UNIVERSIDAD DE QUINTANA ROO
DIVISIÓN DE CIENCIAS E INGENIERÍA

DESARROLLO DE UN SISTEMA PARA SEGURIDAD VEHICULAR

TESIS
PARA OBTENER EL GRADO DE
INGENIERO EN REDES

PRESENTA
ROGELIO DE JESÚS GARCÍA DZUL

DIRECTOR DE TESIS
DR. JAVIER VÁZQUEZ CASTILLO

ASESORES
DR. JAIME SILVERIO ORTEGÓN AGUILAR
DR. VÍCTOR MANUEL SÁNCHEZ HUERTA
MSI. RUBÉN ENRIQUE GONZÁLEZ ELIXAVIDE
MTI. MELISSA BLANQUETO ESTRADA

CHETUMAL QUINTANA ROO, MÉXICO, AGOSTO DE 2015

UNIVERSIDAD DE QUINTANA ROO
DIVISIÓN DE CIENCIAS E INGENIERÍA

TRABAJO DE TESIS ELABORADO BAJO SUPERVISIÓN DEL COMITÉ DE ASESORÍA Y APROBADO COMO REQUISITO PARCIAL PARA OBTENER EL GRADO DE:

INGENIERO EN REDES

COMITÉ DE TRABAJO DE TESIS

UNIVERSIDAD DE QUINTANA ROO
DCI DIVISIÓN DE CIENCIAS E INGENIERÍA

DIRECTOR: _____

DR. JAVIER VAZQUEZ CASTILLO

ASESOR: _____

DR. JAIME SILVERIO ORTEGÓN AGUILAR

ASESOR: _____

DR. VÍCTOR MANUEL SÁNCHEZ HUERTA

UNIVERSIDAD DE QUINTANA ROO
ÁREA DE TITULACIÓN

CHETUMAL QUINTANA ROO, MÉXICO, AGOSTO DE 2015

Agradecimientos

A mi familia, por su apoyo incondicional en todo momento y por darme las bases para alcanzar todas mis metas y poder llegar hasta donde estoy en estos momentos, y lo que esta por venir.

A Dios, quien me ha permitido llegar a este momento, ayudándome a seguir el camino correcto y dándome la fuerza y salud necesaria.

A mis profesores, quienes me han trasmitido su conocimiento y me han guiado durante este ciclo de mi vida, gracias a eso se ha formado un profesionista con grandes conocimientos y valores.

A mis amigos y compañeros de estudio, los cuales me han brindado su ayuda siempre que fue necesario; por todos esos días de estudio en equipo.

Al COQCYT quienes apoyaron este trabajo en el marco de la convocatoria “Apoyos económicos a estudiantes para asistentes de investigador”, despertando en mi el interés en la investigación y la búsqueda de estudiar un postgrado.

Este trabajo fue financiado en la convocatoria 2015 “Apoyo a la Titulación” de la división de Ciencias e Ingeniería con recursos PROFOCIE 2014.

A todas y cada una de las personas que estuvieron detrás de este logro, el cual es muy importante para mi, muchas gracias.

Dedicatoria

A mi madre…

María García quien siempre ha estado detrás de cada uno de los momentos importantes de mi vida, gracias por tus cuidados, atención, orientación, por darme la fuerza para siempre levantarme y seguir adelante y por ser más que una madre para mí, te amo.

Resumen

Ante la problemática del robo de autos, la cual ha ido aumentando exponencialmente, se desarolla este trabajo de tesis que consiste en la integración las tecnologías GSM y GPS en conjunto con el microcontrolador Arduino y los módulos de las tecnologías antes mencionadas, con el fin de crear un sistema de seguridad vehicular.

A lo largo de este trabajo, se abordan diferentes temas que se encuentran intimamente relacionados con el proyecto tales como la tegnologia GSM, GPS, y los microcontroladores, de estos temas se tienen sus principios básicos de funcionamiento asi como una breve historia de sus inicios y como fue creciendo a través de los años.

Se aborda el funcionamiento individual de cada uno de los componentes del sistema, ésto con ayuda de ejemplos individuales de cada uno de los módulos, asi como también la conexión de éstos con el microcontrolador Arduino.

Palabras clave: Arduino, GPS, GSM, seguridad vehicular.

Contenido

Agradecimientos 3
Dedicatoria 4
Resumen 5
Contenido 6
Figuras 8
Capítulo 1. Antecedentes 10
1.1 Antecedentes 10
1.2 Problemática 13
1.3 Justificación. 14
1.4 Objetivo General. 15
1.5 Objetivos Específicos. 15
Capítulo 2. Arquitectura del sistema propuesto 16
2.1 Microcontroladores 16
2.1.1 Arquitecturas Von Neumann y Harvard 19
2.2 GPS 21
2.3 GSM 25
2.3.1 Arquitectura GSM 27
2.4 Arduino 31
2.4.1 Ventajas 31
2.4.2 Modelos tarjetas arduino. 32
2.4.3 Software Arduino 40
Capítulo 3. Desarrollo del trabajo 42
3.1 Arduino 42
3.1.1 Variables 42
3.1.2 Operadores 43
3.1.3 Condicionales y ciclos 44

3.1.4 Ejemplos básicos 45
3.2 Arduino GSM 50
3.2.1 Recibir llamadas 53
3.2.2 Hacer llamadas 57
3.2.3 Recibir Mensajes 59
3.2.4 Enviar mensaje 61
3.3 GPS 64
3.4 Micro-SD 69
3.4.1 Listar archivos de la tarjeta micro-SD 72
3.4.2 Crear archivos en la tarjeta micro-SD 75
3.4 Sistema Final 78
Capítulo 4. Conclusiones 89
Bibliografia 90

Figuras

Figura 1.0. Tarjeta Geolink 10
Figura 1.1. Rastrador GPS para mascotas 11
Figura 1.2. Aplicación Positrace 12
Figura 1.3. Dispositivos Ubitrack 13
Figura 2.0. Esquema básico general de un microcontrolador 17
Figura 2.1. Arquitecturas Von Neumann (a) y Harvard (b) 20
Figura 2.2. Sistema GPS 22
Figura 2.3. Satélite GPS constelación NAVSTAR 23
Figura 2.4. Instalaciones del segmento de control GPS 24
Figura 2.5. Arquitectura funcional del sistema GSM 27
Figura 2.6 estructura interna del NSS 29
Figura 2.7. Modelos de Arduino 33
Figura 2.8. Arduino DUE 34
Figura 2.9. Arduino LilyPad 35
Figura 2.10. Arduino esplora 35
Figura 2.11. Arduino Leonardo 36
Figura 2.12. Arduino Zero 37
Figura 2.13. Arduino Yún 38
Figura 2.14. Arduino UNO 40
Figura 2.15. Entorno software Arduino 41
Figura 3.0. Ejemplos de Variables 43
Figura 3.1. Ejemplo Hola mundo 45
Figura 3.2. Resultado ejemplo Hola mundo 46
Figura 3.3. Conexión LEDs alternados 47
Figura 3.4. Código LEDs alternados 48
Figura 3.5. Conexión foto resistencia y led 49
Figura 3.6. Código foto resistencia y led 50
Figura 3.7. (a) Diagrama funcional SIM 900, (b) Pines de Salida SIM900 51
Figura 3.8. GPRS Shield V1.1 (b) 52
Figura 3.9. Puenteo de pines Tx y Rx 53
Figura 3.10. Código para recibir llamadas 1 54
Figura 3.11. Código para recibir llamadas 2 54
Figura 3.12. Código para recibir llamadas 3 55
Figura 3.13. Código para recibir llamadas 4 56
Figura 3.14. Recibiendo llamada en el arduino 57
Figura 3.15. Código realizar llamadas 1 58

Figura 3.16. Código para realizar llamadas 2 59
Figura 3.17. Código para recibir mensajes 1 60
Figura 3.18. Código para recibir mensajes 2 61
Figura 3.19. Recibiendo mensaje en el arduino 61
Figura 3.20. Código para enviar mensaje de texto 1. 62
Figura 3.21. Código para enviar mensaje de texto 2 63
Figura 3.22. Especificaciones Ublox NEO-6M-0-001 64
Figura 3.23. Módulo GY-GPS6MV2 65
Figura 3.24. Conexión Arduino-Módulo GPS 65
Figura 3.25. Código GPS 1 66
Figura 3.26. Código GPS 2 67
Figura 3.27. Recibiendo información del GPS 68
Figura 3.28. Coordenadas en Google Maps 69
Figura 3.29. Módulo Ethernet Arduino 70
Figura 3.30. Módulo Micro-SD Catalex 70
Figura 3.31. Pines SPI 71
Figura 3.32. Conexión Arduino a módulo micro-SD 72
Figura 3.33. Código enlistar archivos micro-SD 1 73
Figura 3.34. Código enlistar archivos micro-SD 2 74
Figura 3.35. Resultado enlistar archivos micro-SD 74
Figura 3.36. Código escritura en microSD 76
Figura 3.37. Resultados de escritura en microSD 1 77
Figura 3.38. Resultados de escritura en microSD 2 77
Figura 3.39. Conexión sistema para seguridad vehicular 79
Figura 3.40. Funcionamiento básico del sistema 79
Figura 3.41. Código sistema 1 80
Figura 3.42. Código sistema 2 81
Figura 3.43. Código sistema 3 82
Figura 3.44. Código sistema 4 83
Figura 3.45. Código sistema 5 84
Figura 3.46. Código sistema 6 85
Figura 3.47. Sistema para seguridad vehicular 86
Figura 3.48. Resultados prueba 1 87
Figura 3.49. Resultados 1, prueba 2 88
Figura 3.50. Resultados 2, prueba 2 88

Capítulo 1. Antecedentes

1.1 Antecedentes

En la actualidad existen distintos dispositivos y servicios de geo localización utilizados para la seguridad, rescate y/o monitoreo de vehículos, mascotas, personas, entre otros. Algunas de las compañías que ofrecen este tipo de servicios o venden dispositivos enfocados a la geo localización son las siguientes:

Geolink: Esta empresa enfocada a la geo localización de vehículos te vende una tarjeta, la cual podemos ver en la Figura 1.0, pero tenemos que pagar una renta mensual para poder tener soporte.

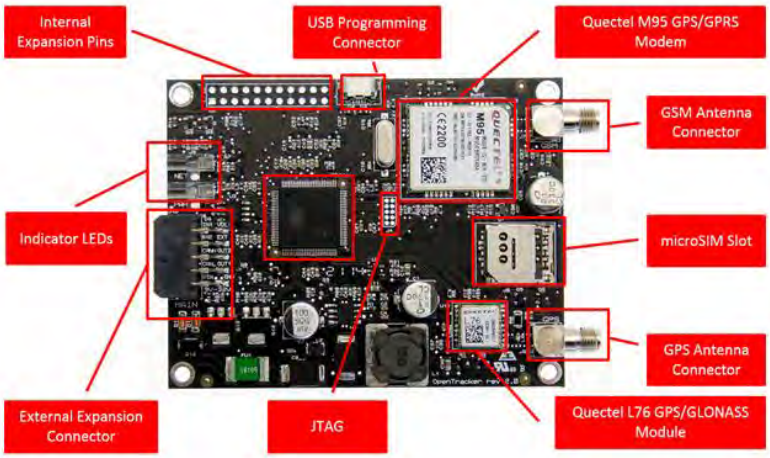


Figura 1.0. Tarjeta Geolink [1].

Whistle/Tagg: Empresa enfocada al cuidado de tu mascota, con un monitor de actividad y venden por separadoun dispositivo de localización GPS. Rastrean la ubicación mediante GPS de tu mascota y en caso de que se extravié puedes recibir alertas a un teléfono móvil, y monitor de actividad te dice si tu perro está sano y salvo. De igual manera, esta empresa vende un dispositivo el cual podemos observar en la Figura 1.1, y para utilizar el servicio se debe pagar una renta mensual fija.

Figura 1.1. Rastrador GPS para mascotas [2].

Positrace: Empresa enfocada al rastreo GPS y gestión de flotas, pensada para medianas y grandes empresas que tienen que administrar y monitorear flotas de vehículos, ofrece una aplicación con distintas funciones como medida de combustible, velocidad, ubicación entre otros. En la Figura 1.2 podemos ver una ventana de dicha aplicación. Al igual que las empresas mencionadas anteriormente se tiene que pagar una mensualidad y adquirir un equipo GPS.

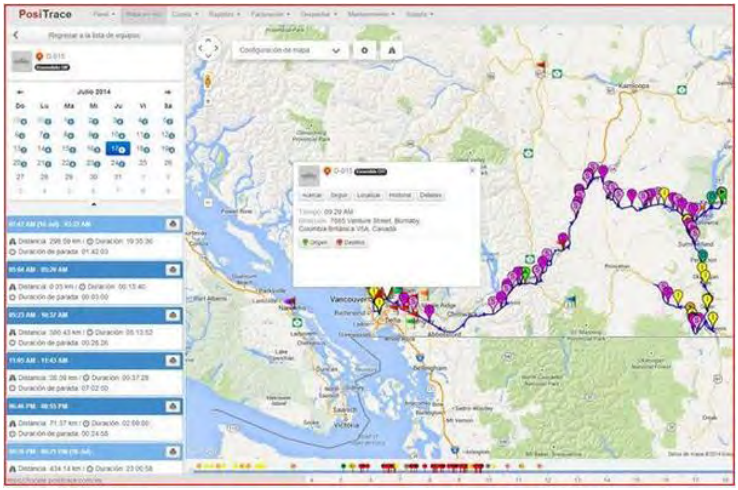


Figura 1.2. Aplicación Positrace [3].

Lojack: Esta empresa te ofrece un sistema diseñado para la localización, recuperación y entrega de vehículos robados, tiene distintos servicios como lojack, skyjack, skytrack y otros. En todos los servicios que ofrece se tiene que pagar una renta mensual y la compra e instalación de su dispositivo.

Ubitrack: Esta empresa provee información en línea para el monitoreo y control de bienes y personal, te proporciona datos como velocidad, rutas utilizadas, tiempos de paradas, salidas del área de trabajo. Esta información se puede visualizar en línea en su sistema o se envían alertas al email o mensajes al celular. Esta empresa de igual manera te cobra una renta mensual. En la Figura 1.3 podemos ver algunos de los dispositivos que manejan.

Esto por mencionar algunas, como se ha podido observar la mayoría o todas las empresas venden su dispositivo y de manera adicional éstas cobran una renta

mensual para proporcionar soporte y todos los beneficios que sus dispositivos ofrecen, los precios varían según el tipo de servicio el usuario requiera.

UBITRACK | UBITRACK POCKET | UBITRACK PORTABLE

Figura 1.3. Dispositivos Ubitrack [4].

1.2 Problemática

Hoy en día, los automóviles se ha hecho indispensables para la mayoría de la población, ya sea particular o que usemos el transporte público, siempre hacemos uso de estas máquinas que son de gran utilidad ya que nos ayudan a recorrer grandes distancias en un corto tiempo, los usamos para viajar y a diario para ir al trabajo, la escuela, el supermercado, entre otros. A medida que la población ha ido creciendo, la demanda vehicular ha ido en aumento; y de igual manera el robo de los mismos.

Los delitos en el país ha ido en aumento en los últimos años, los delitos que se dan con más frecuencia son el robo de personas, casas y autos. Este tipo de violencia afecta a todo tipo de personas, pero mayormente en estratos populares, ya que no cuentan con los recursos para protegerse contra la inseguridad que se ha ido viviendo en los últimos años; por lo que nos podemos percatar que la inseguridad afecta de manera desigual a los diferentes estratos sociales en México ya que en la actualidad existen distintos dispositivos y servicios tales

como chips de seguridad personal, seguros contra secuestros, seguros de vida, seguros para casas, alarmas en residencias, alarmas vehiculares, seguros vehiculares, sistemas de circuito cerrado, entre otros; los cuales sabemos que son muy costosos y la población de escasos recursos no tiene el acceso a estas formas de protección y se encuentran vulnerables antes el constante incremento de delincuencia en el país.

En México, el robo de autos es una problemática que ha venido creciendo exponencialmente en los últimos años, estamos hablando que de 2006 al 2013, según datos de la Asociación Mexicana de Instituciones de Seguros (AMIS) [5] el robo de autos se incrementó en un 53%; esta información nos da una idea de la importancia que debemos darle a este problema. En el país existen muchas aseguradoras que podrían proteger un auto contra robo; sin embargo, es necesario contar con una cobertura amplia que incluya este beneficio, lo que propiciará un incremento en el costo del seguro. Además, si desafortunadamente un vehículo es robado, la aseguradora destinará un tiempo, estipulado en el contrato, que posibilite que el vehículo sea recuperado antes de tener que remunerar económicamente al afectado por el robo. En lo general los dueños de las personas que acceden a la compra de su vehículo, en la mayoría de las ocasiones no cuentan con un seguro vehicular, y para estas personas es una pérdida total de su patrimonio en caso de robo.

1.3 Justificación.

Ante el rápido crecimiento en el porcentaje de robo de autos, surge este proyecto; el cual, con ayuda de las nuevas tecnologías que tenemos al alcance de nuestras manos, como microcontroladores y distintos módulos que nos ayudan a interconectar y relacionar y gestionar distintas tecnologías en un mismo dispositivo, nos ayudará a desarrollar un sistema para seguridad vehicular el cual

tendrá un costo bajo y una manera de operar bastante sencilla, orientado a todo tipo de usuarios.

Se le brindará al usuario la posibilidad de ubicar, de manera instantánea, la posición exacta de su vehículo de manera instantánea con tan solo un mensaje, lo cual es mucho más económico que las rentas mensuales que muchas compañías ofrecen; haciéndolo un sistema económico, rápido y eficiente.

1.4 Objetivo General.

Se diseñará e implementará un sistema para realizar la geo-localización de un vehículo en tiempo real; es decir, mediante este sistema se podrá determinar la ubicación de un determinado vehículo en el momento en que se desee mediante el uso de tecnologías existentes hoy en día. Se brindará al usuario la posibilidad de consultar mediante una página web y/o mediante mensajes de texto la ubicación exacta de su vehículo en tiempo real.

1.5 Objetivos Específicos.

- Diseño del sistema.
- Desarrollo del controlador de los dispositivos.
- Implementación e integración de los diferentes dispositivos.
- Pruebas del producto obtenido.

Capítulo 2. Arquitectura del sistema propuesto

2.1 Microcontroladores

La tecnología se ha ido convirtiendo en parte de nuestra vida cotidiana, algunas veces ni siquiera estamos conscientes de ello, pero siempre estamos en contacto con la tecnología, la usamos para tareas tan simples como realizar una llamada telefónica hasta para tareas de mayor dificultad como ensamblar automóviles, viajes al espacio, robots, entre otros. En la actualidad existe un sin fín de dispositivos electrónicos, entre ellos, los microcontroladores; estos llegaron para evolucionar la electrónica y facilitar muchas de las tareas que antes de su llegada se hacían complicadas. Desde que llegaron en 1971, cuando Intel lanzo los primeros microcontroladores funcionales que fueron el 4004 y el 8008, estos dispositivos se han hecho indispensables para distintas industrias y con el paso del tiempo se ha ido mejorando, surgiendo nuevos modelos con distintas capacidades y así como también distintos enfoques.

Los microcontroladores son circuitos integrados programables, encargados de realizar tareas específicas y están compuestos por el denominado CPU (Unidad Central de Procesamiento), la memoria, las entradas y las salidas. En la Figura 2.0 podemos observar un esquema donde se hace referencia a los elementos que previamente hemos mencionado; podemos ver que los componentes están unidos por buses y estos pueden ser de direcciones, de datos o de control, dependiendo del tipo de información que estén transportando dichos buses [6].

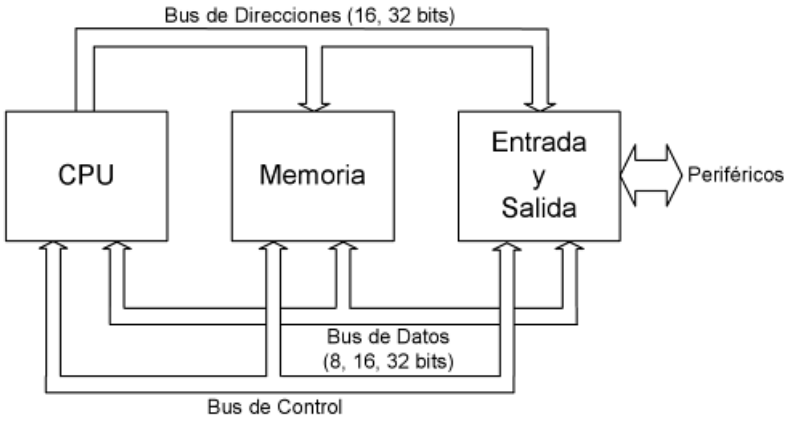


Figura 2.0. Esquema básico general de un microcontrolador [6].

El microcontrolador ejecuta el programa almacenado en memoria, el cual trabaja con algunos datos almacenados temporalmente e interactúa con el exterior a través de las entradas y salidas, las entradas dadas por los diferentes sensores que estemos usando y las salidas por los actuadores. Como ya hemos mencionado el microcontrolador se usa para diferentes aplicaciones, y en las grandes aplicaciones se pueden utilizar más de un microcontrolador, cada uno tendrá alguna tarea asignada. Cuando se requieren controladores avanzados o para tareas muy “pesadas”, se tienen muy en cuenta las características de este, algunas de las características que más se toman en cuenta son:

- *Recursos de entrada y salida*. Muchas veces utilizamos diferentes dispositivos de entrada y salida, ya sean analógicos o digitales, se debe tener en cuenta que el microcontrolador admita el tipo de dispositivos de entrada y salida que usaremos.
- *Espacio optimizado.* El tamaño del microcontrolador es un punto que se tiene en cuenta, al igual que la cantidad de terminales que este permita, todo este a un precio razonable.

- *Microcontrolador ideal según las necesidades.* Se debe tener en cuenta que el microcontrolador que se va a usar este enfocado a la aplicación que se quiere implementar; algunos fabricantes cuentan con una gama extensa de modelos que abarcan diversas necesidades, cambiando desde el diseño, memoria, procesador, tamaño, entre otras opciones.
- *Bajo consumo.* Muchas aplicaciones requieren una fuente de alimentación mediante pilas y esto hace el bajo consumo de energía de los microcontroladores sea una característica importante.
- *Protección de los programas frente a copias.* Cuando se almacena alguna aplicación o un conjunto de instrucciones en un microcontrolador es de gran importancia que este no pueda ser leído por alguna persona ajena al proyecto, eso para evitar copias no autorizadas de la aplicación.

La CPU se encarga de leer las instrucciones del programa almacenado en memoria, las interpreta y ejecuta; realiza operaciones aritméticas y lógicas elementales en su unidad aritmética lógica (ALU). La memoria, una parte importante del sistema del Microcontrolador, este tiene distintos tipos de memoria.

La memoria ROM es donde se almacena el programa que se va a ejecutar, también tenemos la memoria RAM en donde se almacena temporalmente todos los datos con los que el programa va a trabajar. En la actualidad muchos de los microcontroladores igual cuentan con una memoria no volátil EEPROM, la cual nos sirve para almacenar los datos fijos o que solo se cambian esporádicamente.

En los microcontroladores encontraremos que se contará siempre con más cantidad de memoria ROM que memoria RAM ya que la mayoría de los programas hacen procesos con pocos datos, y no requieren de mucha memoria RAM; la memoria RAM ocupa más espacio en el circuito que la memoria ROM y es mucho más costosa.

Para muchas de las aplicaciones se requiere que el microcontrolador interactúe con el exterior por eso, como ya habíamos mencionado, las entradas y salidas son de suma importancia. Las entradas y salidas están compuestos por los puertos paralelos y serie, temporizadores y gestión de interruptores. Como ya se había mencionado las entradas pueden ser digitales o analógicas, estas últimas nos están asociadas a convertidores A/D y D/A.

2.1.1 Arquitecturas Von Neumann y Harvard

Las instrucciones que están almacenadas en la memoria de los microcontroladores, deben pasar por la CPU para su descodificación y ejecución, por consiguiente algunos datos en memoria son leídos por la CPU y otros son escritos en memoria por este mismo. Por lo tanto puede intuirse que la organización de la memoria así como también la comunicación de esta con el CPU son dos aspectos que influyen en las prestaciones del microcontrolador.

Las arquitecturas de Von Neumann y Harvard son modelos generales del hardware de los ordenadores, los cuales representan distintas soluciones al problema de interconexión de la CPU con la memoria y la organización de la memoria como almacén de instrucciones y datos.

En la Figura 2.1 podemos observar los dos modelos que mencionamos en el párrafo anterior. Podemos ver que la arquitectura de Von Newman únicamente utiliza una memoria para almacenar el programa y los datos, utilizan el mismo bus de direcciones para direccionar instrucciones y datos. De igual manera la misma señal de control que emite la CPU sirve para leer datos y leer instrucciones. Como vimos anteriormente la memoria ROM se usa para almacenar el programa y la memoria RAM para almacenar datos, pero para la CPU no hay distinción entre los dos tipos de memoria, sino que ambas memorias

forman una memoria de lectura y escritura para la cual la CPU emite señales de control, de direcciones y datos.

Podemos observar en la misma Figura, que la arquitectura Harvard utiliza memorias separadas, una para instrucciones y otra para datos. La memoria de instrucciones tiene su bus de direcciones, su bus de instrucciones y un bus de control. De igual manera la memoria de datos tiene sus propios buses de direcciones, datos y control que son totalmente independientes de los buses de la memoria de instrucciones.

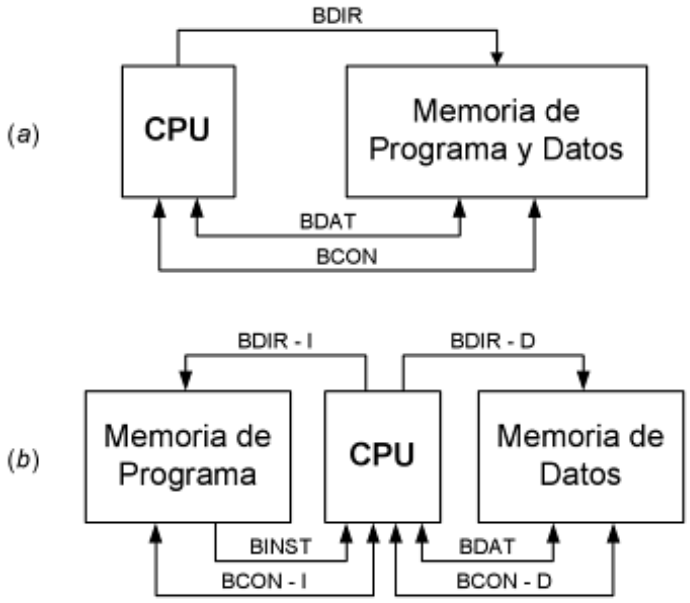


Figura 2.1. Arquitecturas Von Neumann (a) y Harvard (b) [6].

En la arquitectura de Von Neumann se requieren menos líneas para conectar la CPU con la memoria, por lo que tenemos una conexión más simple, pero una desventaja que tenemos ante la arquitectura de Harvard es que no podemos manipular simultáneamente datos e instrucciones porque comparten los buses, y en la arquitectura Harvard esto si es posible ya que cuenta con buses separados, esto le da una mayor velocidad a la hora de ejecutar los programas.

2.2 GPS

El Sistema de Posicionamiento Global (GPS) es un sistema de satélites que te permite determinar una determinada posición durante las 24 horas del día, cualquier día del año y en cualquier parte del mundo, ésto a través de la constante transmisión que estos hacen hacia la tierra de información codificada que puede ser leída por diversos receptores GPS.

Desde tiempos remotos se han usado diversas técnicas para saber la posición en la que estamos y hacia donde debemos ir, tales como la posición de las estrellas en el cielo, el uso de la brújula, entre otras herramientas de navegación tanto terrestre como marítima.

Bajo la supervisión de la fuerza aérea de Estados Unidos, en el año de 1960 dio inicio el proyecto donde nació el GPS como lo conocemos hoy en día, a partir de 1974 otros cuerpos militares se unieron a este proyecto y lo renombraron como “Navigation Satellite Timing and Ranging GPS” (NAVSTAR GPS), y desarollar este sistema tuvo un costo de 10 billones de dólares y fue hasta el 1995 cuando se declaró plenamente operacional.

Para poder recibir las señales emitidas por los satélites es necesario tener un receptor GPS, el cual puede decodificar dichas señales y darnos una ubicación exacta en coordenadas (latitud, longitud); además de esta información, se puede tener información adicional tal como hora, fecha, altitud, velocidad, entre otros.

La precisión del GPS tiene un margen de error de aproximadamente 15 metros, error no superado por ningún otro dispositivo. La gran demanda de este tipo de dispositivos ha llevado a que fabriquen receptores de diferentes tipos, orientados a tareas específicas, como navegación terrestre, navegación marítima e incluso en dispositivos como teléfonos inteligentes y módulos especializados para poder integrar el GPS a microcontroladores y otras plataformas.

Como podemos apreciar en la Figura 2.2 el sistema GPS está conformado por tres segmentos:

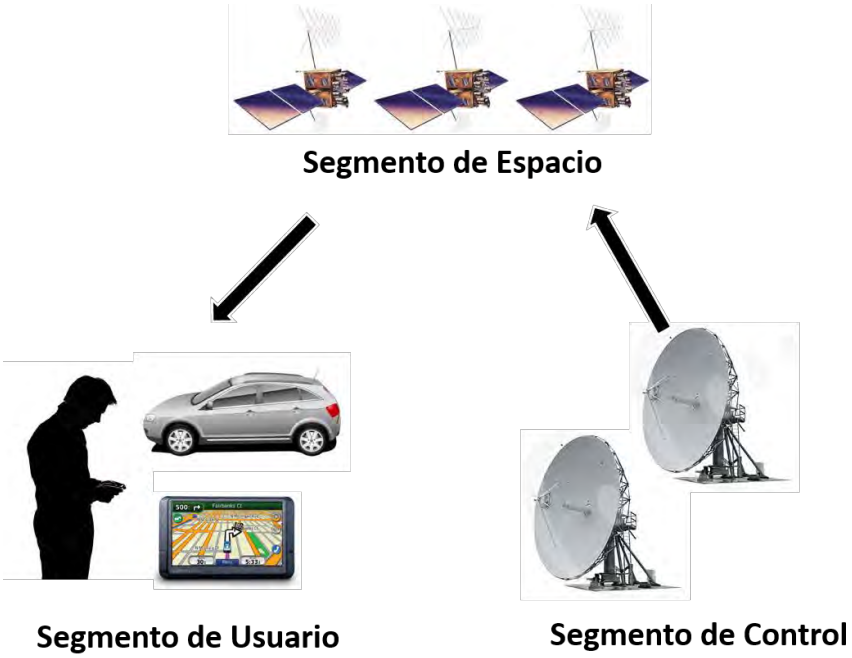


Figura 2.2. Sistema GPS.

Segmento de espacio. Conformado por 24 satélites ubicados en órbita alta a aproximadamente 20,000 kilómetros sobre la superficie terrestre; a esta altura pueden cubrir una mayor área de la tierra a manera que al menos 4 satélites estén disponibles para el receptor GPS.

Impulsados por una serie de cohetes y alimentados por energía solar y bancos de baterías usados como energía de respaldo, dichos satélites se encuentran orbitando alrededor de la tierra a 11,265 kilómetros por hora, lo que los hace dar una vuelta completa al globo terrestre cada 12 horas. En la Figura 2.3 podemos ver un ejemplo de un satélite GPS.

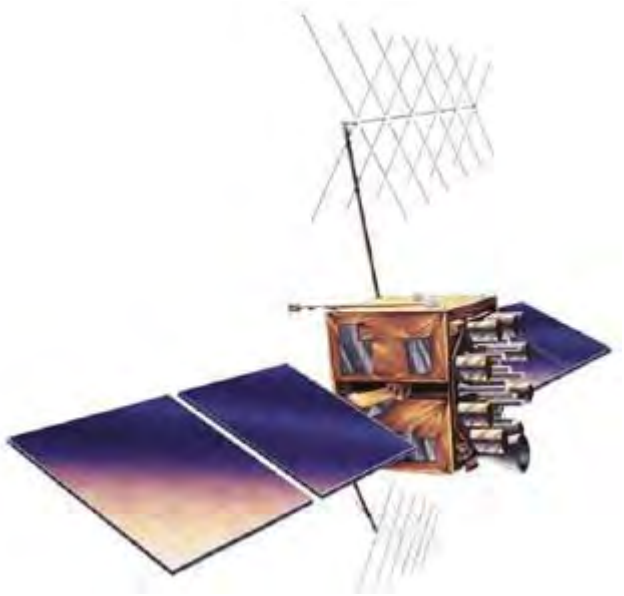


Figura 2.3. Satélite GPS constelación NAVSTAR [7].

Segmento de control. Está conformado por una red de instalaciones que se encuentran en diferentes partes del mundo tal y como podemos observar en la Figura 2.4.

Con una estación de control maestro, una estación de control maestro suplente, 12 estaciones de mando y control de antenas, y 16 sitios de monitoreo, los satélites son monitoreados, analizados y están recibiendo comandos y datos constantemente, todo esto para verificar su sincronización de reloj, posición en órbita y estar al tanto de cualquier desperfecto en cada uno de los satélites.

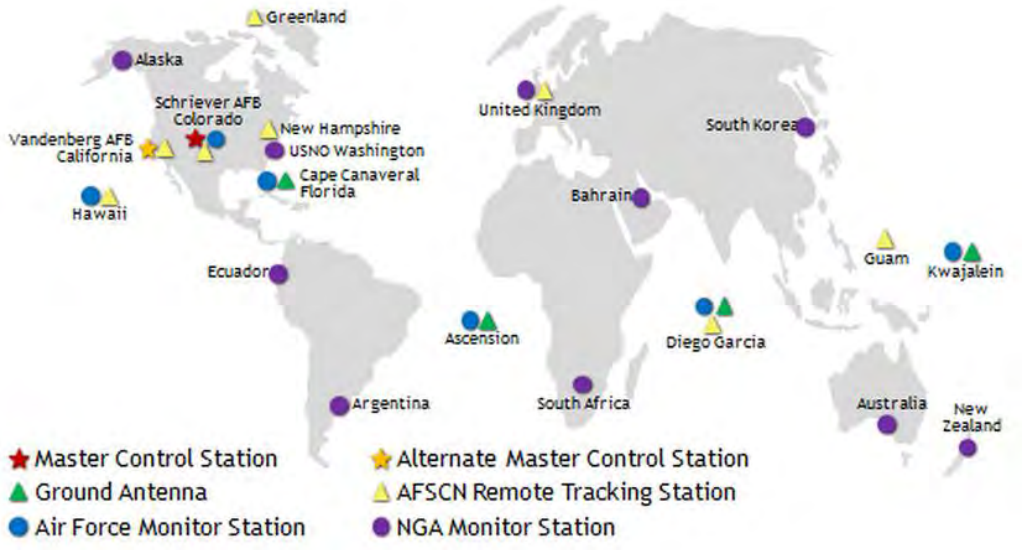


Figura 2.4. Instalaciones del segmento de control GPS [8].

Segmento de usuario. En los inicios de esta tecnología, este segmento estaba orientado únicamente para cuestiones militares, pero al paso del tiempo se fue liberando para todo tipo de usuarios y las empresas lo han aprovechado al máximo, incluyendo esta tecnología en gran parte de sus dispositivos, como celulares, tabletas electrónicas, relojes, vehículos, entre muchos otros dispositivos, y esto ha sido de gran ayuda para todos los usuarios y la ha hecho una tecnología muy común en nuestros días, cuando en un inicio se tenía que tener un receptor GPS especializado, pero ahora incluso hay módulos GPS para que se pueden integrar a microcontroladores y otros dispositivos de desarrollo.

Los receptores solo reciben la transmisión del satélite pero no interactúan con él en ningún momento, por lo que la cantidad de usuarios GPS que pueden estar haciendo uso de esta tecnología simultáneamente es ilimitada.

Todo suena como una tecnología magnifica, pero tiene sus limitaciones, las radioseñales que son emitidas por los satélites no pueden penetrar edificios muy grandes, vegetación densa, rocas o malas condiciones geográficas; aunque eso igual depende de la sensibilidad de la antena del receptor GPS.

Con respecto al código A y P; los satélites envían ondas de radio que transportan secuencias de números llamados códigos, estos envían dos secuencias de números, de precisión (P) y adquisición común (CA). Cada satélite tiene un código P y CA y así el receptor puede dar la diferencia entre señales enviadas por cada satélite. El código P proporciona un grado mayor de precisión, debido a su valor de modulación; orientado para los receptores militares ya que los receptores comunes son incapaces de utilizarlo. Los satélites utilizan dos secuencias para emitir las señales: 1227,6 MHz (L2) y 1575,42 MHz (L1), el código P lo podemos encontrar en ambas frecuencias mientras que el código CA solo en la frecuencia de L1.

2.3 GSM

Hablar de la tecnología GSM (Global System for Mobile Communications) nos lleva años atrás, ya que en el año 1982, a partir de la Conferencia Europea de Correos y Administración de Telecomunicaciones (CEPT, por sus siglas en Francés), surgió GSM como un estándar para las tecnologías móviles europeas. A través de los años GSM se convirtió en el estándar de comunicación móvil más usado a nivel mundial gracias a su facilidad de integración y actualización a mejoras que hacen posible su constante escalabilidad hacia nuevos servicios.

En 1946, la compañía AT&T (American Telephone and Telegraph) introdujo el primer servicio de telefonía móvil en los Estados Unidos; este sistema se basó en la transmisión de frecuencia modulada.

En los años 60, Bell System dio a conocer el Servicio Telefónico Móvil Mejorado (IMTS, Improved Mobile Telephone System), con mejoras en el diseño del transmisor y receptor que permitieron reducir el ancho de banda del canal de FM a 25-30 KHz.

Por otra part, hubo dos tecnologías que dieron un gran realce a la tecnología GSM las cuales fueron la invención del microprocesador y el uso de un enlace de control digital entre el teléfono móvil y la estación base.

En el año de 1989 surge GSM que primero se conocido como Grupo Especial Móvil y ya un tiempo después como lo conocemos actualmente, Sistema Global para comunicaciones móviles, que tuvo la finalidad de unificar los sistemas europeos.

GPRS (General Packet Radio Service) es una mejora de la tecnología GSM, donde se modifica la forma de transmitir los datos, donde se hace a un lado la conmutación de circuitos usada en GSM y entra la conmutación de paquetes. Esto proporciona velocidades de transferencia mucho mayores y esta tecnología puede ser usada en las redes GSM.

Gracias a la conmutación por paquetes que usa esta tecnología, los usuarios GPRS solo harán uso de la red cuando envíen o reciban paquetes de información; cuando se encuentren inactivos, el canal que se le fue asignado podrá ser usado por otros usuarios, haciendo así un uso más eficiente de la red y permitiendo a las operadoras dotar de más de un canal a los usuarios, sin el temor de saturar la red, y mientras más canales tenga un usuario para recibir paquetes, la velocidad será mejor. Con GPRS es posible tener terminales que gestionen cuatro canales de bajada simultáneamente y dos de subida, lo cual podría ofrecer una velocidad mucho mayor a los 9.6 Kbps que ofrecía GSM, y podía llegar a los 40 Kbps de bajada y 20 Kbps de subida y esto te permitía recibir simultáneamente voz y datos.

Con el paso de los años, surgieron muchas mejoras en diferentes tecnologías como el caso de HSCSD (High Speed Circuit Switched Data), cada una ofreciendo mejores servicio y mayor velocidad tal es el caso de la tecnología EDGE (Enhaced Date Rates for GSM Evolution) que es la mejora de GPRS y ofrecía 384 kbps superando los 115 kbps de su antecesor.

Una de las grandes ventajas del estándar GSM es su capacidad de cambiar de proveedor sin necesidad de cambiar de aparatos celulares, al igual de tener una gran variedad de proveedores de equipos GSM.

2.3.1 Arquitectura GSM

En la Figura 2.5 podemos observar que la arquitectura funcional del sistema GSM se divide en cuatro.

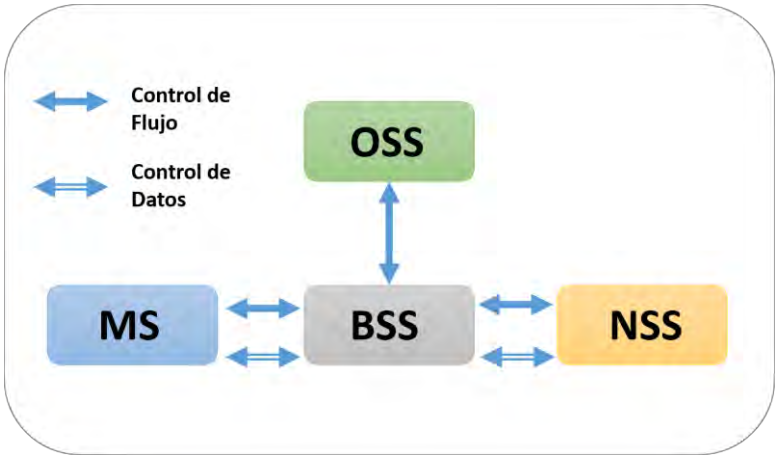


Figura 2.5. Arquitectura funcional del sistema GSM.

Estación Móvil (MS, Mobile Station). Es el equipo al que se le va a brindar el servicio, esto se realiza mediante la tarjeta SIM (Subscriber Identity Module), el SIM es el que permite que el usuario se beneficie de los servicios de telefonía móvil de manera independiente, para usar la tarjeta SIM se necesita insertar en un equipo o terminal que trabaje mediante la tecnología GSM, que puede ser algún teléfono móvil, Tablet, o algún módulo GSM.

Cada estación móvil está identificada por el IMEI (International Equipment Identity), el cual es único, y las tarjetas SIM también están identificadas pero mediante el IMSI (International Mobile Subscriber Identy).

Sistema de Estación Base (BSS). Tiene dos componentes, la Estación Base Transceptora (BTS) la cual se localiza en la antena, y la controladora de estación

base (BSC); estos se comunican a través de una interfaz conocida como A-bis lo cual permite la interconexión entre dispositivos de diferentes fabricantes.

El BTS maneja los protocolos de radio de la interfaz con la estación móvil, los requisitos más importantes con los que debe cumplir una BTS son portabilidad, confiabilidad y costo mínimo. El BSC maneja los recursos de radio para más de una estación base y es la conexión entre la estación base y el MSC.

Entre las funciones principales del sistema de estación base podemos encontrar:

- Gestión de la movilidad
- Tratamiento y transcodificación de la voz
- Control de la red de radio
- Establecimiento de la conexión entre la MS y el NSS
- Recopilación de material estadístico

Sistema de Red (NSS). El Network and Swiching System (NSS) tiene un componente principal que es el Mobile Services Switching Center (MSC). El NSS actúa como un nodo de conmutación PSTN (Public Swiched telephone network) pero además proporciona las funciones necesarias que permiten manipular a un usuario móvil como el registro, autenticación, actualización de la localización, entre otros. El MSC proporciona la conexión a la red fija (PSTN).

En la Figura 2.6 podemos observar los componentes internos que conforman un NSS.

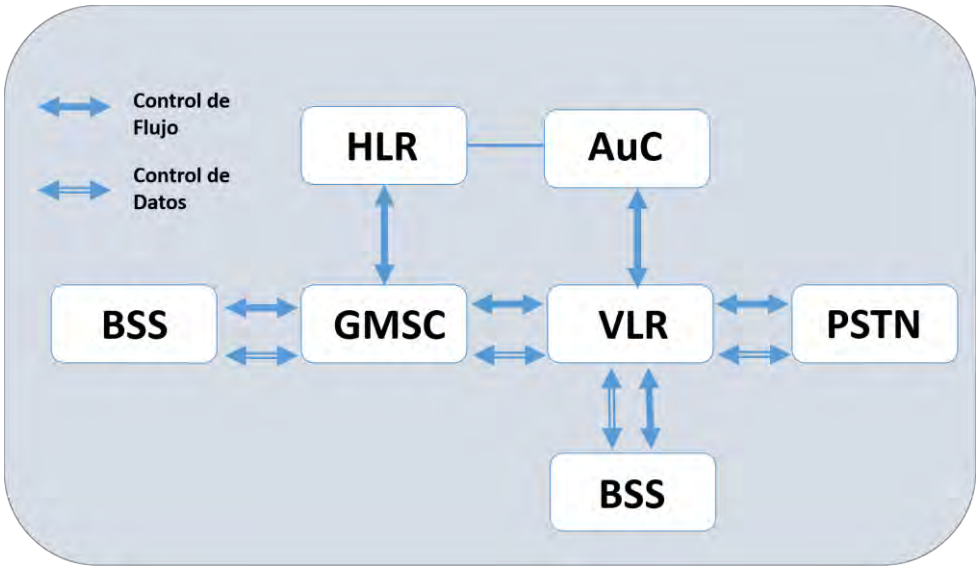


Figura 2.6 estructura interna del NSS.

El HLR (Registro de Posición Base) contiene toda la información de los suscriptores registrados en cierta red GSM, parte de la información almacenada es la localización general del suscriptor, que es usada a la hora de realizar el encapsulamiento de llamadas entrantes. Entre las funciones principales del HLR están la gestión de servicios, datos de usuario, estadísticas y la gestión de la movilidad.

El VLR (Registro de Posición Visitante) cuenta con información administrativa proveniente del HLR, dicha información es necesaria para el control de las llamadas y para proveer de los servicios subscritos.

Es recomendable que el VLR este muy cercano al MSC, ya que la proximidad de estos, acelera el acceso a la información que el MSC necesita durante el proceso de llamada.

El AuC (Centro de Autenticación) es una subdivisión del HLR, se encarga de los aspectos de seguridad, con una base de datos protegida que tiene una copia de la clave secreta que se encuentra guardada en la tarjeta SIM de cada suscriptor, la cual se es usada para su autenticación.

El GMSC (Centro de Conmutación Móvil) es el dispositivo necesario para interconectar la red GSM con las redes externas ya que permite interrogar al HLR para obtener información del encaminamiento para una llamada dirigida a un móvil.

Entre las funciones principales del NSS se encuentran:

- Control de llamada
- Tarificación
- Interconexión de redes
- Datos del suscriptor y gestión de los servicios
- Gestión de la movilidad
- Gestión de la seguridad
- Control del BSS

Sistema de Operación y Mantenimiento (OSS). Conformado por el OMC y el NMC, el primero mencionado es un centro de trabajo donde se encuentran funciones de operación y mantenimiento tales como:

- Gestión indirecta de los datos relativos a cada terminal móvil
- Registro de los datos de transmisión, de tráfico y de alarmas
- Modificación de los parámetros de servicios en la VLR, HLR y BSC
- Configuración de los aparatos de la red
- Registro de los datos de cobro

2.4 Arduino

La placa Arduino es una plataforma microcontroladora al cual se le pueden conectar diversos sensores y actuadores para que se pueda comunicar con el mundo exterior; existen diversos fabricantes que se han enfocado en crear hardware para esta plataforma gracias a la demanda que este ha tenido. Una de las grandes ventajas de esta plataforma es que es de código abierto, y podemos crear grandes proyectos en conjunto con la variedad de hardware disponible, como ya mencionamos anteriormente. La placa Arduino puede trabajar de manera autónoma o mediante el software que se puede descargar de manera gratuita en la página www.arduino.cc. El lenguaje de programación que usa esta plataforma es una implementación de Wiring.

2.4.1 Ventajas

Existen muchos microcontroladores y plataformas con microcontroladores similares a la plataforma arduino, pero mencionaremos algunas de las ventajas de Arduino sobre los demás microcontroladores:

- Económico: Las placas Arduino en comparación con otras plataformas es más económico, además que existe una gran variedad, para diferentes tipos de proyectos y los precios varían desde los 400 hasta los 1800 pesos mexicanos.
- Multiplataforma: La mayoría de los microcontroladores se enfocan para ser usados en Windows, a diferencia de ello, el software de esta plataforma es compatible con Windows, Mac y Linux.
- Programación simple: Arduino está enfocado a todo tipo de usuarios, tanto para usuarios con conocimientos avanzados de programación, así como también usuarios que son nuevos en este ambiente. De igual manera Arduino tiene suporte en su página principal al igual que en otros foros, donde usuarios experimentaros te podrían ayudar en cualquier problema duda que tengas.

- Código abierto: El código Arduino está abierto para que cualquier programador experimentado pueda ampliar dicho lenguaje, por ejemplo crando nuevas librerias. De igual manera los planos de la plataforma están abiertos a todo público para que fabricantes puedan hacer sus propias versiones mejorando las placas oficiales e incluso para que cualquier persona con conocimientos básicos de electrónica pueda crear su propia placa de desarrollo.

2.4.2 Modelos tarjetas arduino.

Existe una gran variedad de modelos de arduino, cada una para proyectos o tareas específicas, más pines de E/S, velocidad de procesamiento aumentada, el diseño y entre otras diferencias, en este proyecto usaremos la versión arduino UNO, pero a continuación mencionaremos algunas de las versiones que existen actualmente de esta plataforma.

- Arduino Uno
- Arduino Leonardo
- Arduino Due
- Arduino Yún
- Arduino Tre
- Arduino Zero
- Arduino Micro
- Arduino Mega ADK
- Arduino Robot
- LilyPad Arduino
- Arduino Nano
- Entre otros

En la Figura 2.7 podemos observar algunas de las versiones mencionadas anteriormente, podemos notar que existen de diferentes tamaños y diseños, al igual que sus especificaciones técnicas, mencionaremos brevemente

algunos de los modelos de Arduino, adentrándonos más en el modelo UNO, ya que es el que estaremos usando para este proyecto.

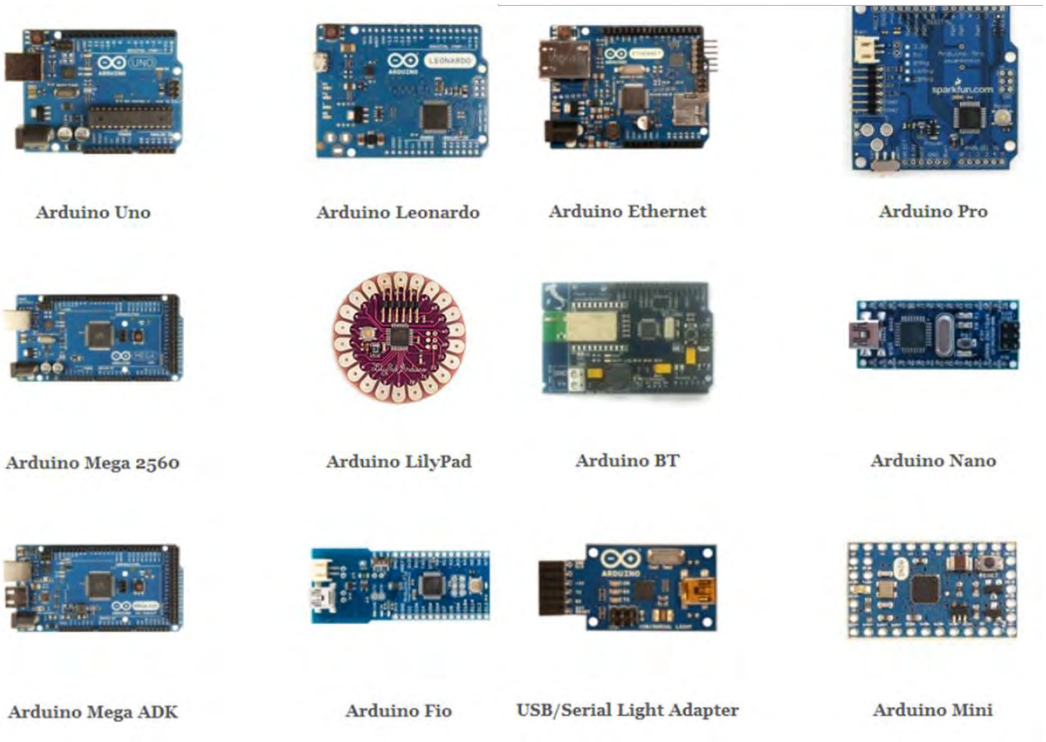


Figura 2.7. Modelos de Arduino.

2.4.2.1 Arduino DUE

Esta placa cuanta con un microcontrolador Ateml SAM3X8E ARM Cortex-M3 de 32 bits que trabaja a 84 MHz, con lo cual tendremos una potencia de cálculo. Ideal para proyectos que requieran de una alta capacidad de procesamiento. Puede realizar operaciones con datos de 4 bytes en tan solo un ciclo de reloj gracias a su procesador de 32 bits. Cuenta con una SRAM de 96kb y viene equipado con DMA para acceso directo a memoria e intensificar el acceso a memoria de la CPU. Tiene 512 Kb de memoria flash y dispone de 54 pines de E/S digitales, 12 de estos pueden ser usados como PWM, tiene 12 pines analógicos y 4 UARTs. Tiene capacidad de conexión USB OTG, dos conexiones DAC, 2 TWI, SPI y JTAG, por lo que nos damos cuenta que es un arduino muy

completo para prácticamente cualquier proyecto. Podemos ver el diseño de esta tarjeta en la Figura 2.8.

Figura 2.8. Arduino DUE [12].

2.4.2.2 Arduino LilyPad

Esta placa puede ser puede ser integrada en prendas y textiles, desarrollada por Leah Buechley y SparkFun electronics, con características limitadas pero con diferentes capacidades de integración y con una base flexible. Cuenta con dos versiones de microcontroladores diferentes Atmega168V y Atmega328V, ambas trabajando a 8 MHz, a diferencia que la segunda es más potente y trabaja a 5.5 V y la primera a 2.7 V. Esta placa dispone de 14 pines digitales, 6 analógicos; viene equipada con 16 Kb de memoria flash, 1 Kb de SRAM y 512 bytes de EEPROM. En la Figura 2.9 podemos ver su particular diseño circular.

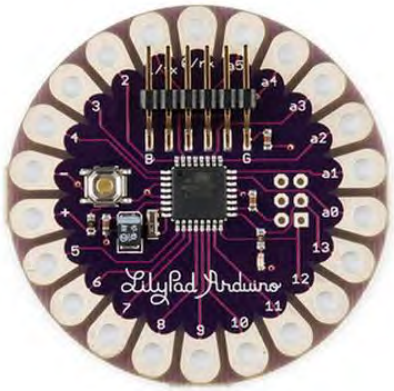


Figura 2.9. Arduino LilyPad [12].

2.4.2.3 Arduino Esplora

Con un particular diseño el cual podemos observar en la Figura 2.11, y un reducido tamaño que viene equipado con diferentes sensores ya integrados como acelerómetro, sensor de temperatura y luz, zumbador, botones, micrófono y un joystick. Esta placa esta orientada para las personas que están iniciando en el mundo de la electrónica y puede hacer proyectos pequeños con los sensores ya incorporados ya que no se tienen mucha capacidad de conectividad, más que una pantalla TFT LCD. Con un microcontrolador Atmega32u4 trabajando a 16 MHz y a 5 V, con 2.5 Kb de SRAM, 1 Kb de EEPROM y 32 Kb de memoria flash.

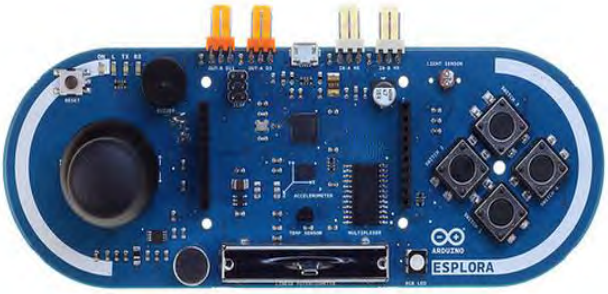


Figura 2.10. Arduino esplora [12].

2.4.2.4 Arduino Leonardo

Esta placa viene equipada con un microcontrolador Atmega32u4 de bajo consumo trabajando a 16 MHz. Con 32 Kb de memoria flash, 2.5 Kb de SRAM y 1 Kb de EEPROM. Cuenta con 20 pines digitales (7 PWM) y 12 pines analógicos. Con una conexión micro-USB ayuda a reducir su tamaño. En la Figura 2.11 podemos observar el diseño de este modelo.

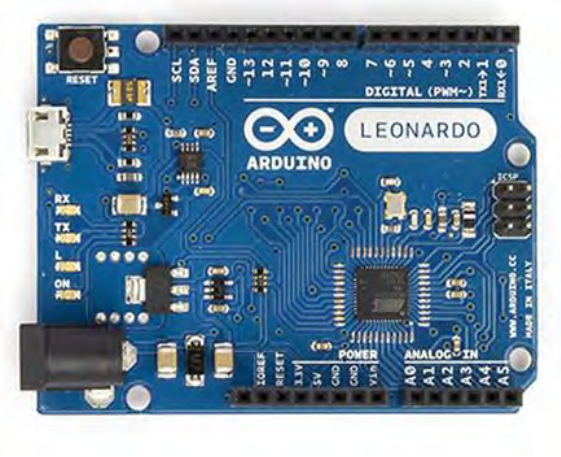


Figura 2.11. Arduino Leonardo [12].

2.4.2.5 Arduino Zero

Una placa que viene con un microcontrolador Atmega basado en arquitectura Atmel SAMD21 MCU de 48 MHz con core ARM Cortex M0 de 32 bits además de 256 Kb de memoria flash, 32 Kb de SRAM y 16 Kb de EEPROM. Cuenta con 14 pines digitales (12 PWM y UART) y 6 pines analógicos. Orientado para usuarios que requieran una gran potencia de procesamiento, en la Figura 2.12 podemos ver el diseño de este modelo, y observamos que es similar al arduino UNO, pero usa un puerto micro-USB.

Figura 2.12. Arduino Zero [12].

2.4.2.6 Arduino YÚN

Esta placa viene con un microcontrolador Atmega32u4 y un chip Atheros AR9331, el cual controla el host USB, el puerto para micro-SD y la red Ethernet y WiFi. El procesador Atheros soporta la distribución OpenWrt-Yun de Linux. Cuenta con 20 pines digitales de los cuales 7 pueden ser usados en modo PWM y 12 como analógicos. Esta plata tiene 32 Kb de memoria flash, 2.5 Kb de SRAM, 1 Kb de EEPROM y por su parte el chip AR9331 trabaja a 400 MHz y cuenta con RAM DDR2 de 64 MB y 16 MB de memoria flash para un sistema Linux embebido. En la Figura 2.13 podemos observar el diseño de este modelo de Arduino.

Figura 2.13. Arduino Yún [12].

2.4.2.7 Arduino UNO

Abordaremos un poco más a fondo el modelo Uno de arduino, del cual podemos observer su diseño en la Figura 2.14, ya que es con la que estaremos trabajando en este proyecto. Arduino Uno es una plataforma que contiene un microcontrolador ATmega328, con regulador de tensión, un puerto USB-Serie para poder programar dicho microcontrolador desde cualquier PC de manera sencilla y hacer pruebas con el monitor serie del programa de arduino.

Este modelo del arduino cuenta con 14 pines, los cuales pueden configurarse como entradas o salidas según sean nuestras necesidades y conectar distintos sensores o actuadores, 6 de los 14 pines son para entradas o salidas analógicas con lo cual nos permite trabajar con señales PWM, al igual que pines de alimentación de 5 y 3.3 V.

Los pines especiales que maneja el Arduino Uno son los siguientes:

- **RX y TX:** Se usan para transmisiones serie de señales TTL.
- **Interrupciones externas:** Los pines 2 y 3 están configurados para generar una interrupción en el atmega. Las interrupciones pueden

dispararse cuando se encuentra un valor bajo en estas entradas y con flancos de subida o bajada de la entrada.

- **PWM:** Arduino dispone de 6 salidas destinadas a la generación de señales PWM de hasta 8 bits.
- **SPI:** Los pines 10, 11, 12 y 13 pueden utilizarse para llevar a cabo comunicaciones SPI, que permiten trasladar información full dúplex en un entorno Maestro/Esclavo.
- **I 2C:** Permite establecer comunicaciones a través de un bus I 2C. El bus I 2C es un producto de Phillips para interconexión de sistemas embebidos. Actualmente se puede encontrar una gran diversidad de dispositivos que utilizan esta interfaz, desde pantallas LCD, memorias EEPROM, sensores, entre otros.

En la Tabla 1.0 podemos ver de manera general las especificaciones del Arduino Uno, como voltaje de entrada, cantidad de memoria, entre otros.

Microcontrolador	Atmega328
Voltaje de operación	5V
Voltaje de entrada (Recomendado)	7 – 12V
Voltaje de entrada (Límite)	6 – 20V
Pines para entrada- salida digital.	14 (6 pueden usarse como salida de PWM)
Pines de entrada analógica.	6
Corriente continua por pin IO	40 mA
Corriente continua en el pin 3.3V	50 mA
Memoria Flash	32 KB (0,5 KB ocupados por el bootloader)
SRAM	2 KB
EEPROM	1 KB
Frecuencia de reloj	16 MHz

Tabla 1.0. Características Técnicas Arduino Uno [12].

Figura 2.14. Arduino UNO [12].

2.4.3 Software Arduino

Ahora veremos el software que nos ayudará a programar y hacer diversas pruebas en nuestro Arduino, veremos cómo funciona y como programar nuestro Arduino, haciendo nuestras primeras pruebas.

La plataforma arduino tiene su propio lenguaje de programación, el cual esta basado en C y soporta todas sus funciones, al igual que algunas de las funciones de C++.

Lo primero que debemos hacer es descargar el software de Arduino de la página http://www.arduino.cc/en/Main/Software, una vez que lo descargemos procedemos a la instalación, el procedimiento de instalación es bastante sensillo, por lo que no lo abordaremos.

En la Figura 2.15 podemos ver una descripción de los componentes de la ventana principal del software Arduino.

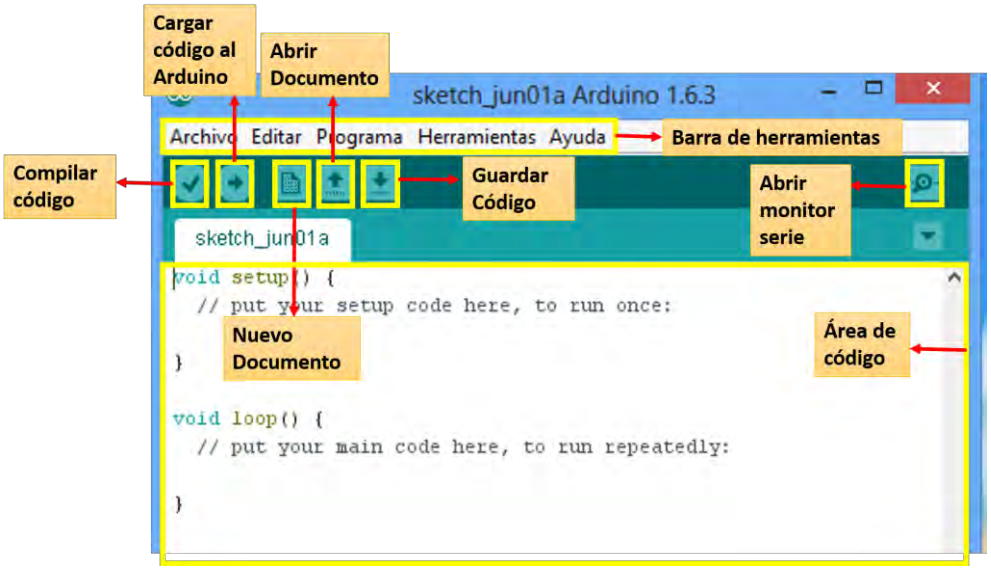


Figura 2.15. Entorno software Arduino.

En la clase *void setup ()* debemos tener declaradas las variables que usaremos, configurar e inicializar los pinMode (declarar los pines como E/S), configuración de la comunicación serie, entre otras.

La clase *void loop ()* tendremos el código que queremos que se ejecute continuamente como podrían ser lecturas de entradas, activación o desactivación de pines según lo que utilicemos.

Ademas de estas dos clases podemos crear nuestras propias clases, las cuales podemos “llamar” dentro de la clase loop para que sea ejecutada continuamente. En el siguiente capitulo veremos mas a detalle como programar nuestro Arduino, incluyendo algunos ejemplos básicos y otros mas avanzados.

Capítulo 3. Desarrollo del trabajo

3.1 Arduino

En este apartado empezaremos a hacer nuestras primeras pruebas con el Arduino UNO. Antes de empezar, haremos una explicación breve sobre los tipos de variables, operadores, condicionales, ciclos, entre otros.

3.1.1 Variables

Empezaremos con las variables, a continuación mostraremos una lista de variables que podemos usar y seguidamente veremos cómo hacer uso de estas. Los tipos de variables que podemos usar son:

- booleano
- char
- byte
- int
- word
- long
- unsigned long
- float
- double
- string
- String
- array

Crear variables es bastante simple, a continuación crearemos algunas variables con diferentes nombre, de diferentes tipos, con y sin asignarle valor inicial, podemos observar algunos ejemplos de variables en la Figura 3.0. De igual manera tenemos declarados algunos arreglos de diferentes tipos, varia un poco la forma de declararlos pero solo debemos seguir la estructura que observamos en los ejemplos.

```
//Variable1, tipo entero con
//valor inicial 4
int variable1=4;
//Variable2, tipo String
String variable2;
//Variable 3, tipo char
//con valor inicial 'C'
char variable3= 'C';
//Variable 4 tipo booleano
//con valos inicial "true"
boolean variable4= true;
//arreglo de enteros de tamaño 8
int array1 [8];
//arreglo de flotantes sin tamaño
//definido pero variables dadas.
float array2 []= {1.9,2.8,3.7,4.6};
//Arreglo de caracteres de tamaño
// 6 y con variables {H, o, l, a}
char array3 [6]= "Hola";
```

Figura 3.0. Ejemplos de Variables.

3.1.2 Operadores

Existen diferentes tipos de operadores, dentro de los cuales podemos encontrar los operadores booleanos, los operadores de comparación y los operadores matemáticos.

Los operadores booleanos son los que mayormente usamos en los condicionales if, tal es el caso de:

- && (Y)
- || (O)
- ! (Negación)

Los operadores de comparación son los que usamos mayormente en los condicionales y ciclos (if, for, while) y son los siguientes:

- == (Igual a)

- =! (Diferente de)
- < (Menor que)
- > (Mayor que)
- <= (Menor o igual)
- >= (Mayor o igual)

Por ultimo tenemos los operadores matemáticos, los cuales usamos, como su nombre lo indica, para hacer todas las operaciones matemáticas que necesitemos, los operadores matemáticos son:

- + (Suma)
- - (Resta)
- * (Multiplicación)
- / (División)
- = (Asignar)
- % (Módulo)

3.1.3 Condicionales y ciclos

Ahora veremos un poco de lo que son los condicionales y ciclos que estaremos usando más adelante. Los condicionales son los que usaremos para realizar una acción después de haber evaluado ciertas condiciones lógicas, y los ciclos, como su nombre lo indica, lo usaremos para repetir cierta acción hasta que se cumpla alguna condición, es importante que exista dicha condición de lo contrario podríamos hacer un bucle infinito, repitiendo la acción infinitas veces. Los ciclos y condicionales que usaremos son:

- If
- Switch
- For
- While

A Continuación pondremos algunos ejemplos usando lo que ya hemos visto, así como también algunas cosas nuevas, pero relativamente sencillas y de fácil entendimiento.

3.1.4 Ejemplos básicos

3.1.4.1 Hola mundo

Empezaremos con el ejemplo tradicional de programación, el “Hola mundo”, para dicho ejemplo usaremos el código que tenemos en la Figura 3.1 donde podemos ver que tenemos dos clases por defecto, void setup y void loop. La primera es donde inicializamos la comunicación serial, definiendo la velocidad a 9600 bps, y seguidamente imprimiremos en pantalla el mensaje de “Hola mundo”, en este caso el loop lo dejaremos vacío, ya que esa clase se utiliza para repetir infinitamente las acciones que se encuentren dentro. En la Figura 3.2 podemos ver el resultado obtenido del código que se usó en este ejemplo.

```
void setup() {
  Serial.begin(9600);
  Serial.println("Hola mundo");
}

void loop() {

}
```

Figura 3.1. Ejemplo Hola mundo.

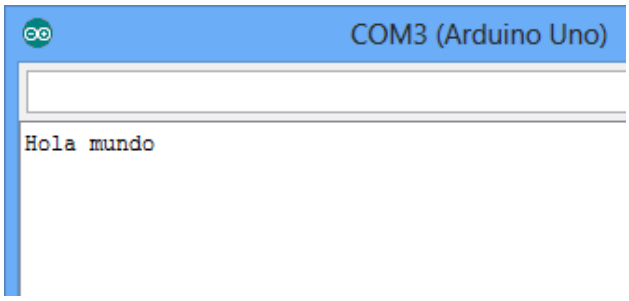


Figura 3.2. Resultado ejemplo Hola mundo.

3.1.4.2 Encender LEDs

El siguiente ejemplo básico será encender dos led de manera alternada, es decir, cuando uno este apagado el otro estará encendido, para esto usaremos la conexión que podemos observar en la Figura 3.3, una vez que tengamos lista la conexión empezaremos a elaborar el código que usaremos para este ejemplo, el código creado lo podemos observar en la Figura 3.4, donde vemos que lo primero que hacemos es declarar dos variables de tipo entero con los números 12 y 13, estos números indican los puertos que usaremos para los leds, en la clase setup inicializamos dichos puertos como salidas, esto quiere decir que en los puertos estaremos enviando pulsos que son los que nos ayudaran a encender los leds. En la clase loop lo que hacemos primero es encender el led 1, y seguidamente apagar el led 2, después le damos un retardo de un segundo, se puede cambiar según qué tan rápido o lento queremos que sea el parpadeo, mientras el numero sea más pequeño, el parpadeo de los leds será más rápido; luego de haber hecho eso procedemos a hacer lo inverso, es decir, primero encendemos el led 2 y seguidamente apagamos el led 1, dejando el mismo tiempo de retardo, con este código haremos que los dos leds se prendan y se apaguen de manera alternada con un segundo entre cada parpadeo.

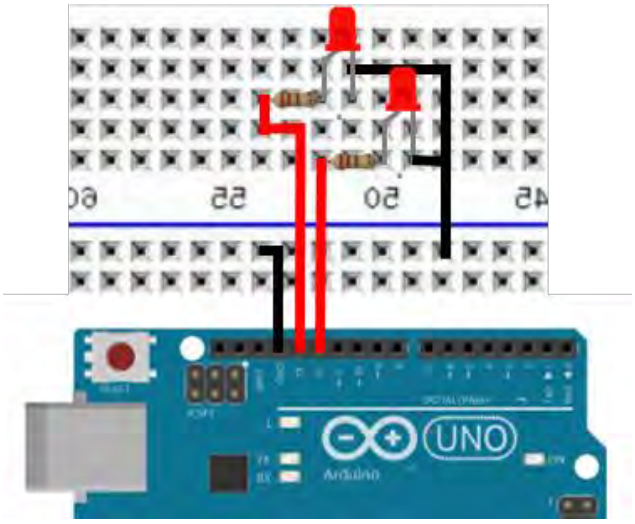


Figura 3.3. Conexión LEDs alternados.

```
//Declaramos los puertos
//que vamos a usar
int led1=12;
int led2=13;
//Inicializamos los puertos
//&como salidas
void setup() {
  pinMode(led1,OUTPUT);
  pinMode(led2,OUTPUT);
}

void loop() {
  //Encendemos el led 1
  digitalWrite(led1,HIGH);
  //Apagamos el led 2
  digitalWrite(led2,LOW);
  //Damos un retardo
  delay(1000);
  //Ahora hacemos lo inverso
  digitalWrite(led2,HIGH);
  digitalWrite(led1,LOW);
  delay(1000);
}
```

Figura 3.4. Código LEDs alternados.

3.1.4.3 Foto resistencia y LED

Ahora veremos un poco acerca de entradas y salidas analógicas con un ejemplo que consiste en encender un led en función de la luz externa, esto con ayuda de la lectura de una foto resistencia, con la cual captaremos la intensidad de la luz en el lugar donde nos encontramos, y haremos que nuestro led brille en función de esa intensidad. Lo primero que debemos hacer son las conexiones que tenemos en la Figura 3.5; ahora entraremos a la parte del código, el cual tenemos en la Figura 3.6, donde podemos ver que iniciamos creando la variable led y dándole un valor de 11, que es el puerto PWM que usaremos que nos permitirá asignarle una salida analógica para indicar con que intensidad brillara el led. Después de esto tenemos la variable luz, la cual nos servirá para asignarle el valor que la foto resistencia nos esté dando, le damos un valor inicial de 0; en la

clase setup inicializaremos el puerto que elegimos como salida, y de igual manera inicializamos la la comunicación serial, ahora en la clase loop, donde primero debemos asignarle a la variable luz la lectura del puero A0 que es donde tenemos conectado nuestra foto resistencia, seguido de esto imprimimos en pantalla dicha lectura para que podamos observar los valores de entrada que se van teniendo y ver cómo funciona el programa, ahora vemos dos condificonales if, cada uno con dos condiciones, lo que hacemos ahí es que en caso de que la variable luz tenga un valor entre 0 y 512 el led brillara con una luz intensa, eso indicado por el valor de 255 en el analogWrite, en el siguiente ciclo tenemos que si la variable tiene un valor entre 512 y 1024 la intensidad de brillo del led disminuirá a 64; este es el código que necesitamos, y para probar nuestro programa, una vez cargado el código en el arduino, podemos abrir el monitor serie y ver las lecturas que nos va dando y comprobar que efectivamente cuando nos da una lectura entre los intervalos dados anteriormente la intensidad de brillo del led incrementa o disminuye según sea el caso.

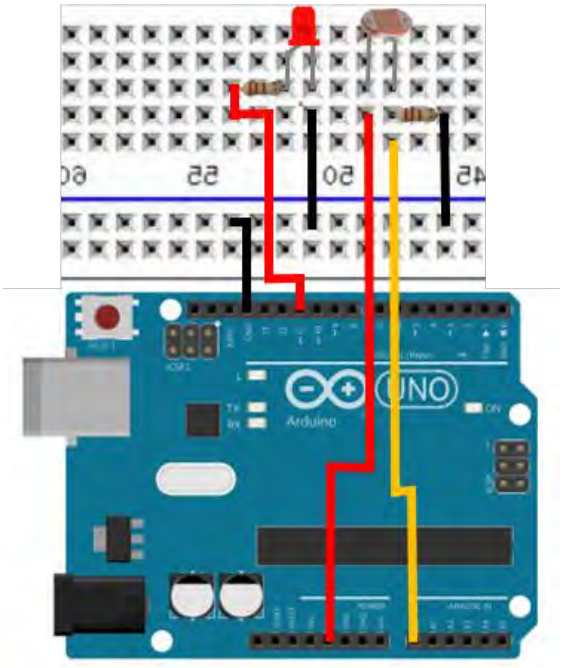


Figura 3.5. Conexión foto resistencia y led.

```
//indicamos el puerto donde conectareos el LED
int led=11;
//variable para lectura de la foto resistencia
int luz=0;
void setup() {
/*Inicializamos nuestro puerto como salida al
igual que la comunicación serial*/
  pinMode(led,OUTPUT);
  Serial.begin(9600);
}
void loop() {
/*Le asignamos a la variable luz el valor de
la foto resistencia*/
  luz=analogRead(A0);
  Serial.println(luz);
/*Ahora de acorde al valor de  entrada que
tengamos, le asignaremos un valor de salida al
puerto del led, lo que nos dara la intensidad
con la que brille este*/
  if (luz<512 && luz >=0){
   analogWrite(led,255);
  }
  if (luz>=512 && luz <=1024){
   analogWrite(led,64);
  }
}
```

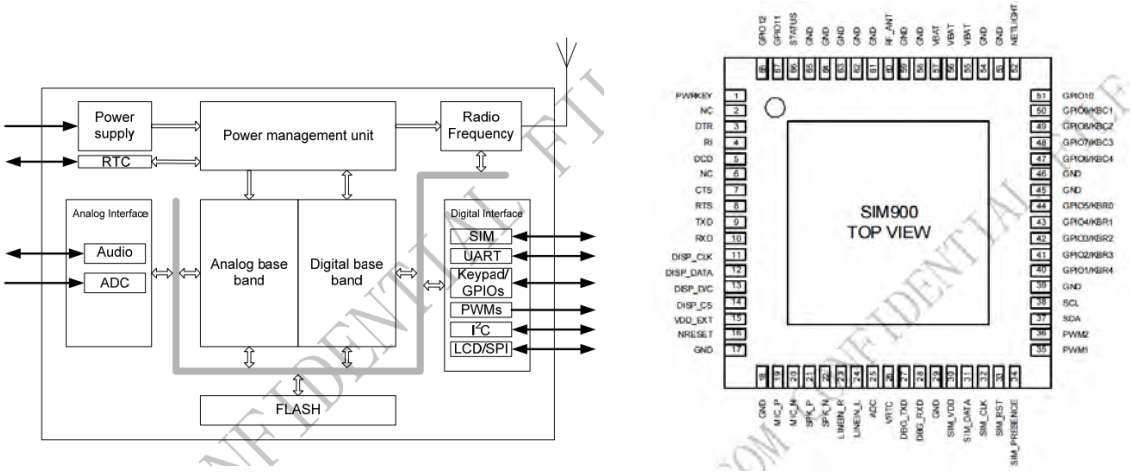
Figura 3.6. Código foto resistencia y led.

Estos son algunos ejemplos básicos, en la página oficial de arduino o en el mismo programa de arduino se encuentran más ejemplos básicos y otras mas complejos donde se pueden empezar a trabajar con diferentes sensores y módulos.

3.2 Arduino GSM

En esta sección empezaremos a hacer las primeras pruebas con el micro controlador Arduino en conjunto con el módulo GSM; buscaremos hacer y recibir llamadas y de igual manera enviar y recibir mensajes.

En este caso trabajaremos con un módulo GSM que tiene integrado un SIM900, del cual, en la Figura 3.7, podemos observar su diagrama funcional y los pines de salida del mismo. Se eligió trabajar con el SIM900 por que soporta la frecuencia de las redes nacionales y por el bajo costo del mismo a comparación de otros integrados.



(a) (b)

Figura 3.7. (a) Diagrama funcional SIM 900, (b) Pines de Salida SIM900 [14].

Para facilitar la interacción del arduino con el SIM900 tenemos la placa GPRS Shield V1.1(b), la cual, como podemos observar en la Figura 3.8, ya nos trae todo lo necesario para poder trabajar en conjunto con el arduino, salidas y entradas de audio para las llamadas, el adaptador para poder utilizar nuestra tarjeta SIM (Subscriber Identity Module), viene con una antena para el escaneo y conexión en la red telefónica de nuestra preferencia, de igual manera nos trae dos indicadores led, uno de encendido (rojo) y el otro nos indicara si hay señal (Verde). Esta tarjeta viene con 9 pines, los cuales nos ayudaran a seleccionar nuestro pin transmisor (Tx) y receptor (Rx) con los que estaremos trabajando.

Figura 3.8. GPRS Shield V1.1 (b).

Para este caso usaremos una de la compañía Telcel, ya que es la que cuenta con una mayor cobertura en el área donde estaremos usando la tarjeta pero, el SIM900 es cuatribanda y puede trabajar a 850/ 900/ 1800/ 1900 MHz, por lo que podremos usar una tarjeta SIM de cualquier compañía nacional.

Para poder trabajar con el módulo GSM debemos tener el arduino conectado a la fuente de alimentación y no solo por cable USB. Estaremos usando el pin 2 como tx y el 3 para rx, por lo que de esa manera debemos tener puenteados los pines, tal como se muestra en la Figura 3.9.

Figura 3.9. Puenteo de pines Tx y Rx.

3.2.1 Recibir llamadas

Para la parte del código, estaremos usando comandos AT como se mostrará a continuación. La primera prueba que haremos será la de recibir una llamada con el arduino; para comenzar debemos abrir el IDE de arduino y empezar con definir las librerías que vamos a usar, en este caso la librería SoftwareSerial, tal como podemos observar en la Figura 3.10, también debemos tener una variable para almacenar el carácter de entrada que nos dará el GSM shield como tono, y para finalizar con esta parte del código crearemos una variable de tipo SoftwareSerial, que en este caso la llamamos SIM900 e indicamos que estaremos trabajando sobre los pines 2 y 3 para enviar y recibir los datos y las variables numring, comring y onoff, las dos primeras las usaremos para que cada tres tonos se encienda/apague un LED, por ese motivo el comring lo inicializamos con un valor de 3, la variable onoff nos servirá para la parte de encender y apagar el LED.

```
#include <SoftwareSerial.h> //incluimos la libreria SofwareSerial
char inchar; // Almacenarálu el caracter entrante
//Creamos una variable de tipo SoftwareSerial y
//le asignamos los puertos con los que va a trabajar
SoftwareSerial SIM900(2, 3); //(Tx, Rx)

int numring=0;
int comring=3;
int onoff=0; // 0 = off, 1 = on
```

Figura 3.10. Código para recibir llamadas 1.

En la Figura 3.11 tenemos la clase setup, en la cual vamos a establecer la velocidad de transmisión de datos en serie indicando que la velocidad será de 9600, luego indicaremos los pines que estarán como salida, seguidamente pasamos a inicializar la tarjeta GSM, la encendemos, indicamos la velocidad de transmisión de datos que será 19200, activamos la notificación de llamadas con el comando AT que se indica en la Figura 3.11.

```
void setup()
{
  Serial.begin(19200);
  //Coniguramos los pines que usaremos en el programa
  pinMode(12, OUTPUT);
  pinMode(13, OUTPUT); // Pines para encender/apagar LEDs
  digitalWrite(12, HIGH);
  digitalWrite(13, LOW);

  // inicializamos la GSM shield
  SIM900encender();
  SIM900.begin(19200); // configuramos el baud rate en 19200
  SIM900.print("AT+CLIP=1\r"); // activamos la notificación
  //de identificación de llamadas
  delay(100);
}
```

Figura 3.11. Código para recibir llamadas 2.

En la Figura 3.12 tenemos las clases SIM900encender y LEDs, las cuales usaremos para encender la tarjeta GSM y para encender y apagar un led que usaremos en la clase loop.

En la Figura 3.13 veremos el estado de la variable SIM900 y en caso de tener alguna entrada, empezara a leer estas y las comparará con los caracteres que se indica, esto para ver si la llamada está dando tono, y en caso de ser así, enviara a la pantalla el texto “ring!” y aumentaremos el contador, cuando el contador sea igual a la variable comring (con valor de 3), se reiniciará el contador se prendera/apagara el LED

```
void SIM900encender()
// Encendemos la GSM Shield
{
  digitalWrite(9, HIGH);
  delay(1000);
  digitalWrite(9, LOW);
  delay(7000);
}

void LEDs()
{ //Encendemos y apagamos un led para verificar la llamada entrante
  if (onoff==0)
  {
    onoff=1;
    digitalWrite(12, HIGH);
    digitalWrite(13, LOW);
    Serial.println("D12 high D13 low");
  }
  else
    if (onoff==1)
    {
      onoff=0;
      digitalWrite(12, LOW);
      digitalWrite(13, HIGH);
      Serial.println("D12 low D13 high");
    }
}
```

Figura 3.12. Código para recibir llamadas 3.

```
void loop()
{ //Una vez que activamos la notificacion de llamadas, verificamos
//que nos este dando tono y mostramos en pantalla el tono "ring"
//simulando el tono de un telefono convencional
  if(SIM900.available() >0)
  {
    inchar=SIM900.read();
    if (inchar=='R')
    {
      delay(10);
      inchar=SIM900.read();
      if (inchar=='I')
      {
        delay(10);
        inchar=SIM900.read();
        if (inchar=='N')
        {
          delay(10);
          inchar=SIM900.read();
          if (inchar=='G')
          {
            delay(10);

            numring++;
            Serial.println("ring!");
            if (numring==comring)
            {
              numring=0; // reiniciamos el contador ring
              LEDs(); //Encendemos y apagamos los LEDs
            }
          }
        }
      }
    }
  }
}
```

Figura 3.13. Código para recibir llamadas 4.

Este es el código que debemos cargar en nuestro arduino una vez que tengamos nuestra tarjeta GSM en el, conectamos a la energía con el alimentador de voltaje, colocamos la antena y el SIM en la tarjeta, después que tengamos el código ya en el arduino debemos verificar que el LED de encendido en la tarjeta este de color rojo; al abrir el monitor serie en el IDE de arduino tendremos que esperar la

llamada entrante, y una vez que este entrando la llamada se mostraran los tonos en la pantalla, tal y como se ve en la Figura 3.14.

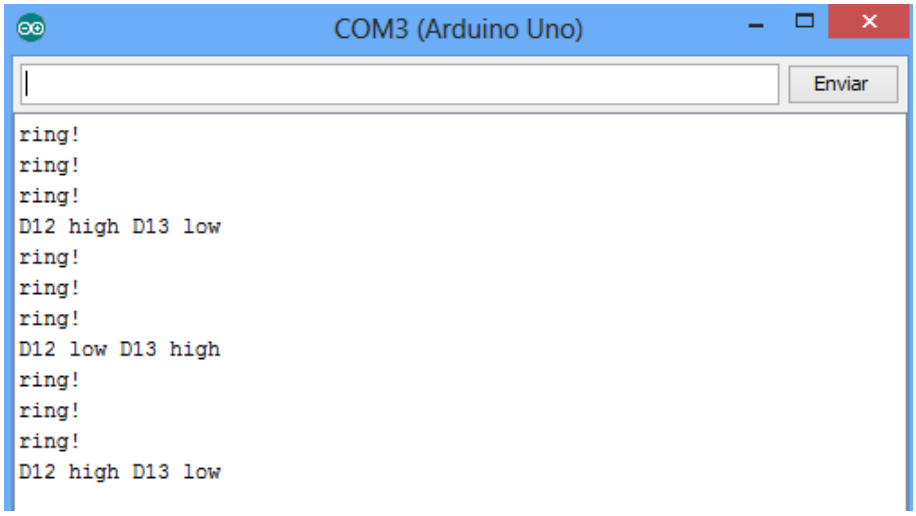


Figura 3.14. Recibiendo llamada en el arduino.

Como podemos observar después de cada tres tonos, estará encendiendo y apagando un LED. Este ejemplo ha sido sencillo y podemos modificarlo para que nos muestre en pantalla más detalles de la llamada tales como el número del que proviene la llamada, la fecha y hora, entre otros más.

3.2.2 Hacer llamadas

Ahora veremos cómo realizar llamadas desde nuestro arduino con ayuda del módulo GSM, las conexiones serán las mismas, y el código muy similar, todos los códigos que veremos para el módulo GSM tiene gran similitud, ya que solo es la clase principal la que cambiara.

En la Figura 3.15 podemos observar lo que se comentó en el párrafo anterior, pero a diferencia que en este código no tendremos variables, ya que no habrá datos de entrada, solo realizaremos la llamada; la clase SIM900 encender tiene la misma función que en el código anterior, encender la tarjeta GSM.

```
#include <SoftwareSerial.h>
SoftwareSerial SIM900(2, 3); // pines Tx y Rx

void setup()
{
  Serial.begin(9600);
  SIM900.begin(19200);
  SIM900encender();
  delay(20000);  // tiempo de espera para conectar a la red
}

void SIM900encender()
// Encendemos la tarjeta GSM
{
  digitalWrite(9, HIGH);
  delay(1000);
  digitalWrite(9, LOW);
  delay(5000);
}
```

Figura 3.15. Código realizar llamadas 1.

En la Figura 3.16 podemos observar la clase realizarLlamada, en donde, como su nombre lo indica, realizaremos la llamada al número que nosotros queramos. Primero indicamos al usuario que ya estaremos realizando la llamada, esto enviando a pantalla el texto “Haciendo llamada…”, luego enviaremos por el serial SIM900 que creamos en un inicio el comando “ATD +” seguido del símbolo “+”, la clave del país el número de teléfono a 10 dígitos; en este caso la clave de México es 52, y el número a realizar la llamada el 9831073065, por lo que el comando AT completo quedaría de la siguiente manera: “ATD + +529831073065”. Damos un retardo de 100 milisegundos para que se enlace la llamada, después de eso, damos un retardo de 30 segundos, este será el tiempo que va a durar la llamada, conteste o el receptor, después de los 30 segundos calgaremos con el comando “ATH”.

```
void realizarLlamada()
{ Serial.println("Haciendo llamada...");
  //Número al que vamos a marcar
  SIM900.println("ATD + +529831073065;");
  delay(100);
  SIM900.println();
  delay(30000);            // esperamos 30 segundos
  SIM900.println("ATH");   // colgamos
  Serial.println("Llamada Terminada");
}

void loop()
{ // mandamos llamar las clases
  realizarLlamada();
  SIM900encender();
  //esta linea nos servira para que no intente realizar
  //otra llamada
  do {} while (1);
}
```

Figura 3.16. Código para realizar llamadas 2.

Al final tenemos la clase loop, donde podemos observar solo se manda a llamar a las clases creadas anteriormente, y la última línea de código tiene la función de que el programa solo realice una llamada por cada vez que se ejecute. Este es todo el código que necesitamos para realizar una llamada, como pudimos observar es más sencillo que el primer código que realizamos.

3.2.3 Recibir Mensajes

Ahora entraremos al tema de los mensajes, lo cual no hay mucha diferencia con lo visto anteriormente, solo cambian los comandos “AT” que estaremos usando. Para empezar veremos cómo recibir mensajes con el arduino, como hemos comentado anteriormente, las conexiones son las mismas, solo cambiaran partes del código que estaremos usando.

En la Figura 3.17 podemos observar que se inicia de la misma manera que los ejemplos anteriores, importando la librería SoftwareSerial y creando la variable SIM900, en este caso tenemos una variable extra, llamada “caract_entrante”, la cual nos servirá para recibir los caracteres que el módulo GMS nos vaya

enviando, para poder mostrarlo en pantalla. Luego tenemos la clase setup donde inicializamos la comunicación serial y la comunicación con el módulo GSM, lo encendemos e iniciamos en modo texto SMS, con los comandos “AT” que se muestran en la figura, con esto el módulo quedará en espera de cualquier mensaje entrante. Seguidamente tenemos la ya conocida clase SIM900encender.

```
#include <SoftwareSerial.h>
SoftwareSerial SIM900(2, 3);

char caract_entrante=0;

void setup()
{
  Serial.begin(19200); // inicializamos comunicacion serial
  SIM900.begin(19200); // inicializamos comunicacion con el modulo GSM
  SIM900encender();  // Encendemos el modulo GSM
  delay(20000);  // reardo de espera para conectar a la red
  //inicializamos el modulo GSM como modo texto SMS
  SIM900.print("AT+CMGF=1\r");
  delay(100);
  SIM900.print("AT+CNMI=2,2,0,0,0\r");
  delay(100);
}

void SIM900encender()
// encender el modulo GSM
{
  digitalWrite(9, HIGH);
  delay(1000);
  digitalWrite(9, LOW);
  delay(7000);
}
```

Figura 3.17. Código para recibir mensajes 1.

En la Figura 3.18 tenemos el resto del código, que es la clase loop, en donde no haremos más que leer lo que el módulo GSM nos esté enviando, almacenarlo en

la variable caract_entrante y luego imprimirlo en el monitor serie para que el usuario pueda visualizarlo.

```
void loop()
{
  //Mostramos en el monitor serie cualquier texto que el modulo GSM envie
  if(SIM900.available() >0)
  {
    //obtenemos el caracter que proviene del puerto serial
    caract_entrante=SIM900.read();
    //imprimimos el caracter recibido en el monitor serie
    Serial.print(caract_entrante);
  }
}
```

Figura 3.18. Código para recibir mensajes 2.

Una vez que compilemos y carguemos el código al arduino, ya podremos recibir mensajes de cualquier parte del mundo, como resultado tendremos algo como lo que aparece en la Figura 3.19.

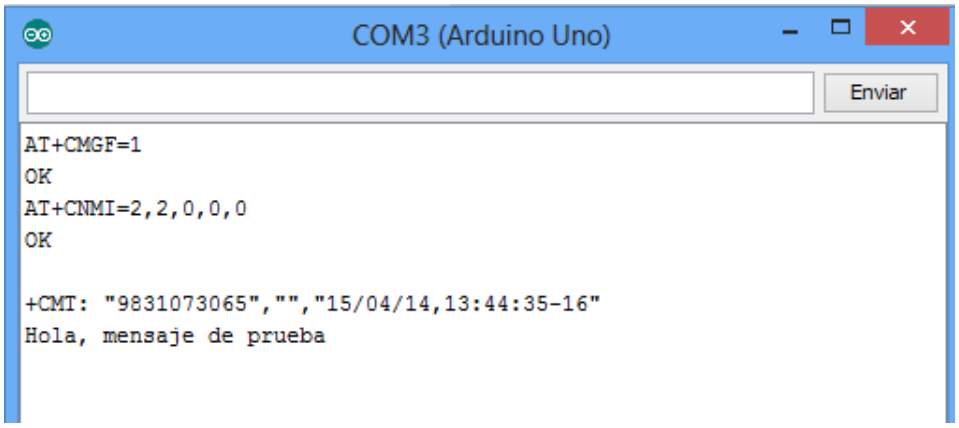


Figura 3.19. Recibiendo mensaje en el arduino.

3.2.4 Enviar mensaje

Para terminar con estos ejemplos básicos que tenemos para el módulo GSM tenemos el ejemplo de enviar un mensaje, para empezar tenemos la Figura 3.20, la cual nos va a mostrar que el código de este ejemplo es sencillo, solo seguimos

la base de los ejemplos anteriores, primero importamos la librería "SoftwareSerial", creamos la variable SIM900 ese tipo. En la clase setup, inicializamos la comunicación serial y con el módulo GSM, lo encendemos y le damos un tiempo de espera para que se conecte a la red; como ya hemos visto e los ejemplos anteriores aquí igual tenemos la clase "SIM900encender" que, como ya habíamos explicado, nos servirá para encender el módulo GSM y este se pueda conectar a la red, por lo que siempre va a estar en nuestro códigos relacionados con este módulo.

```
#include <SoftwareSerial.h>
SoftwareSerial SIM900(2, 3);

void setup()
{
  Serial.begin(9600);
  SIM900.begin(19200);
  SIM900encender();
  delay(20000);  // tiempo de espera para conectar a la red
}

void SIM900encender()
// encender el modulo GSM
{
  digitalWrite(9, HIGH);
  delay(1000);
  digitalWrite(9, LOW);
  delay(5000);
}
```

Figura 3.20. Código para enviar mensaje de texto 1.

En la Figura 3.21 tenemos la parte diferente a los otros códigos, la clase que nos va a ayudar a enviar el mensaje que queramos, con ayuda de los comandos "AT" nuevamente, primero indicamos al usuario que ya estamos enviando el mensaje, seguidamente inicializamos el envió de mensajes con el comando "AT+CMGF=1", después de esto tenemos el comando "AT+CMGS=\"+529831073065\"", en el cual tenemos el número al cual se va a enviar el mensaje, después tenemos lo que es el texto que enviaremos, que en

este caso es el clásico “Hola mundo, Arduino”; finalizamos los comandos y damos un retardo para que se envié el mensaje e indicamos al usuario que el mensaje ha sido enviado. Para finalizar el programa tenemos la clase loop donde mandaremos a llamar a la clase “enviarSMS” y con la última línea de código estamos indicando que solo envié el mensaje una vez

```
void enviarSMS()
{
  Serial.println("Enviando mensaje...");
  //inicializamos el modulo para enviar el mensaje
  SIM900.print("AT+CMGF=1\r");
  delay(100);
  //Proporcionamos el número al cual será enviado el mensaje
  SIM900.println("AT + CMGS = \"+529831073065\"");
  delay(100);
  //Aqui va el texo que queramos enviar
  SIM900.println("Hola Mundo, Arduino");
  delay(100);
  //Terminar el comando "AT" con ^Z, código ASCII 26
  SIM900.println((char)26);
  delay(100);
  SIM900.println();
  //Retardo para que el modulo GSM envíe el mensaje
  delay(5000);    // give module time to send SMS
  //apagamos el modulo GSM
  SIM900power();
  Serial.println("Mensaje Enviado");
}
void loop()
{ //Llamamos la clase enviarSMS
  enviarSMS();
  do {} while (1);
}
```

Figura 3.21. Código para enviar mensaje de texto 2.

3.3 GPS

Empezaremos las pruebas con el módulo GPS, para esto usaremos el módulo GY-GPS6MV2, que trabaja con Ublox NEO-6M-0-001, del cual podemos ver las especificaciones en la Figura 3.22 y en la Figura 3.23 podemos observar el módulo que estaremos usando en esta sección.

Parameter	Specification			
Receiver type	50 Channels GPS L1 frequency, C/A Code SBAS: WAAS, EGNOS, MSAS			
Time-To-First-Fix[1]		NEO-6G/Q/T	NEO-6M/V	NEO-6P
	Cold Start[2]	26 s	27 s	32 s
	Warm Start[2]	26 s	27 s	32 s
	Hot Start[2]	1 s	1 s	1 s
	Aided Starts[3]	1 s	<3 s	<3 s
Sensitivity[4]		NEO-6G/Q/T	NEO-6M/V	NEO-6P
	Tracking & Navigation	-162 dBm	-161 dBm	-160 dBm
	Reacquisition[5]	-160 dBm	-160 dBm	-160 dBm
	Cold Start (without aiding)	-148 dBm	-147 dBm	-146 dBm
	Hot Start	-157 dBm	-156 dBm	-155 dBm
Maximum Navigation update rate		NEO-6G/Q/M/T	NEO-6P/V	
		5Hz	1 Hz	
Horizontal position accuracy[6]	GPS	2.5 m		
	SBAS	2.0 m		
	SBAS + PPP[7]	< 1 m (2D, R50)[8]		
	SBAS + PPP[7]	< 2 m (3D, R50)[8]		
Configurable Timepulse frequency range		NEO-6G/Q/M/P/V	NEO-6T	
		0.25 Hz to 1 kHz	0.25 Hz to 10 MHz	
Accuracy for Timepulse signal	RMS	30 ns		
	99%	<60 ns		
	Granularity	21 ns		
	Compensated[9]	15 ns		
Velocity accuracy[6]		0.1m/s		
Heading accuracy[6]		0.5 degrees		
Operational Limits	Dynamics	≤ 4 g		
	Altitude[10]	50,000 m		
	Velocity[10]	500 m/s		

Figura 3.22. Especificaciones Ublox NEO-6M-0-001 [15].

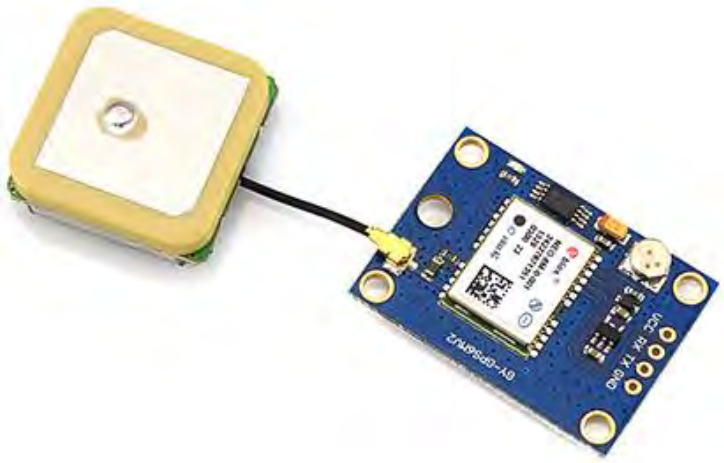


Figura 3.23. Módulo GY-GPS6MV2.

Como podemos observar en la Figura 3.23 el módulo GPS viene con cuatro pines VCC, RX, TX, GND los cuales conectaremos al arduino tal y como se muestra en la Figura 3.24, el pin Rx del módulo no es necesario conectarlo, ya que esté solo estará enviando las coordenadas, no estará recibiendo ninguna información.

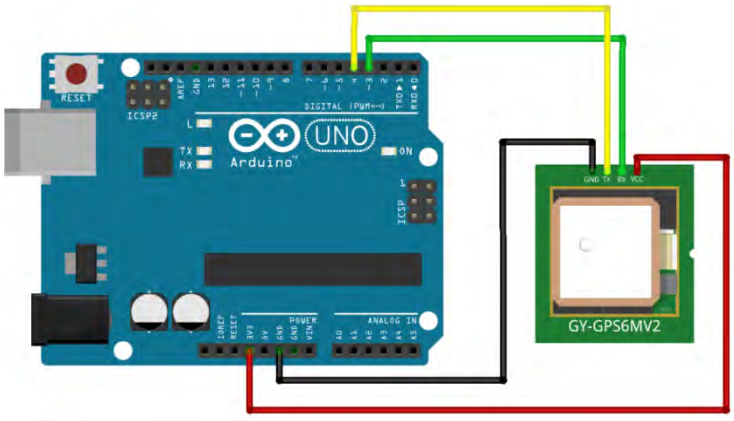


Figura 3.24. Conexión Arduino-Módulo GPS.

Una vez que tenemos la conexión procedemos a la parte del código que como podremos ver a continuación es bastante sencilla usando la librería TinyGPS++. En la Figura 3.25 podemos ver las primeras líneas de código donde incluimos las librerías TinyGPS++ y SoftwareSerial, creamos un objeto llamado gps y otro

llamado ss para el software serial indicando que usaremos los pines dos y tres para Tx y Rx, aún que como mencionamos en el párrafo anterior, no es necesario conectar el pin 3 que es el que asignamos para Rx.

```
//Librerias para GPS
#include <TinyGPS++.h>
#include <SoftwareSerial.h>
//Objeto GPS
TinyGPSPlus gps;
//Software serial GPS. Pin 2 a TX del GPS. Pin 3 no es necesario conectarlo.
SoftwareSerial ss(2, 3);
void setup() {
  //Init Serial GPS
  ss.begin(9600);
  Serial.begin(9600);

  }
}
```

Figura 3.25. Código GPS 1.

En la Figura 3.26 podemos observar la clase Coordenadas, en la cual ya tenemos el llamado de coordenadas e información extra que podemos usar como información complementaria, tal como el número de satélites disponibles, hora, fecha, altura, la velocidad que son los que estaremos usando. En el código solo verificamos que esté disponible la conexión del GPS, y después de eso empezamos a hacer la petición de los datos que necesitemos e imprimimos lo recibido en pantalla.

```
void Coordenadas() {
   for (int i=0;i<16;i++){
  //Lee los datos recibidos por el serial del GPS
  while (ss.available() > 0)
  if (gps.encode(ss.read()));
  delay (100);
  //imprime coordenadas
  Serial.println("Latitud:");
  Serial.println(gps.location.lat(),6);
  Serial.println("Longitud:");
  Serial.println(gps.location.lng(),6);
  Serial.println("Fecha(DDMMAA):");
  Serial.println (gps.date.value ());
  Serial.println("Hora:");
  Serial.println (gps.time.value ()); // tiempo Raw en formato HHMMSSCC
  Serial.println("Velocidad(km/h):");
  Serial.println (gps.speed.kmph ()); // velocidad en k/h
  Serial.println("Altura:");
  Serial.println (gps.altitude.meters ()); // La altura en metros
  Serial.println("Numero de Satelites en uso:");
  Serial.println (gps.satellites.value ()); // Número de satélites en uso
  delay (100);

}
```

Figura 3.26. Código GPS 2.

En la Figura 3.27 podemos ver los resultados del código. Este es todo el proceso para trabajar con el módulo GPS y como ya habíamos mencionado en párrafos anteriores y como pudimos ver durante este proceso, la librería TinyGPS++ nos facilita el trabajo y hace que sea un proceso bastante corto, sencillo y de fácil entendimiento.

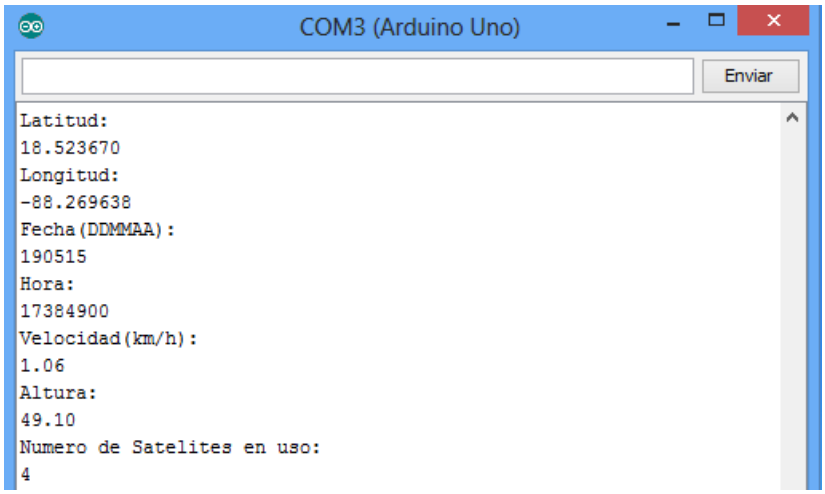


Figura 3.27. Recibiendo información del GPS.

Una vez que tenemos los datos podemos verificar que las coordenadas sean correctas, usando Google Maps. Abrimos el explorador, buscamos la página de Google Maps (https://www.google.com.mx/maps) y en la parte del buscador de direcciones, colocamos las coordenadas que obtuvimos separadas por una coma (,). En la Figura 3.28 podemos observar que efectivamente las coordenadas recibidas pertenecen al lugar donde se estuvieron haciendo las pruebas, en este caso la Universidad de Quintana Roo.

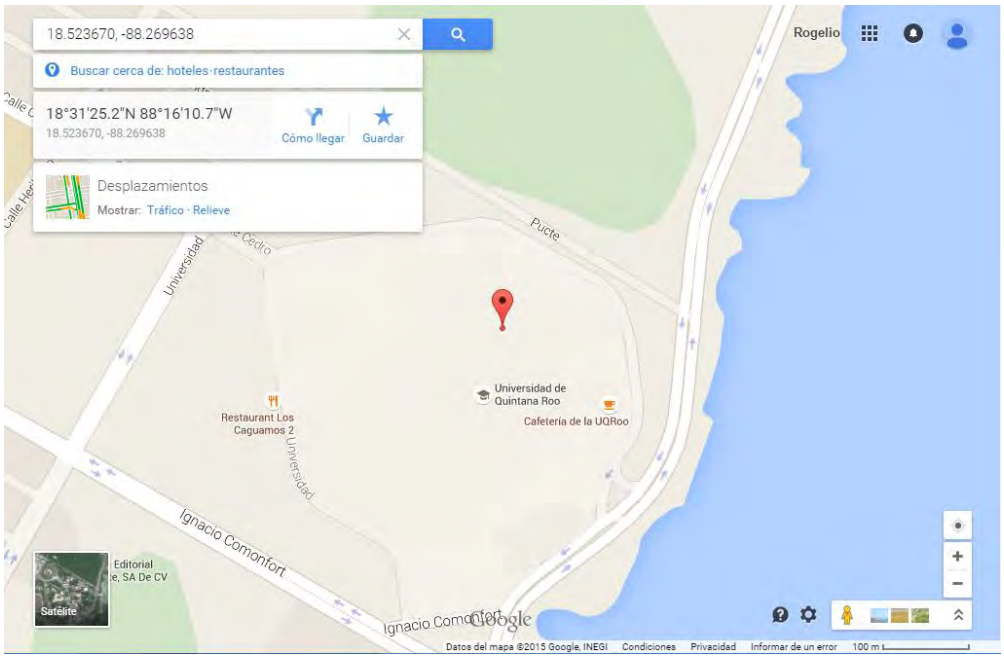


Figura 3.28. Coordenadas en Google Maps.

3.4 Micro-SD

Ahora continuaremos trabajando con un módulo micro-SD con el cual podemos crear un archivo y almacenarlo dentro de una tarjeta micro SD. Para poder hacer esto debemos tener el módulo Micro-SD que vemos en la Figura 3.29 o también se puede hacer desde el módulo Ethernet de Arduino, el cual podemos observar en la Figura 3.30. Veremos cómo usar los dos módulos, la única diferencia es la manera de conectarlo con el Arduino. La ventaja del módulo Ethernet es que además del slot para la micro-SD contamos con una entrada para un plug RJ-45 con el que podremos tener acceso a internet, pero en este proyecto no usaremos esa tarjeta, la ventaja del módulo que usaremos es su reducido tamaño.

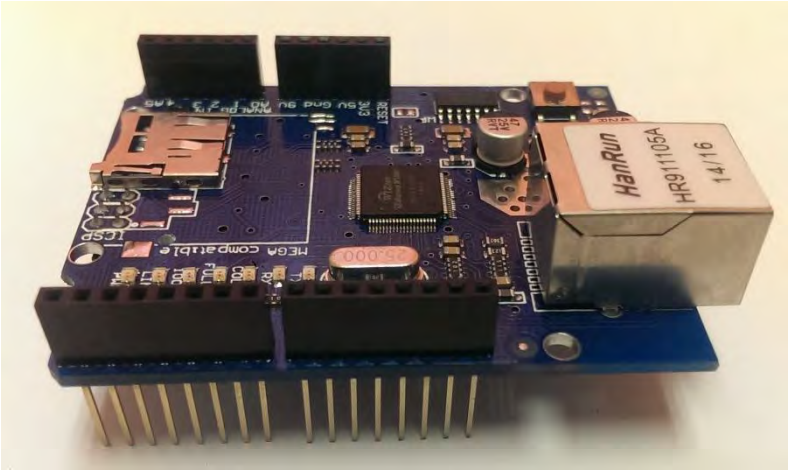


Figura 3.29. Módulo Ethernet Arduino.

Figura 3.30. Módulo Micro-SD Catalex.

Para empezar podemos ver la manera de conectar los dos módulos, como podemos ver en la Figura 3.29 el modulo Ethernet cuenta con los mismos pines que el Arduino, por lo que solo necesitamos empotrarlo a este.

Antes de empezar la conexión con el modulo SD, debemos tener en cuenta los pines SPI (Interfaz Periferico Serial) de Arduino, los cuales podemos observar en la Figura 3.31 y a continuación una descripción de la función de cada pin.

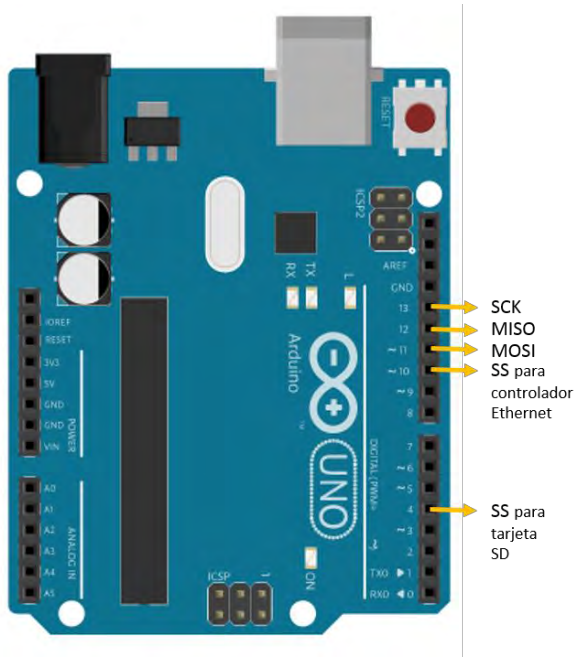


Figura 3.31. Pines SPI.

- SCK (Serial Clock): Los impulsos de reloj que sincronizan la transmisión de datos generada por el maestro.
- MISO (Master In Slave Out): La línea esclavo envía datos al maestro.
- MOSI (Master Out Slave In): La línea maestro envía datos al esclavo.
- SS (Slave Select): El pin en cada dispositivo que el maestro puede utilizar para activar y desactivar dispositivos específicos.

En la Figura 3.32 podemos observar como conectar nuestro módulo al Arduino, una vez que tenemos la conexión procedemos a insertar la tarjeta micro-SD y pasaremos a la parte del código.

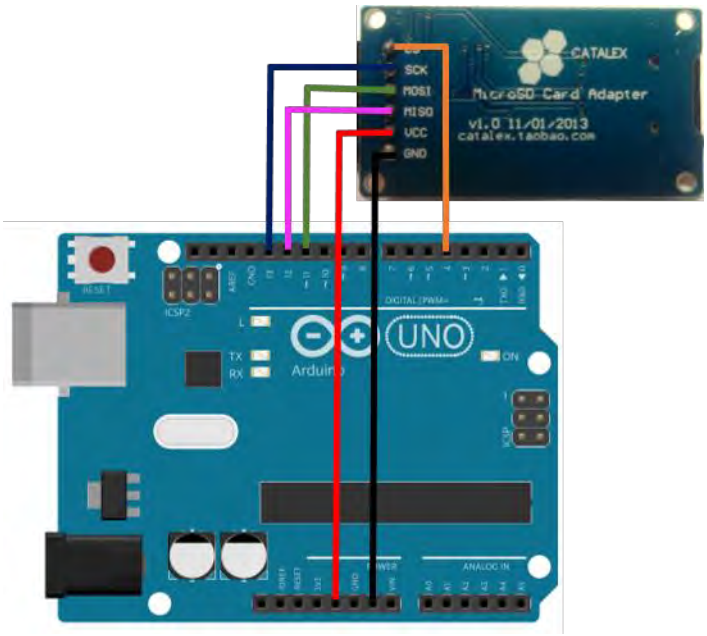


Figura 3.32. Conexión Arduino a módulo micro-SD.

3.4.1 Listar archivos de la tarjeta micro-SD

El primer programa que haremos con este módulo será crear una lista con los archivos que contiene nuestra memoria micro-SD, para eso usaremos le código que vemos en la Figura 3.33 donde primero importamos las librerías SPI y SD, luego tenemos la clase setup en la cual vamos a inicializar la comunicación serial y a verificar que podemos inicializar la tarjeta micro-SD para leerla y en caso de poder hacerlo llamamos a la clase enlistarArchivos, la cual vamos a examinar más adelante.

```
//Incluimos las librerias SPI y SD
#include <SPI.h>
#include <SD.h>
//Creamos una variable de tipo File
File root;

void setup()
{
  // Inicializar la comunicación serial
  Serial.begin(9600);
  Serial.println("Inicializando Tarjeta micro-SD");
  //En caso de no poder leer la tarjeta
  //nos mostrara un mensaje de error
  if (!SD.begin(4)) {
    Serial.println("ERROR! No se pudo leer la tarjeta");
    return;
  }
  //Si se inicializa la tarjeta correctamente nos lo indicará
  Serial.println("Tarjeta inicializada");

  root = SD.open("/");
  //Llamamos ala clase enlistarArchivos
  enlistarArchivos(root, 0);
  Serial.println("Terminado, no se encontraron mas archivos");
}
```

Figura 3.33. Código enlistar archivos micro-SD 1.

En la Figura 3.34 tenemos la clase loop, la cual dejaremos vacía, y la clase enlistarArchivos donde haremos el procedimiento principal para enlistar los archivos de la tarjeta, primero veremos si existen archivos disponibles, y de ser ese el caso imprimiremos su nombre y después su extensión.

Una vez que hayamos compilado y cargado el código podremos abrir el monitor serie para ver los resultados de nuestro código. En la Figura 3.35 podemos ver el resultado que nos da el código.

```
void loop() {
  //En la clase loop no haremos ni una acción
}

  //En esta clase enlistaremos los archivos
void enlistarArchivos(File dir, int numTabs) {
   while(true) {
     File entry =  dir.openNextFile();
     if (! entry) {
       /* Si ya no hay algun archivo que mostrar
       la clase termina y nos mostrara el mensaje
       que se encuentra en la parte de setup:
       "terminado, no se encontraron mas archivos"*/
       break;
     }
     for (uint8_t i=0; i<numTabs; i++) {
       Serial.print('\t');
     }
     //Imprimimos en pantalla el nombre del archivo
     Serial.print(entry.name());
     if (entry.isDirectory()) {
       Serial.println("/");
       enlistarArchivos(entry, numTabs+1);
     } else {
       // Imprimimos en pantalla el tamaño de cada
       //uno de los archivos.
       Serial.print("\t\t");
       Serial.println(entry.size(), DEC);
     }
     entry.close();
   }
}
```

Figura 3.34. Código enlistar archivos micro-SD 2.

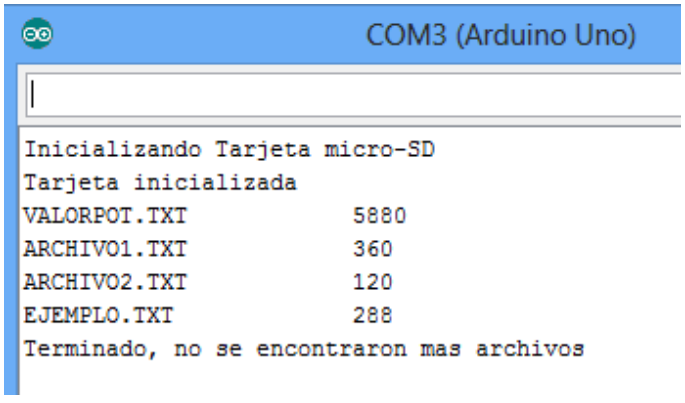


Figura 3.35. Resultado enlistar archivos micro-SD.

3.4.2 Crear archivos en la tarjeta micro-SD

Ahora veremos cómo crear un archivo y escribir en este, o escribir en un archivo ya existente. Una vez que tenemos conectado correctamente nuestro módulo microSD al Arduino continuamos con la elaboración del código.

En la Figura 3.36 tenemos la primera parte del código que como vemos tenemos incluidas las librerías SPI y SD, después de esto tenemos una variable de tipo File, la cual estaremos usando más adelante, seguidamente tenemos la clase setup, que es donde tendremos todo nuestro código; primero inicializamos la comunicación serial como ya hemos visto en repetidas ocasiones y le mandamos un mensaje al usuario para que sepa que ya se inició el programa y estamos inicializando la tarjeta de memoria microSD, en caso de no poder leer la tarjeta debemos mandar un mensaje de error para que el usuario sepa que algo no está funcionando correctamente y debemos revisar que la conexión sea correcta, si se puede leer la tarjeta sin problema alguno se lo indicamos al usuario enviando un mensaje. En la siguiente parte del código vamos a proceder a abrir el archivo estado con la línea de código microSD= SD.open(“Prueba1.txt”, FILe_WRITE); donde podemos notar que microSD es el nombre de la variable que creamos al inicio del programa, y Prueba1.txt es el nombre que le daremos a nuestro archivo, o en caso de tener un archivo ya creado donde queramos escribir pues le indicamos el nombre de dicho archivo; en caso de poder escribir correctamente en el archivo le indicamos al usuario que estamos escribiendo y con la línea de código microSD.println(“”); dentro de los paréntesis podemos poner lo que queremos escribir en el archivo, para terminar cerramos el archivo con el comando microSD.close(); e indicamos que el proceso se realizó correctamente, en caso de no poder escribir en el archivo enviamos un mensaje de error con ayuda de un else. Este sería todo el código que como podemos observar es bastante sencillo y de fácil entendimiento, la clase loop se quedara vacia.

```
#include <SPI.h>
#include <SD.h>

File microSD;

void setup() {
  Serial.begin(9600);
  Serial.print("Inicializando microSD");

  //El pin 4 es el que usaremos como CS en la conexión
  pinMode(4, OUTPUT);

  //Si no podemos inicializar la memoria
  //enviamos un mensaje de error
  if (!SD.begin(4)) {
    Serial.println("Error al inicializar");
  return;}
  Serial.println("Se inicializo correctamente");

  //Abrimos el archivo para escribir en el
  microSD = SD.open("Prueba1.txt", FILE_WRITE);

  /*Si el archivo se abrió correctamente
   procedemos a la escritura*/
  if (microSD) {
    Serial.println("**Escribiendo en el archivo**");
    microSD.println("Prueba de escritura con Arduino");

    // cerramos el archivo
    microSD.close();
    Serial.println("Escritura correcta");
  } else {
    // En caso de no poder abrir el archivo
    // se envia un mensaje de error
    Serial.println("Error abriendo el archivo");
  }
}

void loop() {
}
```

Figura 3.36. Código escritura en microSD.

En la Figura 3.37 podemos observar lo que el Arduino nos manda en el monitor serie, y con ayuda de un lector de tarjetas, podemos examinar que efectivamente se creó el archivo con el nombre que elegimos, y si abrimos dicho archivo verificaremos que el archivo contiene las palabras que le indicamos en el código, tal como lo podemos ver en la Figura 3.38.

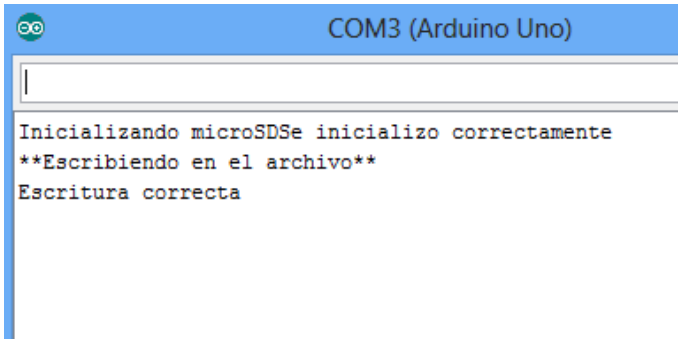


Figura 3.37. Resultados de escritura en microSD 1.

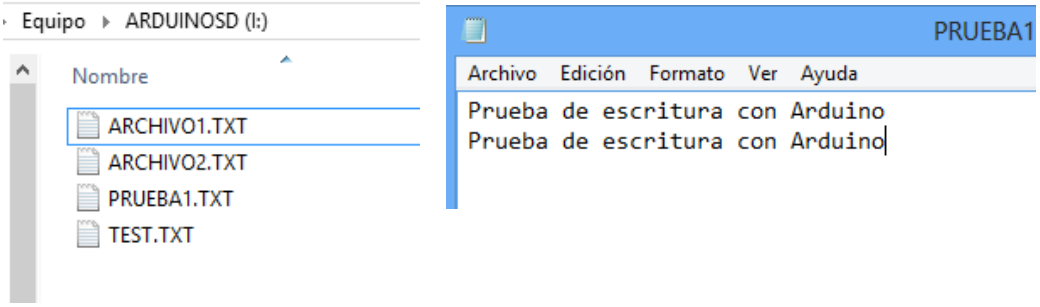


Figura 3.38. Resultados de escritura en microSD 2.

3.4 Sistema Final

Lo visto hasta ahora son las bases para poder desarrollar el sistema para seguridad vehicular, el cual, es el objetivo principal de este trabajo de tesis. Como se mencionó al principio del documento el sistema consiste en enviar un mensaje de texto desde cualquier teléfono al número del SIM que contiene el sistema con el código clave que en este caso es ‘AB22’ y este código devolverá un mensaje de texto a un número ya registrado en el sistema con un enlace que contiene una consulta a Google Maps con las coordenadas en tiempo real del sistema y el cual podrá ser abierto desde cualquier teléfono inteligente. Además, el sistema estará guardando automáticamente en una memoria microSD la ubicación del automóvil cada dos minutos (este tiempo puede ser modificable en los códigos fuente), creando un archivo de Excel con el día, la hora, la latitud y longitud, esto para tener un registro del recorrido que tuvo el automóvil durante el día.

Empezaremos con la parte física del sistema, integrando todas las partes que lo conforman, tal como podemos ver en la Figura 3.39, el sistema está conformado por el microcontrolador Arduino, el módulo GPS, el módulo SD y el módulo GSM, este último con su respectiva SIM; el módulo GSM únicamente hay que empotrarlo en el Arduino y seguir las indicaciones que se vieron anteriormente en el apartado Arduino GSM, de igual manera se vio como conectar el módulo GPS y el módulo SD, pero en esta ocasión conectaremos el tx del módulo GPS al puerto 6 del Arduino. En la Figura 3.40 Podemos observer un grafico del funcionamiento basico del sistema para seguridad vehicular. Un punto importante que se mencionó anteriormente pero vale la pena recalcarlo, es que para poder hacer pruebas usando el módulo GSM es necesario tener conectado el Arduino a una fuente de alimentación de 9 volts.

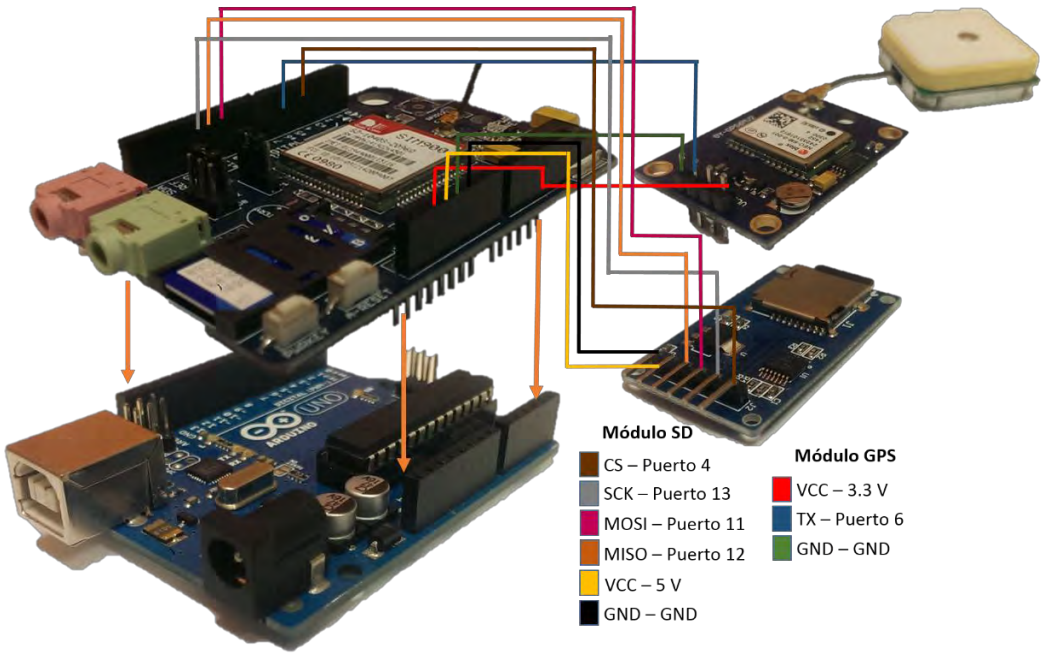


Figura 3.39. Conexión sistema para seguridad vehicular.

Figura 3.40. Funcionamiento básico del sistema.

Una vez que tenemos las conexiones correctamente, procedemos a elaborar el código que estaremos usando. En la Figura 3.41 tenemos la primera parte del código, la cual consiste en importar las librerías que estaremos usando que son SoftwareSerial, SPI, SD y TinyGPS++, seguidamente creamos las variables y objetos que estaremos empleando, en el caso de los objetos SIM900 y ss, debemos asignar los puertos tx y rx que estaremos usando.

```
/*Incluimos las librerias
 que usaremos, las cuales ya
 hemos usado anteriormente */
#include <SoftwareSerial.h>
#include <TinyGPS++.h>
#include <SPI.h>
#include <SD.h>
/*Creamos todas las variables
que usaremos, SIM900 se usará
para recibir y enviar mensajes,
ss se usará para el módulo GPS*/
TinyGPSPlus gps;
File microSD;
SoftwareSerial SIM900(2, 3);
SoftwareSerial ss(6, 7);
char caract_entrante;
double latitud;
double longitud;
int hora, minutos;
int dia, mes, anho;
String mensaje;
long int i=0;
```

Figura 3.41. Código sistema 1.

En la clase setup, la cual podemos ver en la Figura 3.42 inicializaremos la comunicación serial, la comunicación con módulo GPS así como también la comunicación con el módulo GSM, los dos primeros con una velocidad de 9600 baudios y el último mencionado con una velocidad de 19200 baudios; indicamos que el pin 4 será una salida e inicializamos la SD indicando que estaremos usando dicho puerto, después procedemos a llamar a la clase SIM900encender, la cual va a encender el módulo GSM y se le da un tiempo de espera para que

este se conecte a la red. Por último inicializamos el módulo GSM en modo texto SMS para que pueda recibir mensajes de texto.

```
void setup()
{
  // inicializamos comunicacion serial
  Serial.begin(9600);
  // comunicación con el módulo GPS
  ss.begin(9600);
  // comunicación con el módulo GSM
  SIM900.begin(19200);
  //pin 4 para SD como salida
  pinMode(4, OUTPUT);
  SD.begin(4);
  // Encendemos el modulo GSM
  SIM900encender();
  // tiempo de espera para conectar
  delay(20000);
  //inicializamos el módulo GSM
  //como modo texto SMS
  SIM900.print("AT+CMGF=1\r");
  delay(100);
  SIM900.print("AT+CNMI=2,2,0,0,0\r");
  delay(100);
}
```

Figura 3.42. Código sistema 2.

En la Figura 3.43 tenemos dos clases que ya hemos visto anteriormente, la clase SIM900encender, la cual nos servirá para encender el módulo GSM y este pueda conectarse a la red, y la segunda es la clase enviarSMS, que como podemos observar es muy similar a la que hemos visto con anterioridad. Lo primero que tenemos en esta clase es inicializar el módulo para el envío de mensajes de texto, daremos un retardo mínimo y procederemos a proporcionar el número al cual se estará enviando el mensaje de texto, esto con ayuda de los comandos AT y la sentencia SIM900.println, con la cual estaremos enviando los comandos AT; seguidamente asignamos el texto que se enviará en el mensaje con ayuda de la variable mensaje, lo que vamos a proporcionar será un link el cual contendrá una

consulta con en Google maps con las coordenadas que recibimos con el módulo GPS.

```
void SIM900encender() {
  // encender el modulo GSM
  digitalWrite(9, HIGH);
  delay(1000);
  digitalWrite(9, LOW);
  delay(7000);
}

void enviarSMS(){
  //inicializamos el modulo para enviar el mensaje
  SIM900.print("AT+CMGF=1\r");
  delay(100);
  //Proporcionamos el número al cual será enviado el mensaje
  SIM900.println("AT + CMGS = \"+529831073065\"");
  delay(100);
  //Aqui va el texo que queramos enviar
  mensaje = "http://maps.google.es/?q=" + String(latitud, 6)
          + "%20" + String(longitud, 6);
  SIM900.println(mensaje);
  delay(100);
  //Terminar el comando "AT" con ^Z, código ASCII 26
  SIM900.println((char)26);
  delay(100);
  SIM900.println();
  //Retardo para que el modulo GSM envie el mensaje
  delay(5000);
}
```

Figura 3.43. Código sistema 3.

La clase Coordenadas, que podemos observar en la Figura 3.44, contiene primeramente la sentencia ss.listen(), ya que al estar trabajando simultáneamente con dos objetos de tipo SoftwareSerial debemos indicar cuando estaremos recibiendo datos del puerto asignado en el primer y segundo objeto declarados al principio del código; después de eso tenemos un ciclo for, y dentro de éste lo primero que hacemos es verificar si han llegado caracteres desde el GPS, de ser así, procedemos a asignar la latitud, longitud, hora y fecha recibidas a las variables creadas al inicio del programa.

```
void Coordenadas() {
  //recibimos todo lo que proviene
  //del puerto asignado en SS
  ss.listen();
  for (int i = 0; i < 22; i++) {
    //Lee los datos recibidos por el serial del GPS
    while (ss.available() > 0)
      gps.encode(ss.read());
    if  (gps.altitude.isUpdated ());
    //Asignamos las coordenadas a
    //las variables
    latitud = (gps.location.lat());
    longitud = (gps.location.lng());
    //Asignamos la hora a las variables
    hora= (gps.time.hour());
    //Convertimos la hora UTC a hora local
    hora-=5;
    minutos= (gps.time.minute());
    //Asignamos la fecha a las variables
    dia= (gps.date.day());
    mes= (gps.date.month());
    anho= (gps.date.year());
    delay (100);

  }
}
```

Figura 3.44. Código sistema 4.

En la Figura 3.45 tenemos la clase AlmacenarSD, la cual, como su nombre lo indica, nos almacenará en la memoria microSD las coordenadas de la ubicación en el momento que sea llamada dicha clase. Primeramente abrimos el archivo para poder escribir en él, en caso de que dicho archivo con el nombre indicado no exista, se creará un archivo con dicho nombre y extensión. Una vez que hayamos abierto o creado el archivo correctamente procedemos a la escritura con ayuda del comando microSD.print. Debemos tomar en cuenta que estamos trabajando con un archivo con extensión .csv, el cual es un archivo de Excel donde las comas indican un cambio de celda y podemos pasar a la siguiente fila con ayuda del comando microSD.println. Para finalizar con esta clase solo debemos cerrar el archivo con el comando microSD.close().

```
void AlmacenarSD(){
  //Abrimos el archivo para escribir en el
  microSD = SD.open("lectur.csv", FILE_WRITE);
  /*Si el archivo se abrio correctamente
   procedemos a la escritura*/
  if (microSD) {
    microSD.print(String(dia) + "/" + String(mes) +
                  "/" + String(anho));
    microSD.print(",");
    microSD.print(String(hora) + ":" + String(minutos));
    microSD.print(",");
    microSD.println(String(latitud,6) + ","
                    + String(longitud,6));
    // cerramos el archivo
    microSD.close();
  }
}
```

Figura 3.45. Código sistema 5.

La última parte del código está dada por la clase loop, en la Figura 3.46 podemos observar esta clase y vemos que inicia aumentando la variable i en uno y luego tenemos la sentencia SIM900.listen() al igual que la clase Coordenadas, pero usando el objeto SIM900, ésto, como ya fue explicado, para indicar que estaremos recibiendo todos los datos provenientes del puerto 2, que es el asignado al objeto SIM900 al inicio del código. Verificamos si hay alguna entrada en el puerto asignado para el objeto SIM900 y de ser así, le asignamos a la variable caract_entrande dicho valor de entrada, los condicionales que están después servirán para verificar que el mensaje que hayamos recibido contenga el código clave, y en caso de ser dicho código el que estemos recibiendo, mandamos a llamar a las clases Coordenadas y enviarSMS, que como se vio anteriormente la primera clase obtendrá la las coordenadas de la ubicación actual y la segunda enviará un mensaje de texto a un número registrado en el sistema. Al final de la clase tenemos un condicional, el cual nos ayudará a que cada dos minutos se guarde automáticamente la fecha, hora y ubicación en un archivo de Excel mediante el módulo SD.

```
void loop(){
  i++;
  //recibimos todo lo que proviene
  //del puerto asignado en SIM900
  SIM900.listen();
  //Verificamos si hay entradas en
 // el puerto del SIM900
  if (SIM900.available() > 0){
  //asignamos el valor de entrada
  //en SIM900 a la variable caract_entrante
    caract_entrante = SIM900.read();
    /*Verificamos si el valor de la variable
    es igual a la letra A */
    if (caract_entrante == 'A'){
     /*Esperamos 10 segundos para que haya
    un nuevo caracter y le asignamos ese
    nuevo caracter a la misma variable*/
      delay(10);
      caract_entrante = SIM900.read();
      /*Ahora verificamos si el valor de la
     variable es igual a la letra B y asi
     sucesivamente para verificar que estamos
     recibiendo el código clave 'AB28'*/
      if (caract_entrante == 'B'){
        delay(10);
        caract_entrante = SIM900.read();
        if (caract_entrante == '2'){
          delay(10);
          caract_entrante = SIM900.read();
          if (caract_entrante == '2'){
           /*En caso de haber recibido el
          código clave procedemos a obtener
          la ubicación y envarla via SMS*/
            Coordenadas();
            enviarSMS();
          }
        }
      }
    }
  }
  /*Este condicional nos sirve para que cada
  dos minutos se guarde en la microSD las
  coordenadas de la ubicación del automovil*/
  if (i>=12000){
    Coordenadas();
    AlmacenarSD();
    i=0;
  }
  //Retardo
  delay(10);
}
```

Figura 3.46. Código sistema 6.

Una vez que tenemos el código cargado en el Arduino, podemos proceder a hacer nuestras pruebas, en la Figura 3.47 tenemos el sistema conectado y funcionando, almacenado cada dos minutos la ubicación y en espera de algún mensaje de texto con el código clave para contestar con un mensaje de la ubicación en tiempo real del dispositivo.

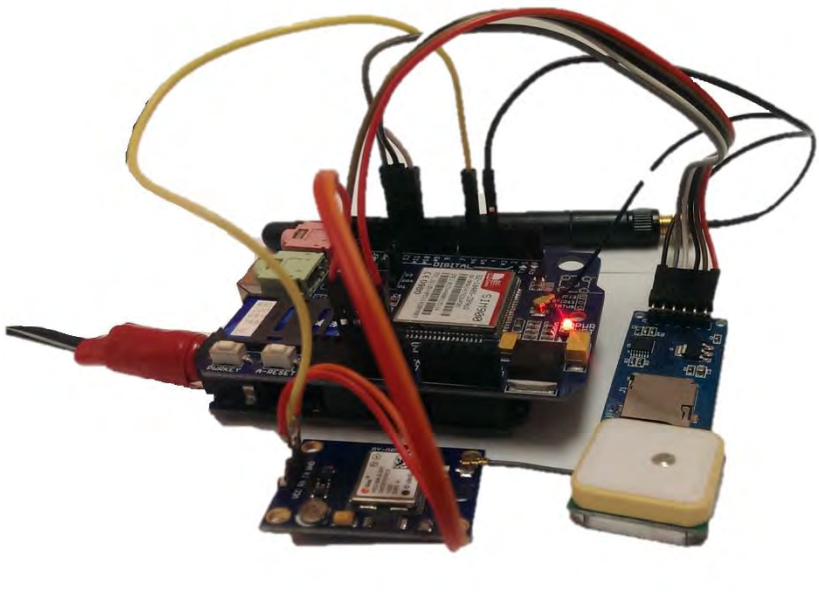


Figura 3.47. Sistema para seguridad vehicular.

Como se mencionó anteriormente el sistema guarda automáticamente cada dos minutos la ubicación donde se encuentre el sistema, la primera prueba consistió en conectar el dispositivo e ir recibiendo las coordenadas y verificar que el archivo de Excel se creó correctamente, en la Figura 3.48 podemos ver que el archivo se encuentra en la memoria microSD, la cual pudimos examinar con un adaptador de microSD a USB, de igual manera observamos que el archivo de

Excel que se obtuvo de dicha prueba, y vemos que el archivo se creó satisfactoriamente.

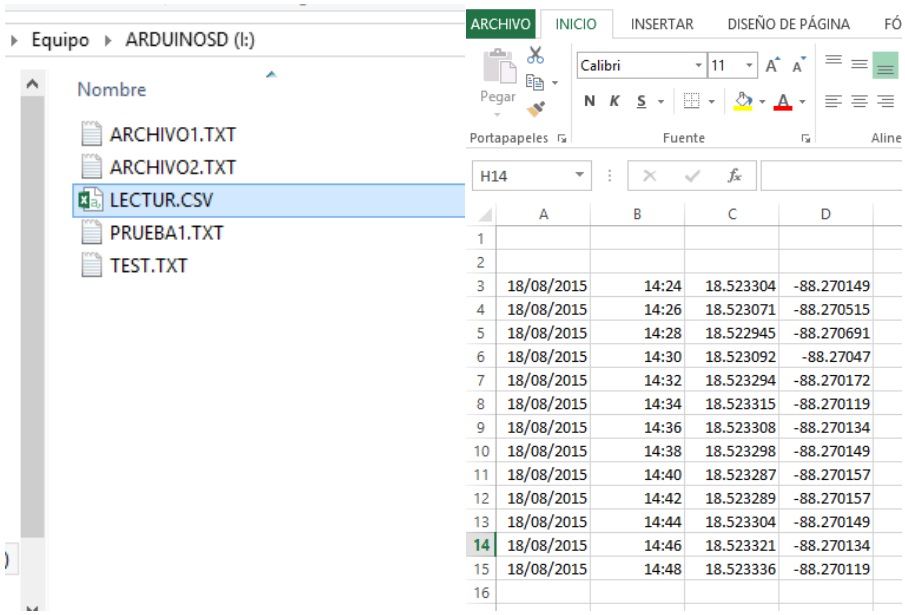


Figura 3.48. Resultados prueba 1.

En la Figura 3.49 tenemos los resultados de la prueba 2, con la cual verificamos que el sistema de mensajes esté funcionando correctamente, vemos como se envía un mensaje de texto con la clave definida “AB28” desde un celular y éste es respondido con un enlace de una consulta de ubicación a Google Maps con la ubicación actual del sistema.

En la Figura 3.50 tenemos el proceso para abrir dicho enlace desde nuestro teléfono inteligente, basta con seleccionar el enlace y se mostrará una ventana en pantalla la cual nos indica que seleccionemos la aplicación con la que queremos abrir el enlace, seleccionamos la aplicación de Google Maps y le damos la opción siempre o solo una vez según sea su preferencia. Para finalizar con esta prueba podemos observar que en la aplicación de Google Maps tenemos la ubicación que en este caso es la Universidad de Quintana Roo ya que en este lugar es donde se han realizado las pruebas del sistema.

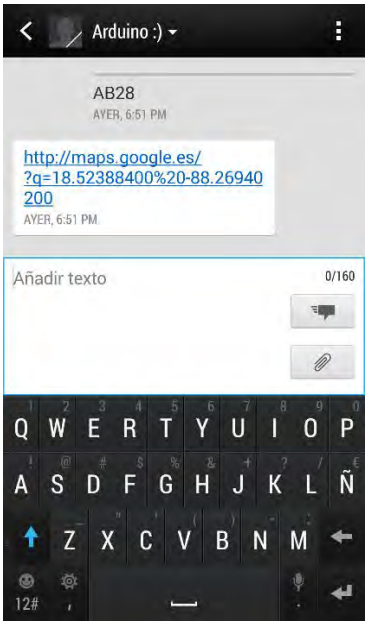


Figura 3.49. Resultados 1, prueba 2.

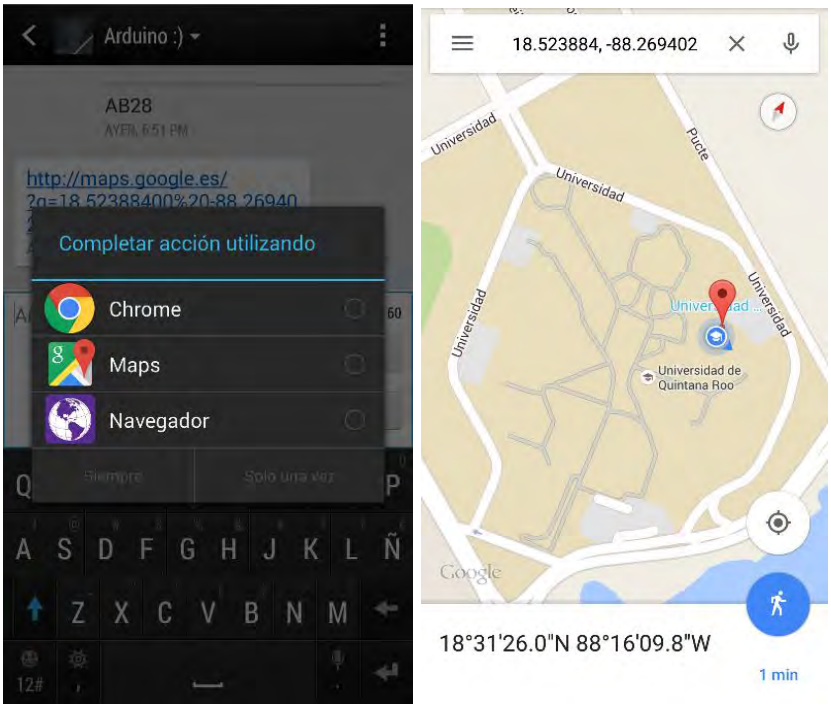


Figura 3.50. Resultados 2, prueba 2.

Capítulo 4. Conclusiones

Las nuevas tecnologías nos dan la oportunidad de cubrir necesidades que van surgiendo con el paso de los año, como se mencionó en este trabajo de tesis, el robo de vehículos es un problema que ha ido aumentando de una manera importante en los últimos años y con el motivo de dar una solución a esa problemática surgió este proyecto de tesis.

Durante el proceso de realización, pudimos estudiar y trabajar con las diferentes tecnologías que se usaron, dándonos una perspectiva mucho más amplia de su funcionamiento. De igual manera se vio la importancia que tienen hoy en día los microcontroladores, los cuales forman parte de nuestra vida cotidiana.

Al concluir el trabajo y realizando las respectivas pruebas, se pudo observar que los resultados han sido satisfactorios y se alcanzaron todos los objetivos del proyecto. Dicho proyecto cuenta con una explicación amplia de cada una de las partes que lo conforman, dando así, una herramienta para poder desarrollar futuros proyectos que tengan relación con el mismo.

La interacción y aplicación de las diferentes tecnologías nos da una visión de nuevos proyectos que pueden ser realizados con las mismas herramientas, y de igual manera nuevas aplicaciones para el proyecto que ya se ha realizado.

Bibliografia

[1] **geolink.** geolink. [En línea] [Citado el: 26 de Mayo de 2015.] https://geolink.io/opentracker.php.

[2] **Whistle.** Whistle. [En línea] [Citado el: 28 de Mayo de 2015.] http://www.whistle.com/.

[3] **Positrace.** Positrace. [En línea] [Citado el: 28 de Mayo de 2015.] http://tracking.positrace.com/.

[4] **Ubitrack.** Ubitrack. [En línea] [Citado el: 30 de Mayo de 2015.] http://www.ubicargps.com/GPS/GPSproductos.aspx.

[5] **Pérez, C.** Robo de autos en aumento. [En línea] [Citado el: 30 de Mayo de 2015.] htps://www.autopistas.com.mx/4076.html.

[6] **Valdés, Fernando.** *Microcontroladores: fundamentos y aplicaciones con PIC.* España : MARCOMBO, 2007. 84-267-14XX.

[7] **INEGI.** Instituto Nacional de Estadística y Geografía. [En línea] [Citado el: 4 de Junio de 2015.] http://www.inegi.org.mx/geo/contenidos/geodesia/gps.aspx?dv=c1.

[8] **GPS.gov.** [En línea] [Citado el: 2 de Junio de 2015.] www.gps.gov/systems/gps/control.

[9] **El-Rabbany, Ahmed.** *Introduction to GPS: The Global Positioning System.* Norwood : ARTECH HOUSE, 2002. 1-58053-183-0.

[10] **Letham, Lawrence.** GPS fácil. Uso del sistema de posicionamiento global. Barcelona : Paidotribo, 2001. 84-8019-591-6.

[11] **Rodriguez Palma, Miguel.** Telecomunicaciones móviles. Barcelona : Marcombo, 1998. 84-267-1149-9.

[12] **Arduino.** Arduino. [En línea] [Citado el: 12 de Junio de 2015.] https://www.arduino.cc/.

[13] **Blum, Jeremy.** Exploring Arduino: Tools and Techniques for Engineering Wizardry. Indiana : John Wiley & Sons, 2013. 978-1-118-54936-0.

[14] **SIMCom.** SIM900 Hardware Design. 2010. SIM900_Hardware Design_V2.00.
[15] **Ublox.** u-blox 6 GPS Modules. Data Sheet. GPS.G6-HW-09005-E.