación dada. Si bien la información aquí fue presentada a modo de instructivo, se espera que sean los estudiantes organizados en pequeños grupos quienes vayan encontrando las mejores formas para construir los dispositivos. Esto implica preparar los materiales para que cada grupo cuente con todo lo necesario para la construcción del proyecto. Además, al interior de cada grupo, los estudiantes deben distribuirse los roles y las tareas de acuerdo a las demandas que van teniendo en las actividades.

Es importante que los docentes acompañen las producciones de cada grupo monitoreando los avances de todos los estudiantes y presentando la información que se considere necesaria para continuar la tarea. Pero, al mismo tiempo, es necesario que habiliten espacios para que los alumnos realicen hipótesis, planteen interrogantes, indaguen, prueben y realicen ajustes de acuerdo a lo que ellos mismo van pensando sobre cómo llevar a cabo el proyecto.

En este sentido, registrar lo que se va haciendo, las preguntas de los alumnos, las pruebas, los errores, y cómo se fueron construyendo los dispositivos, permite reflexionar sobre la propia práctica, reforzar los aprendizajes construidos a lo largo de este proceso y poder volver a ese material disponible para próximos proyectos que se realicen.

Una vez terminado el proyecto, se sugiere reunir y organizar con el grupo el registro que se hizo del proceso realizado. Esta instancia de sistematización también permite movilizar capacidades vinculadas a la comunicación porque implica tomar decisiones respecto a cómo se quiere mostrar el proyecto a otros (otros grupos, otras escuelas, otros docentes, a la comunidad, etc.).

Te recomendamos pasar por el repositorio de saberes Digitales para obtener más materiales y otros ejemplos: www.enfoco.net.ar/sd

Glosario

Electrónica y arduino

Arduino: Placa electrónica que contiene un microcontrolador programable y sistema de comunicación (USB y serial) que permite al usuario cargarle diversos programas así como también comunicarse con la misma. Del lado de la computadora se utiliza un IDE de programación para generar el código, compilarlo y quemarlo en la placa. Existen múltiples IDE compatibles con las placas Arduino.

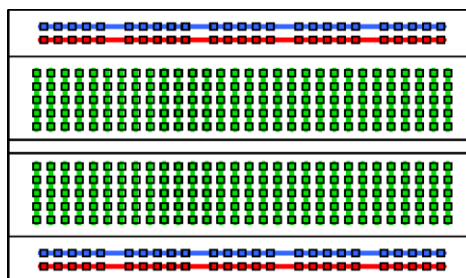
El microcontrolador posee entradas analógicas y digitales así como salidas digitales, PWM y servo. Las entradas y salidas digitales son las que permiten leer o escribir estados del tipo binarios. Pueden adoptar la forma de 0 ó 1, alto o bajo, verdadero o falso. Para prender y apagar los LED del semáforo utilizamos salidas digitales, las mismas están nombradas con números desde el 0 al 13.

Las entradas analógicas permiten leer información que puede adoptar diferentes niveles de tensión, tal como la lectura de un termómetro analógico, la posición de un potenciómetro, etc. Las mismas están identificadas en la placa como A0 a A5.

Puerto COM: Es el puerto de comunicaciones a través del cual un sistema operativo informático se comunica con un dispositivo externo tal como una placa Arduino. La asignación de los mismos suele realizarse de forma automática al conectar la placa vía USB. Dicha asignación suele ser dinámica, lo que significa que a veces cambia el número al conectar una misma placa en otro puerto USB o al conectar varias placas. En todos los IDE de programación es necesario especificar el puerto COM a través del cual nos comunicaremos con la placa Arduino.

Protoboard: Es una placa experimental que permite el prototipado rápido de circuitos electrónicos. Tiene orificios para insertar las patas de los componentes permitiendo que se conecten sin tener que recurrir a la soldadura.

El mismo posee una grilla de orificios que se encuentran conectados entre sí siguiendo el esquema de la imagen. Las líneas de conexión superior e inferior recorren la placa de punta a punta y suelen utilizarse para la alimentación del circuito, mientras que las líneas verdes se suelen utilizar para interconectar componentes. Tomar en cuenta que las líneas verdes se interrumpen en el centro de la placa. Generalmente se utilizan cables del tipo dupont para realizar conexiones en la protoboard



Servomotor: Un servo es un dispositivo que se compone de un motor y un sistema de control que le permite ubicarse en una posición específica. Los servos más comunes pueden moverse en un rango de 0° a 180°, sin poder girar de forma continua. Se suelen utilizar en aplicaciones tipo barreras o brazos mecánicos. La programación de los mismos es muy simple, teniendo que especificar únicamente el ángulo al que se lo quiere posicionar.

Existen también servos de “rotación continua” que permiten realizar un control relativamente preciso del movimiento así como también que el eje de giros continuos sin estar acotado a un rango de movimiento como el caso de los servos estándares. Este tipo de servo requiere una lógica de programación un poco más compleja que el caso anterior.

Los servos tienen 3 pines de conexión, dos de ellos se ocupan en alimentación eléctrica (VCC marrón y GND negro) y un tercer pin que se conecta a una salida digital del Arduino (cable naranja). Para controlar el servo el Arduino genera una señal con una frecuencia particular y un método de modulación de pulsos cuyo ciclo de trabajo equivale al ángulo que se desea posicionar el servo, no confundir este método de modulación con el PWM.

Pulsador: Es un interruptor eléctrico que se encuentra normalmente abierto y al presionarlo se cierra el circuito entre sus patas. Se suele usar como botón para que el usuario pueda interactuar con un dispositivo así como también como sensor para detectar cuando un móvil llegó a toparse con un obstáculo (sensor de choque o sensor de fin de carrera).

LED: Componente electrónico tipo diodo que emite luz. Es necesario tomar en cuenta la polaridad del mismo para ponerlo en funcionamiento. Conectándolo con la polaridad invertida generalmente no va a traer mayores consecuencias que la imposibilidad de hacer que encienda. Existen dos formas de distinguir la polaridad del mismo: podemos identificar la pata



negativa como la pata más corta u observando el lado plano en el encapsulado del mismo.

Resistencia: La resistencia eléctrica es una característica de todo material conductor eléctrico de hacer oposición al paso de la corriente eléctrica, es uno de los componentes más utilizados en la electrónica. El valor resistivo se mide en ohm y se representa con el símbolo Ω . Existen resistencias de valores que van desde menos de 1 ohm hasta varios millones. Se utilizan colores para codificar su valor. Se suelen utilizar para determinar la cantidad de corriente de una rama de circuito, por ejemplo para evitar que se queme el LED por exceso de corriente.

Display 7 Segmentos: El *display* de siete segmentos es un componente que se utiliza para la representación de números en muchos dispositivos electrónicos. Está compuesto de siete LED con forma de línea que se pueden encender o apagar individualmente.

Dado que este dispositivo no tiene ningún tipo de circuito de control incorporado, se controla directamente el encendido o apagado de cada segmento desde sus respectivas salidas

digitales de Arduino. Para poder mostrar un determinado número es necesario realizar una conversión desde la programación que se encargue de determinar cuáles segmentos representan a cada cifra.

Generalmente tienen 7 pines de conexión para los respectivos segmentos y a veces hay un pin adicional para prender un punto que se encuentra en el margen inferior. Hay un último pin que es el neutro común (GND).

Internet de las cosas

Panel de Control Adafruit: Los sistemas IoT trabajan apoyándose en un servidor que se encarga de centralizar y gestionar la información que reportan los diversos sensores así como responder a las consultas de los dispositivos que buscan acceder a dicha información (mostrarla en pantalla, tomar decisiones, etc). Adafruit es una plataforma online con posibilidad de uso gratuito que ofrece el servicio de gestión de esta información. La misma ofrece un alto grado de compatibilidad con diversos estándares de trabajo IoT y se encuentra principalmente orientada al uso educativo.

Feed: fuente de datos en la que uno puede publicar y a la que puede suscribirse. Es decir, permite enviar datos, para que estos sean almacenados en el tiempo así como también leerlos, recibiendo las actualizaciones de quienes estén publicando allí. Es una forma de almacenar información en una gran base de datos de forma ordenada, utilizando el concepto de etiquetas tanto al momento de escribirla como el de leerla.

Reconocimientos

Este trabajo es fruto del esfuerzo creativo de un enorme equipo de entusiastas y visionarios de la pedagogía de la innovación, la formación docente, la robótica, la programación, el diseño y la impresión 3D. Les agradecemos por el trabajo en equipo inspirador para traer a la realidad la obra que, en forma conjunta, realizamos INET y EDUCAR del Ministerio de Educación, Cultura, Ciencia y Tecnología de la Nación Argentina.

Contenidos

Equipo INET

Alejandro Anchava
Joreliz Andreyana Aguilera Barragán
Omar Leandro Bobrow
Alejandro Cesar Cáceres
Ezequiel Luberto
Gustavo Roberto Mesiti
Alejandro Palestrini
Judit Schneider
Pablo Trangone

Equipo Educar:

Pablo Aristide
Mayra Botta
Anabela Cathcarth
Eduardo Chiarella
María Laura Costilla
Diego Dorado
Facundo Dyszel
Federico Frydman
Matías Rinaldi
Uriel Rubilar
Camila Stecher
Carolina Sokolowicz
Nicolás Uccello

Para la confección de esta obra se contó con el apoyo de la Universidad Pedagógica Nacional "UNIPE". En particular en el desarrollo de los capítulos 1 y 2, los cuales estuvieron a cargo de los profesores Fernando Raúl Alfredo Bordinon y Alejandro Adrián Iglesias.

Producción y comunicación

Juliana Zugasti

Diseño y edición

Leonardo Frino
Mario Marrazzo

Corrección de estilo

María Cecilia Alegre

Agradecimientos especiales

Mariano Consalvo. Equipo ABP

Damián Olive. Equipo de ABP

María José Licio Rinaldi, Directora Nacional de Asuntos Federales INET, quien siempre acompañó a este equipo en todas las gestiones para su implementación

Estamos comprometidos en instalar la innovación en la escuela secundaria técnica: la robótica, la programación, el pensamiento computacional, los proyectos tecnológicos, el ABP, la impresión 3D, de manera más accesible para todos.

Agradecemos enormemente, docente, tu continua dedicación y compromiso con el futuro de tus estudiantes.

¡Estamos ansiosos por saber qué es lo que vamos a crear juntos!