ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

SISTEMA PROTOTIPO DE DETECCIÓN DE OBJETOS EN EL ÁNGULO CIEGO Y LÍMITES DE VELOCIDAD PARA VEHÍCULOS LIVIANOS.

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE

INGENIERO ELECTRÓNICO Y REDES DE INFORMACIÓN

JUAN CARLOS BOLAÑOS VELASCO

DIRECTOR: ING. CARLOS EGAS MSc.

Quito, marzo 2018

DECLARACIÓN

Yo, JUAN CARLOS BOLAÑOS VELASCO, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la Normatividad Institucional Vigente.

_____

Juan Carlos Bolaños Velasco

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Juan Carlos Bolaños Velasco bajo mi supervisión.

_____

ING. CARLOS EGAS MSc.

DIRECTOR DEL PROYECTO

AGRADECIMIENTO

Agradezco a mi madre Sofía Bolaños, a mi tía Rosa Jiménez, a Maricela Subia quienes han sido mi principal soporte, brindándome su apoyo incondicional y guía durante toda mi formación personal y profesional.

También mi profundo agradecimiento al Ing. Carlos Egas MSc. por sacar adelante este proyecto exitosamente con las directrices y recomendaciones adecuadas.

A mis familiares y amigos por su apoyo.

Juan Carlos Bolaños Velasco

DEDICATORIA

A Dios quien me ha guiado en este proyecto

A mi familia Sofía y Rosa quienes han sido mi principal motivación.

Juan Carlos Bolaños Velasco

CONTENIDO

DECLARACIÓN I

CERTIFICACIÓN II

AGRADECIMIENTO III

DEDICATORIA IV

CONTENIDO V

TABLAS XIII

FIGURAS XIV

CÓDIGO DE LA APLICACIÓN XIX

NOMENCLATURA XXI

RESUMEN XXII

CAPÍTULO 1 – MARCO TEÓRICO 1

1.1 INTRODUCCIÓN 1

1.2 SISTEMAS DE SEGURIDAD DE ENTORNO EN AUTOMÓVILES 1

1.2.1 DETECCIÓN DE FATIGA O FALTA DE ATENCIÓN 2

1.2.2 DETECCIÓN DE CAMBIO DE CARRIL INVOLUNTARIO 3

1.2.3 RECONOCIMIENTO DE SEÑALÉTICA EN EL CAMINO 3

1.2.4 ADVERTENCIA DE SENTIDO CONTRARIO 4

1.2.5 CONTROL DE VELOCIDAD ADAPTATIVO (CON SISTEMA DE DISTANCIA DE SEGURIDAD) 5

1.2.6 RECONOCIMIENTO DE OBJETOS 6

1.2.7 CÁMARAS PARA VISIÓN LATERAL Y POSTERIOR 6

1.2.8 DETECCIÓN DEL ÁNGULO CIEGO 7

1.3 TECNOLOGÍAS UTILIZADAS EN SENSORES PARA VEHÍCULOS LIVIANOS..... 8

1.3.1 PROCESADOR DE IMAGEN Y VÍDEO 8

1.3.2 DETECTORES INFRARROJOS 9

1.3.2.1 Sensores infrarrojos activos 9

1.3.2.2 Sensores infrarrojos pasivos 9

1.3.3 DETECTORES ULTRASÓNICOS 10

1.3.3.1 Principio de funcionamiento de un sensor ultrasónico..... 11

1.3.3.2 Pulso Emitido..... 12

1.3.3.3 Zona Ciega 12

1.3.3.4 Patrón de radiación 13

1.3.3.5 Materiales granulados 13

1.3.4 MICROONDAS / RADAR DE ONDAS MILIMÉTRICAS 14

1.3.5 DETECTORES DE ACELERACIÓN 15

1.3.6 SENSORES LIDAR..... 15

1.4 TECNOLOGÍAS ARDUINO Y ANDROID 16

1.4.1 PLATAFORMA ARDUINO 16

1.4.1.1 Tarjetas de desarrollo Arduino..... 17

1.4.1.2 Placas de expansión (*shields*)..... 21

1.4.1.2.1 Ethernet Shield 21

1.4.1.2.2 Arduino Wifi Shield 22

1.4.1.2.3 Arduino GSM Shield 22

1.4.1.2.4 Arduino Motor Shield 23

1.4.1.2.5 GPS Shield 23

1.4.1.2.6 Xbee Shield 24

1.4.1.2.7 Bluetooth Shield 24

1.4.1.3 Software Arduino 24

1.4.1.3.1 IDE de desarrollo 25

1.4.1.4 Plataforma de Web desarrollo Arduino 25

1.4.2 SISTEMA OPERATIVO ANDROID 26

1.4.2.1 Definición 26

1.4.2.2 Arquitectura del sistema operativo Android 27

1.4.2.2.1 Capa Aplicación 28

1.4.2.2.2 Capa de Inter Proceso de Comunicación 28

1.4.2.2.3 Sistemas de servicios 29

1.4.2.2.4 Capa de Abstracción de Hardware (HAL) 29

1.4.2.2.5 Kernel de Linux 30

1.5 METODOLOGíA DE DESARROLLO KANBAN 30

1.5.1 DEFINICIÓN DE KANBAN 30

1.5.2 PRINCIPIOS FUNDAMENTALES DEL MÉTODO KANBAN 31

1.5.2.1 Visualizar el trabajo y sus fases de ciclo de producción o flujo de trabajo 31

1.5.2.2 Delimitar el Trabajo en Progreso (WIP) 32

1.5.2.3 Optimizar el flujo de trabajo 32

CAPÍTULO 2 - DISEÑO DE LA APLICACIÓN 34

2.1 INTRODUCCIÓN 34

2.2 REQUERIMIENTOS DE SOFTWARE Y HARDWARE 35

2.2.1 SOFTWARE DE DESARROLLO 35

2.2.2 HARDWARE 35

2.3 SENSOR ULTRASÓNICO 36

2.3.1 BENEFICIOS 36

2.3.2 CARACTERÍSTICAS 37

2.3.3 DESCRIPCIÓN DEL ALCANCE DE DETECCIÓN 37

2.3.4 APLICACIONES Y USOS 38

2.3.5 DESCRIPCIÓN DE PINES 38

2.3.6 CONEXIÓN DEL SENSOR CON EL MÓDULO ARDUINO 39

2.3.6.1 Interfaz análoga 39

2.3.6.2 Interfaz por modulación de ancho de pulsos 40

2.3.6.3 Interfaz Serial 41

2.3.7 CONEXIÓN MÚLTIPLE DE SENSORES POR MODULACIÓN ANCHO DE PULSO (PW) 41

2.3.7.1 Libre Conexión 41

2.3.7.2 Operación Simultánea 42

2.3.7.3 Lectura Secuencial 43

2.4 PLACA DE DESARROLLO ARDUINO UNO 44

2.4.1 DESCRIPCIÓN DE PINES 44

2.5 MÓDULO BLUETOOTH HC-06 45

2.5.1 ESTADOS DE FUNCIONAMIENTO DEL MÓDULO HC-06 46

2.5.1.1 Desconectado 46

2.5.1.2 Conectado 46

2.6 DIAGRAMA DE CONEXIÓN ELECTRICA DEL VEHÍCULO 46

2.7 APLICACIÓN ARDUINO 47

2.7.1 CÓDIGO DE LA APLICACIÓN 48

2.7.2 CONFIGURACIÓN de la conexión Android – Arduino 50

2.8 APLICACIÓN ANDROID 51

2.8.1 DIAGRAMAS DE SECUENCIA 51

2.8.1.1 Secuencia de funcionamiento para información de ángulo ciego 51

2.8.1.2 Secuencia de funcionamiento para la información de Velocidad de circulación y exceso de límites de velocidad 53

2.8.2 DIAGRAMAS DE CLASE 54

2.8.3 MÓDULO VELOCIDAD 55

2.8.3.1 getSpeed 57

2.8.3.2 LocationListener 57

2.8.3.3 onLocationChanged 57

2.8.3.4 onProviderDisabled 58

2.8.3.5 onProviderEnabled 58

2.8.3.6 onStatusChanged 58

2.8.3.7 Clase Ubicación.java 60

2.8.4 MÓDULO ÁNGULO CIEGO 62

2.8.4.1 Clase BuscarBT.java 62

2.8.4.2 Clase SeleccionarBT.java 64

2.8.4.3 Clase AnguloCiego.java 66

2.8.5 MÓDULO CONFIGURACIÓN 67

2.8.5.1 Clase ConfiguraciónActivity.java 67

2.8.5.2 Clase ConfiguracionFragment.java 67

2.8.6 CLASE PRINCIPAL.JAVA 68

CAPÍTULO 3 – IMPLEMENTACIÓN DEL PROTOTIPO 73

3.1 IMPLEMENTACIÓN DEL PROTOTIPO 73

3.2 REQUERIMIENTOS PARA LA IMPLEMENTACIÓN DEL PROTOTIPO 73

3.3 HERRAMIENTAS UTILIZADAS PARA EL DESARROLLO 73

3.3.1 ARDUINO IDE 1.8.3 73

3.3.1.1 INSTALACIÓN DEL SOFTWARE ARDUINO IDE 74

3.3.1.2 INSTALACIÓN DE LIBRERIAS EN ARDUINO IDE 76

3.3.2 ANDROID STUDIO 77

3.3.2.1 INSTALACIÓN DE ANDROID STUDIO 77

3.4 IMPLEMENTACIÓN DEL SOFTWARE EN ARDUINO 78

3.4.1 IMPLENTACIÓN DEL CÓDIGO 79

3.5 IMPLEMENTACIÓN DEL SOFTWARE EN ANDROID 80

3.5.1 INTERFAZ DE LA APLICACIÓN EN ANDROID 80

3.5.1.1 VISTA PRINCIPAL: 81

3.5.1.2 VISTA DE CONFIGURACION 81

3.5.1.3 PROTOTIPO ARDUINO 82

3.6 INSTALACIÓN EN EL VEHÍCULO 83

3.6.1 UBICACIÓN DE SENSORES 84

CAPÍTULO 4 - PRUEBAS DEL PROTOTIPO 85

4.1 ACTIVACIÓN DE ALERTA AL SOBREPASAR EL LÍMITE DE VELOCIDAD 85

4.2 ÁNGULO CIEGO 87

4.2.1 PRUEBAS DE FUNCIONAMIENTO DE LOS SENSORES 88

4.2.2 DETECCIÓN DE OBJETOS CON EL VEHÍCULO EN MOVIMIENTO 89

4.3 INSTALACIÓN FINAL EN EL VEHÍCULO 92

CAPITULO 5 - CONCLUSIONES Y RECOMENDACIONES 95

5.1 CONCLUSIONES 95

5.2 RECOMENDACIONES 96

BIBLIOGRAFÍA 98

TABLAS

Tabla 1.1 Clasificación de las placas Arduino 18

Tabla 1.2 Comparación de las placas Arduino 20

Tabla 2.1 Características de la placa Arduino Uno 44

Tabla 2.2 Configuración por defecto del módulo HC-06 46

Tabla 2.3 Conexión de pines entre Arduino y módulo Bluetooth 50

Tabla 3.1 Descripción de pines de conexión entre el sensor ultrasónico y Arduino 78

Tabla 4.1 Valores de distancia y ángulo ciego..... 90

FIGURAS

Figura 1.1 Sensor de detección de fatiga 2

Figura 1.2 Sensor de detección de cambio de carril involuntario 3

Figura 1.3 Señal de límite máximo permitido 4

Figura 1.4 Señal de Advertencia de sentido contrario 4

Figura 1.5 Control de distancia. 5

Figura 1.6 Reconocimiento de objetos 6

Figura 1.7 Cámaras de visión lateral 7

Figura 1.8 Aviso luminoso de Ángulo Ciego 7

Figura 1.9 Procesador de imagen y video 8

Figura 1.10 Sensores infrarrojos activos 9

Figura 1.11 Sensor de infrarrojos pasivo 10

Figura 1.12 Detectores ultrasónicos 10

Figura 1.13 Principio de funcionamiento de un sensor ultrasónico 11

Figura 1.14 Pulso Emitido por un sensor ultrasónico 12

Figura 1.15 Zona ciega en un sensor ultrasónico 12

Figura 1.16 Patrón de radiación de un sensor ultrasónico 13

Figura 1.17 Refracción de pulsos en un material granulado grueso 13

Figura 1.18 Radar de ondas milimétricas 14

Figura 1.19 Sensores Lidar 15

Figura 1.20 Arduino Entry Level 18

Figura 1.21 Arduino Enhance Features 19

Figura 1.22 Arduino Wearables 19

Figura 1.23 Arduino Internet of things 20

Figura 1.24 *Shields* apilados sobre una placa Arduino 21

Figura 1.25 Shield Ethernet 21

Figura 1.26 Shield Wi-Fi 22

Figura 1.27 Shield GSM/GPRS 22

Figura 1.28 Shields para control motores DC 23

Figura 1.29 Shield GPS 23

Figura 1.30 Shield para comunicación con Xbee 24

Figura 1.31 Shield Bluetooth 24

Figura 1.32 Software de desarrollo Arduino para Windows 25

Figura 1.33 Plataforma de desarrollo Web para Arduino 26

Figura 1.34 Interfaz Web de desarrollo para Arduino 26

Figura 1.35 Stack Android 27

Figura 1.36 Arquitectura del Sistema Operativo Android. 28

Figura 1.37 Componentes de la capa de abstracción HAL 29

Figura 1.38 Ilustración Pizarra de Kanban 31

Figura 1.39 Límites WIP 32

Figura 1.40 Gráfico de la optimización del flujo de trabajo 33

Figura 2.1 Descripción gráfica del diseño que se implementará..... 35

Figura 2.2 Sensor Maxbotix MB1000 36

Figura 2.3 Lóbulos de alcance del sensor Maxbotix MB1000 37

Figura 2.4 Descripción de pines del sensor Maxbotix MB1000 LV 38

Figura 2.5 Conexiones mediante interfaz de salida análoga 40

Figura 2.6 Conexiones mediante interfaz de salida PW 40

Figura 2.7 Conexiones mediante interfaz de salida serial 41

Figura 2.8 Interferencia de los sensores en libre conexión. 42

Figura 2.9 Conexión simultánea de los sensores. 42

Figura 2.10 Conexión de funcionamiento secuencial de los sensores. 43

Figura 2.11 Conexión con encadenamiento continuo. 43

Figura 2.12 Descripción de pines – Arduino Uno 45

Figura 2.13 Presentación física y descripción de pines del módulo Bluetooth HC-06. 45

Figura 2.14 Diagrama de conexión eléctrica del vehículo. 47

Figura 2.15 Diagrama de funcionamiento entre Sensores-Arduino-Bluetooth 47

Figura 2.16 Diagrama de conexión eléctrica entre Arduino y el módulo Bluetooth HC-06 50

Figura 2.17 Diagrama funcionamiento ángulo ciego 52

Figura 2.18 Diagrama funcionamiento velocidad y alertas límites de velocidad ... 53

Figura 2.19 Diagrama de clases de la aplicación Android 54

Figura 2.20 Diagrama de Flujo para obtener la ubicación del dispositivo. 56

Figura 3.1 Instalación Arduino IDE - Componentes 74

Figura 3.2 Instalación Arduino IDE - Ubicación de la instalación 74

Figura 3.3 Instalación Arduino IDE - Instalación 75

Figura 3.4 Configuración de la placa Arduino 75

Figura 3.5 Instalación de Librería Arduino 76

Figura 3.6 Instalación de librería Arduino - Ubicación de archivo 76

Figura 3.7 Página web de *Android Developers* para descargar Android Studio ... 77

Figura 3.8 Asistente de instalación de Android Studio 78

Figura 3.9 Conexiones entre sensor y Arduino. 79

Figura 3.10 Vista principal de la aplicación en Android 82

Figura 3.11 Opciones de la vista de configuración 82

Figura 3.12 Prototipo para pruebas de funcionamiento 83

Figura 3.13 Prototipo de pruebas en funcionamiento 83

Figura 3.14 Ubicación de los sensores en el vehículo. 84

Figura 3.15 Ubicación física de los sensores ultrasónicos 84

Figura 4.1 Alerta de límite de velocidad establecido en 50 Km/h 85

Figura 4.2 Alerta de límite de velocidad establecido en 90Km/h. 86

Figura 4.3 Alerta de límite de velocidad establecido en 100 Km/h. 86

Figura 4.4 Interfaz de alerta para ángulo ciego en el lado derecho e izquierdo del vehículo. 87

Figura 4.5 Recopilación de datos del sensor en un vehículo en movimiento..... 88

Figura 4.6 Pruebas de funcionamiento de los sensores para detectar el ángulo ciego. 89

Figura 4.7 Valores de distancia obtenidos y detección de ángulo ciego..... 89

Figura 4.8 Detección de un objeto en el ángulo ciego en una muestra de 10 ms para el sensor 1. 91

Figura 4.9 Detección de un objeto en el ángulo ciego en una muestra de 10 ms para el sensor 2. 91

Figura 4.10 Ubicación de componentes en la placa de circuito impreso 92

Figura 4.11 Placa de circuito impreso..... 93

Figura 4.12 Placa de cobre para conexión de sensores y módulo Bluetooth 93

Figura 4.13 Instalación de sensores y módulo Bluetooth 93

Figura 4.14 Prototipo final para instalar en el vehículo 94

Figura 4.15 Prototipo instalado en el portaequipaje del vehículo..... 94

CÓDIGO DE LA APLICACIÓN

Código 2.1 Definición de pines Arduino 48

Código 2.2 Pin de disparo Arduino 48

Código 2.3 Variables para almacenar distancia..... 48

Código 2.4 Inicialización de enlace serial 48

Código 2.5 Lectura de datos del sensor ultrasónico 49

Código 2.6 Lectura de datos del sensor 49

Código 2.7 Inicio de Lectura del sensor..... 49

Código 2.8 Estructura de la cadena de texto a enviarse por Bluetooth 49

Código 2.9 Declaración de la interfaz para actualizar ubicación de GPS 59

Código 2.10 Declaración de la clase Ubicación.java 60

Código 2.11 Función que calcula la velocidad a la que se mueve el vehículo..... 60

Código 2.12 Declaración de clase Velocidad.java 60

Código 2.13 Conversión a texto del valor de velocidad obtenido mediante GPS 61

Código 2.14 Detección de alertas a partir de la velocidad. 62

Código 2.15 Inicializa los controles de la interfaz de busqueda de bluetooth..... 63

Código 2.16 Permite interactuar con los datos de una lista en una vista..... 63

Código 2.17 Proceso para emparejar el dispositivo Bluetooth con el teléfono 64

Código 2.18 Actualiza la lista de dispositivos emparejados..... 64

Código 2.19 Añade un nuevo dispositivo Bluetooth..... 65

Código 2.20 Verifica el estado de la conexión Bluetooth..... 66

Código 2.21 Declaración de la clase AnguloCiego y Constructor..... 66

Código 2.22 Accesorios para asignar valores a un objeto AnguloCiego 67

Código 2.23 Inicializa el código para modificar la Configuración de la actividad. 67

Código 2.24 Inicialización del Fragment para guardar las configuraciones 68

Código 2.25 Inicialización de las variables para ángulo ciego..... 68

Código 2.26 Comunicación Bluetooth..... 69

Código 2.27 Permisos para acceder a la ubicación del dispositivo 69

Código 2.28 Archivo de sonido de alerta 69

Código 2.29 Actualización de sensores de ángulo ciego..... 70

Código 2.30 Actualización de la velocidad de circulación del vehículo..... 71

Código 2.31 Verifica si se sobrepasó el límite de circulación en ciudad..... 71

Código 2.32 Función que emite la alerta sonora 72

Código 3.1 Programación de la placa Arduino..... 80

NOMENCLATURA

ABS	Antilock Brake System
ADK	Accessory Development Kit
ANT	Agencia Nacional de Tránsito
API	Interfaz de Programación de Aplicaciones
DC	Corriente directa
ESC	Control Electrónico de Estabilidad
GPS	Sistema de Posicionamiento Global
GPRS	Global System for Mobile Communications
HAL	Capa de abstracción de hardware
Hz	Hertz
IR	Sensor infrarrojo
LAN	Local Area Network
LED	Diodo emisor de luz
LIDAR	Light Detection and Ranging
mA	miliamperios
ms	milisegundos
PWM	Pulse Width Modulation
RS-232	Recommended Standard 232
TTL	Transistor-transistor logic
USB	Universal Serial Bus
V	Voltios
Vcc	Voltaje de corriente directa
VIP	Video Image Processor
Web	World Wide Web
WIP	Work In Progress

RESUMEN

Este proyecto propone la implementación de un prototipo de un sistema electrónico de seguridad pasiva de costo moderado en un vehículo liviano que permita alertar al conductor si se exceden límites de velocidad y la detección de objetos en el ángulo ciego con el fin de superar los inconvenientes antes mencionados.

En la fase teórica se presentará un estudio de los sistemas electrónicos de seguridad pasiva existentes en el mercado y relacionados con la detección de objetos en el ángulo ciego de un vehículo liviano y alertas de límites de velocidad.

Se describirán también los sensores electrónicos utilizados actualmente en la seguridad pasiva de vehículos livianos, su funcionamiento y características, para posteriormente determinar cuál de ellos permitirá cumplir los objetivos planteados. Se incluirá además una descripción de las tecnologías Android y Arduino, así como la metodología ágil de desarrollo Kanban, la cual permitirá adoptar un proceso gradual y planificado reduciendo el tiempo de desarrollo del proyecto.

En la fase de diseño se determinarán los sensores adecuados para el funcionamiento del prototipo junto con el diagrama de la conexión eléctrica para que inicie su funcionamiento y se determinarán los requerimientos de software y hardware para la plataforma Arduino y Android.

Se realizarán los diagramas de secuencia la información que proveen los sensores, procesarla de tal manera que permita obtener la información de distancia a la que se encuentra el objeto en el ángulo ciego y el lado del vehículo del que se está recibiendo los datos, sea derecho o izquierdo para enviarla mediante un módulo Bluetooth hacia el dispositivo Android.

En la aplicación de Android para procesar las alertas visuales y acústicas se realizarán los diagramas de secuencia y clase respecto a los datos recibidos desde el Arduino.

También se diseñarán los diagramas de secuencia y clase para la función que permitirá procesar y mostrar en pantalla la velocidad de circulación del vehículo

calculada mediante el uso del GPS del teléfono para analizar si se está excediendo el límite de velocidad máximo permitido el cual está establecido en 50 Km/h (zona urbana), 90 Km/h (vías perimetrales) y 100 Km/h (carreteras) según la ANT, en caso de detectar alguno de estos límites se emitirá en el teléfono un aviso acústico y se mostrará en la interfaz el valor del límite superado, esta alerta visual se mantendrá en pantalla hasta alcanzar el siguiente límite de velocidad y generará un nuevo el aviso acústico.

No se tiene en cuenta la vía por la que circula el vehículo sino la velocidad registrada por el GPS. El usuario dispondrá de un menú de configuración para modificar estos valores en caso de ser necesario, por defecto la aplicación detectará los tres límites indicados.

En la etapa de implementación se realizarán las conexiones necesarias de los sensores con el vehículo previamente diseñadas. Se implementará la aplicación en Arduino para gestionar la información que proveen los sensores y la aplicación en Android que se encargará de procesar la información del Arduino enviada mediante Bluetooth y mostrarla en su interfaz al usuario.

La etapa final corresponde a las pruebas de funcionamiento, donde se verificarán las alertas de velocidad máxima en un vehículo liviano en movimiento y la detección de objetos en el ángulo ciego.

CAPÍTULO 1 – MARCO TEÓRICO

1.1 INTRODUCCIÓN

La conducción de un vehículo es una tarea expuesta a riesgo inminente. El conductor debe siempre estar consciente de su entorno, considerando factores como tráfico, peatones, calzada, condiciones climáticas entre otros.

Esto ha impulsado a los fabricantes a la implementación de elementos de seguridad en el vehículo con el fin de proteger tanto la integridad del conductor como de los peatones según sea el caso.

Ante esta problemática, en la actualidad se han desarrollado varios sistemas electrónicos de seguridad pasiva que permiten alertar al conductor sobre un posible siniestro a fin de salvaguardar la vida del conductor y la de su entorno.

En este capítulo se realizará un análisis de los sistemas electrónicos de seguridad pasiva relacionados con las causas de accidentes de tránsito más comunes. Además, se describirán los sensores electrónicos utilizados en la seguridad pasiva, su funcionamiento y características.

Finalmente, se describirá la metodología ágil de desarrollo Kanban, la cual permitirá adoptar un proceso gradual y planificado reduciendo el tiempo de desarrollo del prototipo.

1.2 SISTEMAS DE SEGURIDAD DE ENTORNO EN AUTOMÓVILES

Sin duda la manera más eficaz de protección de accidentes es la prevención. Durante varios años las empresas automotrices se han dedicado al desarrollo de sistemas de protección activa y pasiva que han contribuido significativamente a la reducción de accidentes.

En la actualidad, a más de los sistemas de seguridad implementados como son, el Sistema ABS y ESC existen los denominados detectores de entorno que brindan al

conductor una experiencia de manejo más confiable disminuyendo el riesgo causado por nuestro propio vehículo o por vehículos a nuestro entorno [1].

Se describirán algunos de los sistemas principales con los que cuenta un vehículo de media alta gama.

1.2.1 DETECCIÓN DE FATIGA O FALTA DE ATENCIÓN

Figura 1.1 Sensor de detección de fatiga [1]

Se trata de un sistema electrónico provisto de un sensor en el volante que capta el número de movimientos o correcciones realizadas por el conductor en un intervalo de tiempo de un minuto como se indica en la figura 1.1.

Al conducir, es necesario realizar cambios en el volante para ajustar el vehículo al camino por el que circulamos por tanto al no detectar este cambio o al detectar un número menor de cambios al que se considera “normal” se activa el sensor de fatiga.

El tipo de aviso que puede emitir este sensor varía desde una alarma sonora, o inclusive la vibración del volante [1].

1.2.2 DETECCIÓN DE CAMBIO DE CARRIL INVOLUNTARIO

Este sistema está constituido por un microprocesador que sensa permanentemente la trayectoria seguida por el vehículo.

Figura 1.2 Sensor de detección de cambio de carril involuntario [1]

En la figura 1.2 se indica el funcionamiento que consiste en la detección de las líneas marcadas en la carretera ya sean continuas o discontinuas guiando al vehículo por el camino establecido.

La aplicación de este sistema puede ser por medio de sensores ubicados uno a cada lado del vehículo, en el guardacoches o muy cerca del suelo o a su vez puede tratarse de una cámara colocada en el parabrisas al nivel del espejo retrovisor.

Si nuestro propósito es el cambio de carril y olvidamos activar la luz intermitente, el sistema interpreta que el cambio es involuntario emitiendo la vibración en el asiento o volante. Existen alertas que pueden ser acústicas o avisos luminosos en el panel de instrumentos [1].

1.2.3 RECONOCIMIENTO DE SEÑALÉTICA EN EL CAMINO

El sistema en mención reconoce las señales circulares presentes en la carretera y las muestra en la pantalla del panel de instrumentos.

La información se registra a través de una cámara de gran resolución colocada en la parte delantera del vehículo centrado en la parte superior del retrovisor, la cual indica los valores de velocidad máxima permitida y de prohibido adelantar.

Figura 1.3 Señal de límite máximo permitido [1].

Los valores registrados en el panel se actualizan conforme a lo establecido en cada tramo según cambien las señales como se muestra en la figura 1.3 [1].

1.2.4 ADVERTENCIA DE SENTIDO CONTRARIO

Figura 1.4 Señal de Advertencia de sentido contrario [1].

El sistema, con ayuda del GPS incorporado en el vehículo determina el sentido de la vía por la cual estamos circulando emitiendo una alarma sonora y visual en la

pantalla del navegador. En la figura 1.4 se puede ver en el panel del vehículo la alerta detectada por circular en sentido contrario.

Aun cuando parezca poco usual, existen estadísticas de que en Alemania por ejemplo se dan aproximadamente 1800 casos al año en el que el conductor ingresa en contra vía [1].

1.2.5 CONTROL DE VELOCIDAD ADAPTATIVO (CON SISTEMA DE DISTANCIA DE SEGURIDAD)

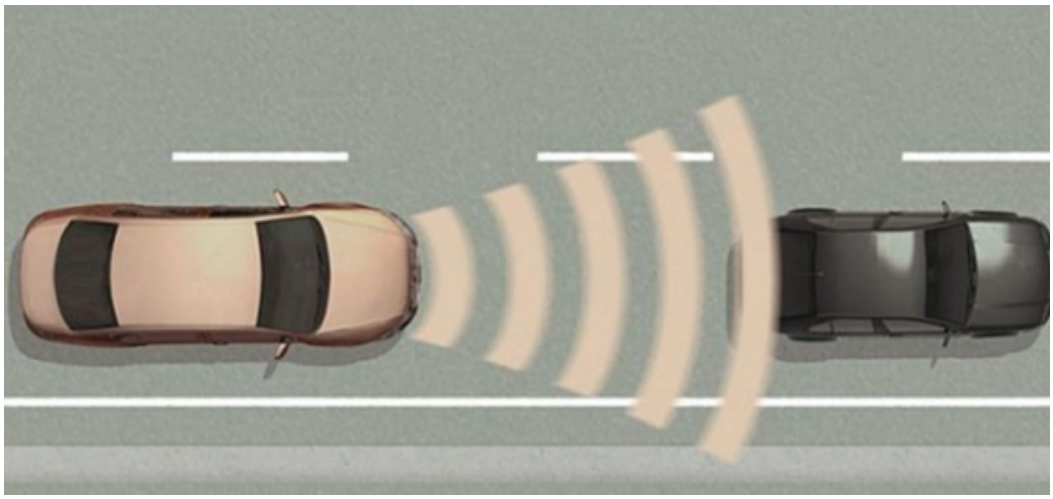


Figura 1.5 Control de distancia [1].

El sistema de control de velocidad adaptativo permite circular a una velocidad constante sin exceder los límites de velocidad. Con la ayuda de un pequeño procesador que ubica en el frente del vehículo se puede controlar el acelerador siendo este desactivado en caso de necesitar frenar o acelerar como indica la figura 1.5.

Además, un radar colocado en la parte delantera (guardachoques) mide la distancia existente entre vehículos y en función de la velocidad a la que vayamos, el microprocesador permite determinar la distancia de seguridad correcta con el vehículo precedente evitando posibles colisiones con vehículos cercanos.

En caso de no tener suficiente distanciamiento, el sistema desacelera el vehículo para aumentar la distancia a la necesaria.

1.2.6 RECONOCIMIENTO DE OBJETOS

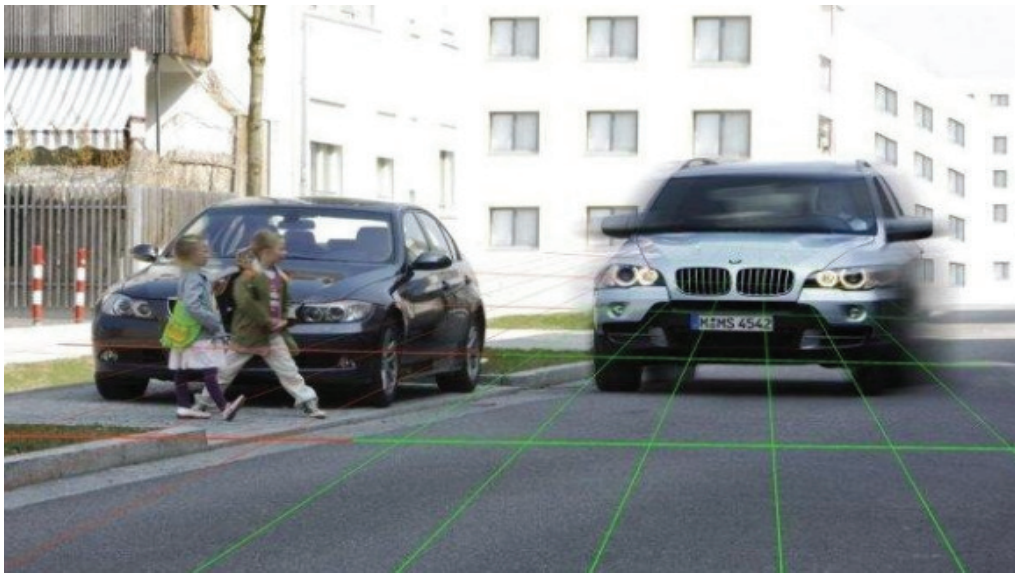


Figura 1.6 Reconocimiento de objetos [1].

Este sistema es uno de los más utilizados y de mayor acogida ya que, a más de distinguir entre peatones, ciclistas, vehículos y objetos en general permite medir la distancia entre ellos y predecir su trayectoria determinando si existiría una posible colisión.

El sistema alertará al conductor y si el riesgo es alto inclusive se accionarán automáticamente los frenos de los vehículos [1].

En la figura 1.6 se puede observar el campo de visión del vehículo que rastrea el entorno reconociendo los diferentes objetos que se presentan en el camino.

1.2.7 CÁMARAS PARA VISIÓN LATERAL Y POSTERIOR

El sistema dispone de dos cámaras, una cámara de visión para marcha atrás y otra cámara de visión lateral.

La cámara de visión marcha atrás es colocada ya sea en el portón del maletero, sobre la matrícula o en lugares imperceptibles permitiendo determinar si hay objetos o niños que a simple vista no se pueden observar. Así mismo indica al conductor la posición adecuada a la cual estacionarse.

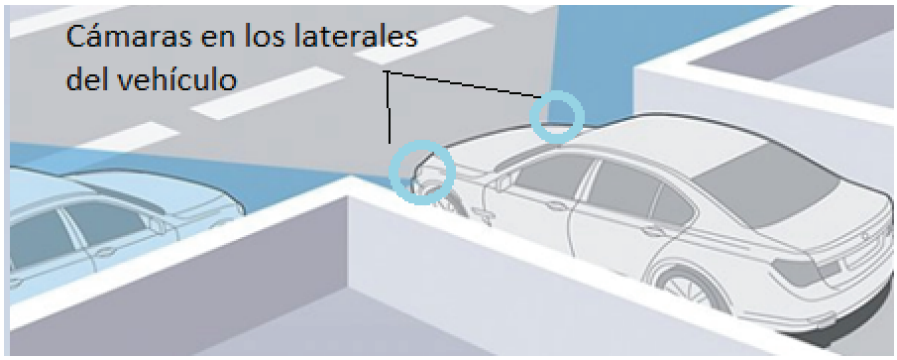


Figura 1.7 Cámaras de visión lateral [1]

Por otro lado, existen las cámaras de visión lateral colocadas en las esquinas del guardachoque delantero como indica la figura 1.7 permitiendo al conductor determinar si es seguro continuar el trayecto en caso de que no se pueda observar a simple vista si un vehículo se aproxima [1].

1.2.8 DETECCIÓN DEL ÁNGULO CIEGO

Figura 1.8 Aviso luminoso de Ángulo Ciego [1].

El ángulo ciego corresponde al área a la cual no se tiene campo visual ya que el espejo no es capaz de cubrir esa zona.

Existe un gran número de accidentes tanto entre vehículos livianos como motocicletas, bicicletas o peatones que eventualmente se aproximan al punto ciego sin ser detectados desde la posición del conductor. A través de los años, la industria automovilística ha intentado resolver el problema modificando la curvatura de los espejos retrovisores logrando aumentar el área de visión.

Actualmente, con la utilización de la tecnología electrónica se ha conseguido dar solución a este inconveniente.

El uso de radares ubicados en las esquinas del parachoques trasero o lateral alertan al conductor de la presencia de un objeto (vehículo, peatón, bicicleta, etc.) La figura 1.8 indica por medio de una señal visual localizada junto al espejo retrovisor permitiéndole el cambio de carril o adelantamiento [1].

1.3 TECNOLOGÍAS UTILIZADAS EN SENSORES PARA VEHÍCULOS LIVIANOS

Existe un amplio rango de tecnologías de sensores disponibles para detectar variables físicas en vehículos, se describen las más comunes:

1.3.1 PROCESADOR DE IMAGEN Y VÍDEO

Figura 1.9 Procesador de imagen y video [2].

Un procesador de imagen y video (VIP) es una combinación de hardware y software el cual extrae información de los datos que provee un sensor de imagen. El sensor de imagen puede ser una cámara de televisión convencional o una cámara infrarroja. En la figura 1.9 indica como un VIP puede detectar velocidad, presencia y reconocimiento de objetos. Una de las ventajas de este tipo de sensor es que muestra información en tiempo real acerca de su entorno, la cual puede ser

procesada mediante algoritmos y utilizada para obtener información de tráfico, velocidad de circulación entre otras aplicaciones. La desventaja del uso de esta tecnología es que la detección de imagen se ve afectada por falta de luz ambiental, clima y reflejo de la superficie del camino [2].

1.3.2 DETECTORES INFRARROJOS

Se definen dos tipos de sensores infrarrojos (IR), activos y pasivos.

1.3.2.1 Sensores infrarrojos activos

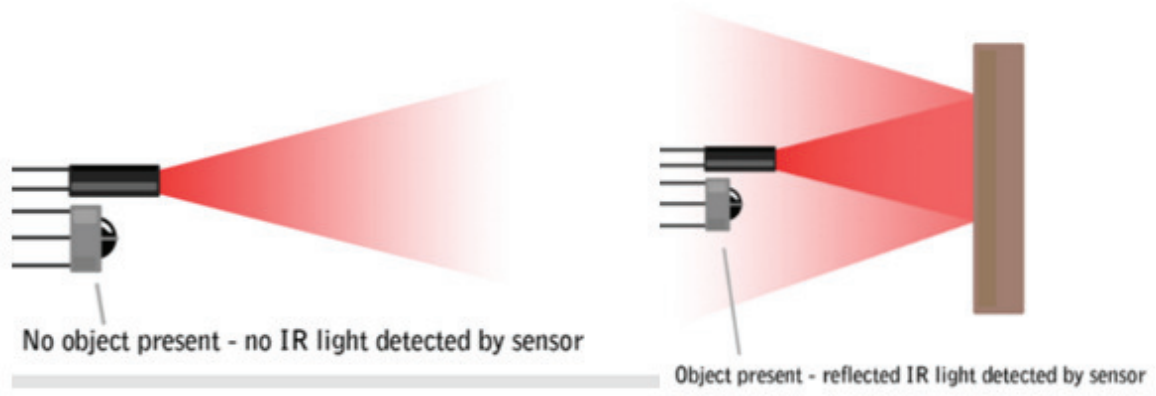


Figura 1.10 Sensores infrarrojos activos [3].

Los sensores infrarrojos activos funcionan enviando pulsos de luz desde un diodo emisor de luz (LED) o diodo láser. Estos pulsos reflejan el haz de luz en un objetivo el cual detecta la señal consistente a un pixel o un grupo de pixeles como indica la figura 1.10. La señal es procesada mediante algoritmos de procesamiento de señales para extraer la información deseada. Los detectores IR proveen datos sobre conteo, presencia, velocidad, ocupación operando en la noche y en el día. Un diodo láser también se puede utilizar para la clasificación de vehículos ya que permite obtener datos sobre perfiles y formas [2].

1.3.2.2 Sensores infrarrojos pasivos

Un sensor infrarrojo pasivo detecta la energía emitida por los objetos que se encuentren dentro del campo de visión y utiliza algoritmos de procesamiento de

señales para obtener la información deseada. Este sensor no emite ningún tipo de energía y puede detectar presencia y cuentas de objetos [2]. En la figura 1.11 se puede observar el funcionamiento de un sensor infrarrojo pasivo, cuando cruza una persona frente al lente de detección.

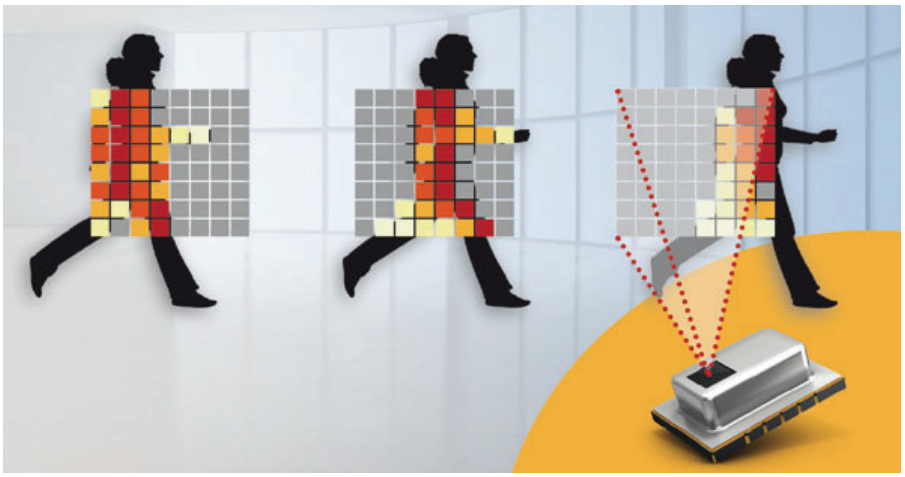


Figura 1.11 Sensor de infrarrojos pasivo [4]

Las ventajas de los sensores infrarrojos son que pueden trabajar tanto en el día como en la noche, la desventaja es que se ven afectados por factores climáticos como niebla, lluvia y polvo.

1.3.3 DETECTORES ULTRASÓNICOS

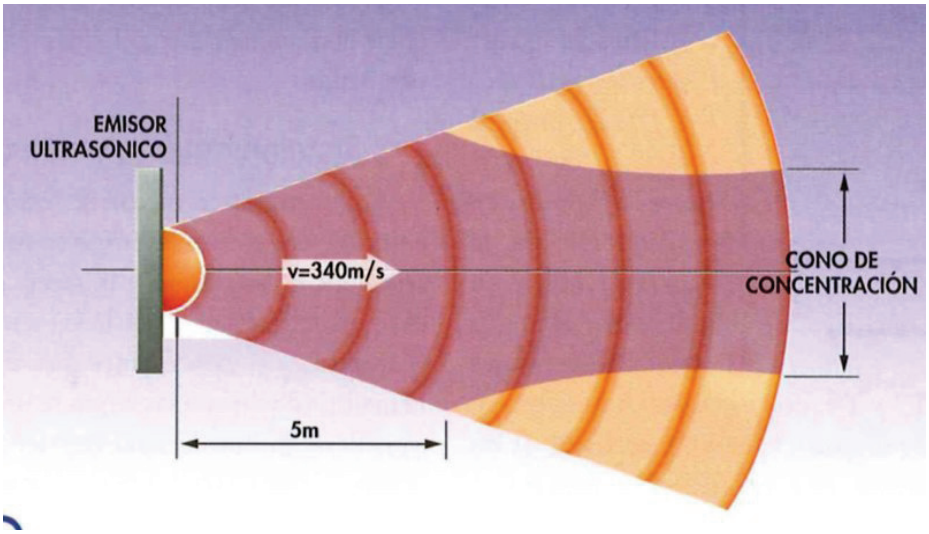


Figura 1.12 Detectores ultrasónicos [5]

Este tipo de sensores operan transmitiendo sonidos ultrasónicos como indica la figura 1.12 y se mide la energía que es reflejada por un objeto dentro del rango de alcance del sensor. Este tipo de medidas son procesadas para obtener datos sobre presencia de vehículos, velocidad y bloqueo por objetos.

La ventaja de un sensor ultrasónico es que opera en cualquier situación climática, la desventaja es que necesita ser montado de manera perpendicular hacia el objetivo, presentado dificultad en la detección de objetos diagonales a su campo de acción. Alguna de las desventajas indicadas anteriormente pueden ser compensadas a través de técnicas de procesamientos de datos [2].

1.3.3.1 Principio de funcionamiento de un sensor ultrasónico

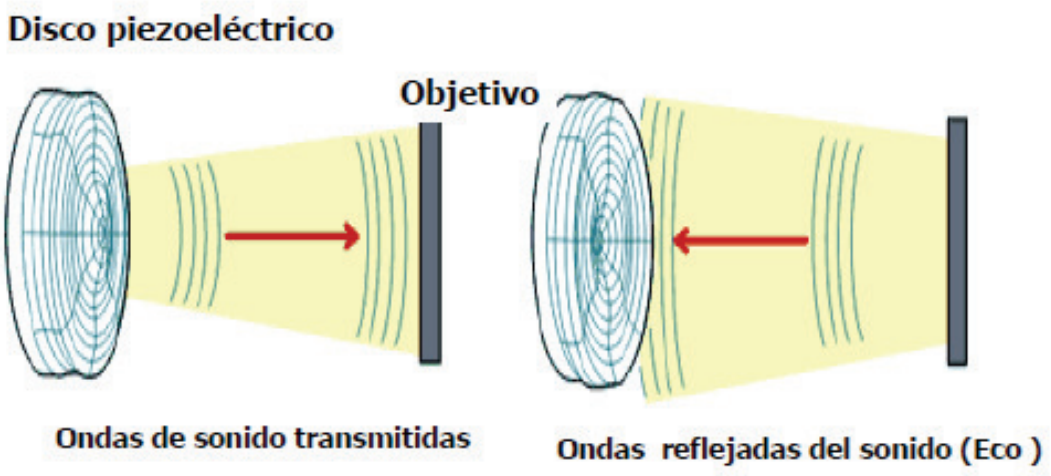


Figura 1.13 Principio de funcionamiento de un sensor ultrasónico [2]

El sensor mediante una placa piezoeléctrica emite ondas de sonido de alta frecuencia, las ondas son reflejadas al pegar sobre un objeto generan eco. La duración del pulso que se refleja es analizada por el transductor, se observa el funcionamiento del sensor en la figura 1.13.

De acuerdo con el rango de operación previamente establecido y el objeto se encuentra dentro de ese rango, la salida del sensor cambia de estado. Al salir el objeto del rango establecido, la salida del sensor vuelve a su estado por defecto [2].

1.3.3.2 Pulso Emitido

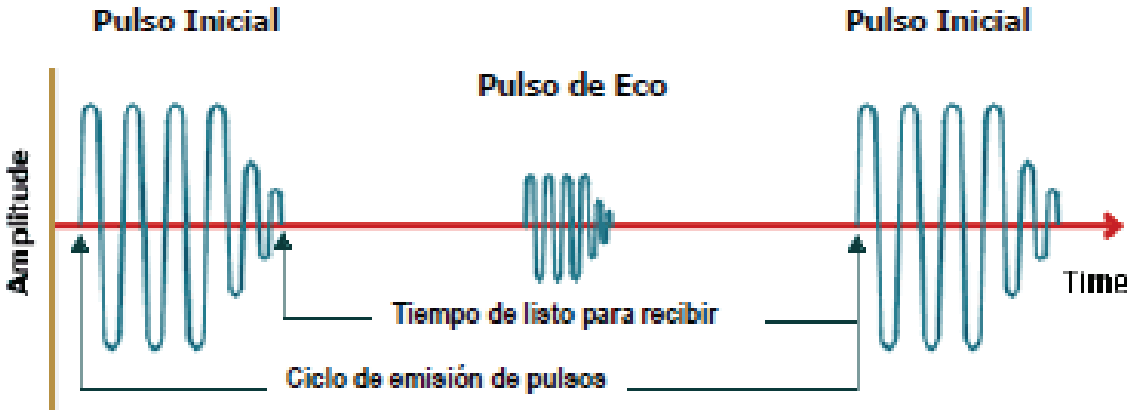


Figura 1.14 Pulso Emitido por un sensor ultrasónico [6].

En la figura 1.14 se observa que el pulso emitido es una pequeña “explosión” corto de energía ultrasónica de gran amplitud. El pulso de eco es normalmente de amplitud más baja. El intervalo de tiempo entre la señal transmitida y el eco generado es directamente proporcional a la distancia existente entre el sensor y el objeto.

1.3.3.3 Zona Ciega

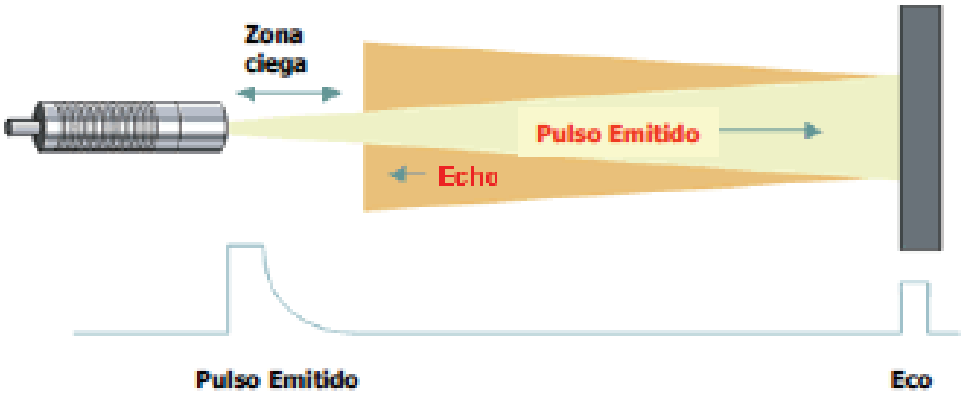


Figura 1.15 Zona ciega en un sensor ultrasónico [6].

En la figura 1.15 indica el rango de detección y la zona ciega, con las siguientes características:

- Directamente frente al sensor hay una zona ciega.
- Dependiendo del sensor, la zona ciega es de 6 a 80 cm del frente del sensor.
- Si existe un objeto colocado en la zona ciega produce una salida inestable [6].

1.3.3.4 Patrón de radiación

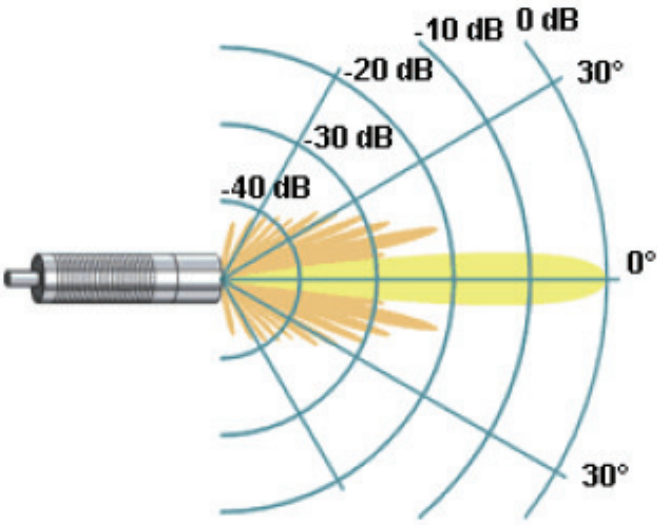


Figura 1.16 Patrón de radiación de un sensor ultrasónico [6]

El patrón de radiación de un sensor ultrasónico está constituido de un cono principal y varios conos como vecinos. El valor del ángulo del cono principal es de 5° aproximadamente como se indica en la figura 1.16.

1.3.3.5 Materiales granulados

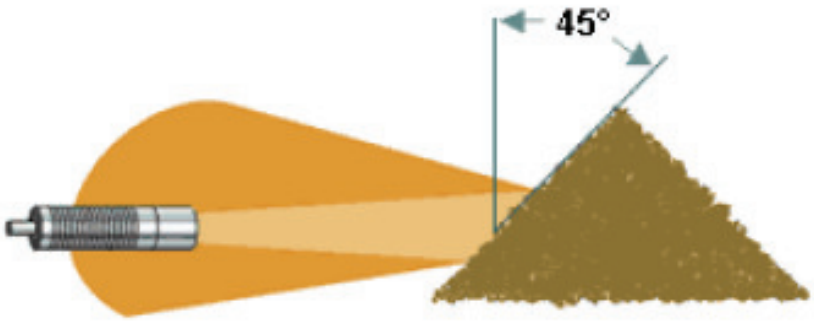


Figura 1.17 Refracción de pulsos en un material granulado grueso [6]

Los materiales granulados gruesos como se observa en la figura 1.17 tal como la arena pueden tener una desviación angular de hasta 45°. Esto se debe a que el sonido es reflejado sobre un gran ángulo [6].

1.3.4 MICROONDAS / RADAR DE ONDAS MILIMÉTRICAS

Figura 1.18 Radar de ondas milimétricas [7]

Los detectores microondas han sido usados extensivamente en Europa. La operación funciona mediante medición de energía reflejada de los vehículos dentro de un campo de visión.

Mediante el procesamiento de la información recibida en la energía reflejada, el detector mide la velocidad, ocupación y presencia.

En la figura 1.18 se puede observar como las microondas detectan una persona en la cercanía del vehículo,

Algunas de las ventajas de los detectores de microonda es que se trata de una tecnología que ha sido desarrollada con bastante anterioridad debido a las aplicaciones militares que consiguieron detectar la velocidad directamente y un solo detector puede cubrir múltiples carriles utilizando las técnicas de procesamiento de señal apropiadas.

Así mismo, las desventadas de este sistema es la detección de velocidad de vehículos no deseado debido a la recepción de radiación lateral y la detección falsa debido a la trayectoria múltiple.

Estos inconvenientes se han visto resueltos mediante la colocación adecuada de los detectores y algoritmos de procesamiento de señales y diseño de antena [7].

1.3.5 DETECTORES DE ACELERACIÓN

Este sistema se ha desarrollado con el fin de evitar colisiones al girar a la izquierda debido a la velocidad de ingreso a la curva.

Utilizando la información del efecto *doppler* se puede determinar la velocidad de alcance de un vehículo mediante la medición de tres detectores en forma lineal que proporciona una aproximación de la aceleración del vehículo a partir de la cual se puede determinar si se activa o no el aviso de giro a la izquierda.

1.3.6 SENSORES LIDAR

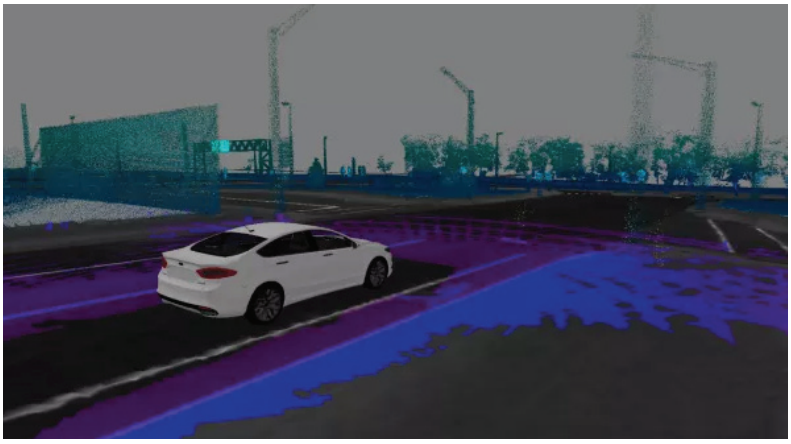


Figura 1.19 Sensores Lidar [8].

LIDAR es un acrónimo *de Light Detection and Ranging* o *Laser Imaging Detection and Ranging* y constituye una de las piezas más importantes en los vehículos ya que permite medir la distancia entre el emisor laser y el objeto empleando un haz de laser pulsado. La distancia se determina midiendo el tiempo de retraso entre la emisión del pulso y la detección del reflejo de la señal.

Para el caso de vehículos este sensor se encuentra ubicado en la parte superior permitiendo una visión de 360° al entorno como indica la figura 1.19.

Cada pixel del Lidar registra el tiempo que tarda el láser en recorrer la escena y rebotar en la cámara recogiendo muestras de la corriente entrante enviando al computador imágenes en 3D del exterior, así como la ubicación e intensidad de cada reflexión.

Este sistema de cálculo es directo gracias al implemento de la física con la velocidad de la luz. En la actualidad cada pixel puede recoger hasta 44 muestras por cada impulso [8].

1.4 TECNOLOGÍAS ARDUINO Y ANDROID

Una ventaja de las tecnologías que se basan en código abierto es que permite a la comunidad de desarrolladores trabajar con diferentes lenguajes y plataformas de tal manera que sean compatibles y puedan interactuar entre sí para compartir información.

En Arduino, mediante el uso de módulos de comunicación basados en estándares permite establecer conexión con un dispositivo Android permitiendo la interconexión con dispositivos electrónicos como sensores, ampliando las funciones y usos del dispositivo.

Se describirá las plataformas Arduino y Android.

1.4.1 PLATAFORMA ARDUINO

Arduino nació como un proyecto para estudiantes en el Instituto Interactivo de Diseño IVREA en Italia, es una plataforma electrónica *open source* basada en software y hardware abierto de fácil implementación. Una tarjeta Arduino dispone de entradas de datos las cuales pueden recibir la información de un sensor de luz, un pulsador, en mensaje desde una plataforma web como twitter, la cual permita la activación de un motor, encender un LED, publicar mensajes *online.*

El software utilizado permite programar mediante un conjunto de instrucciones que se envían al microcontrolador de la tarjeta, para lograr esto se hace uso del lenguaje de programación Arduino (basado en Wiring) y el software de desarrollo Arduino, basado en Processing.

Los componentes de la tarjeta Arduino consta básicamente de un microcontrolador, puertos de entrada/salida, los cuales se pueden conectar a tarjetas de expansión que se conocen como *shields* permitiendo nuevas funciones a la tarjeta Arduino. La alimentación y comunicación se realiza mediante un puerto de conexión USB, usando comunicación serial y un convertidor de niveles RS-232 a serial TTL [9].

Las tarjetas Arduino buscan desarrollar proyectos interactivos con el entorno mediante el uso de sensores y activadores, presentando un bajo coste y facilidad de uso. La empresa Google colaboró con el desarrollo del Kit Android ADK, una tarjeta Arduino capaz de establecer comunicación con teléfonos móviles inteligentes que funcionan bajo el sistema operativo Android, permitiendo controlar desde el teléfono motores, luces y otros dispositivos, así como también recibir información de sensores los cuales miden variables físicas del entorno para luego ser procesadas en el teléfono.

Actualmente Arduino maneja la marca Genuino para su distribución mundial y permitiendo que nuevas empresas se unan al desarrollo y continuidad del producto [10].

1.4.1.1 Tarjetas de desarrollo Arduino

Los modelos de Arduino se categorizan en placas de desarrollo, placas de expansión (*shields*), *kits* y accesorios.

Como indica la Tabla 1.1 las placas Arduino se dividen en:

- *Entry Level* (Productos de fácil uso para comenzar proyectos).
- *Enhanced Features* (Funcionalidades avanzadas y rápido rendimiento).

- *Internet of things* (Permiten conectividad a la web).
- *Wearables* (Permite incorporar tecnología en la vestimenta) [11].

Tabla 1.1 Clasificación de las placas Arduino [12]

ENTRY LEVEL	GENUINO UNO, GENUINO 101, GENUINO MICRO, GENUINO STARTER KIT, GENUINO MKR2UNO ADAPTER
ENHANCED FEATURES	GENUINO MEGA, GENUINO ZERO, PROTO SHIELD, MKR PROTO SHIELD, MKR PROTO LARGE SHIELD
INTERNET OF THINGS	GENUINO MKR1000, MKR1000 BUNDLE, GENUINO YÚN SHIELD

En las figuras 1.20, 1.21, 1.22, 1.23 se muestra algunos ejemplos de placas desarrolladas para la plataforma Arduino para uso básico (*Entry Level*), uso avanzado (*Enhanced Features*) y uso domótico conectado a la red (*Internet of Things*)

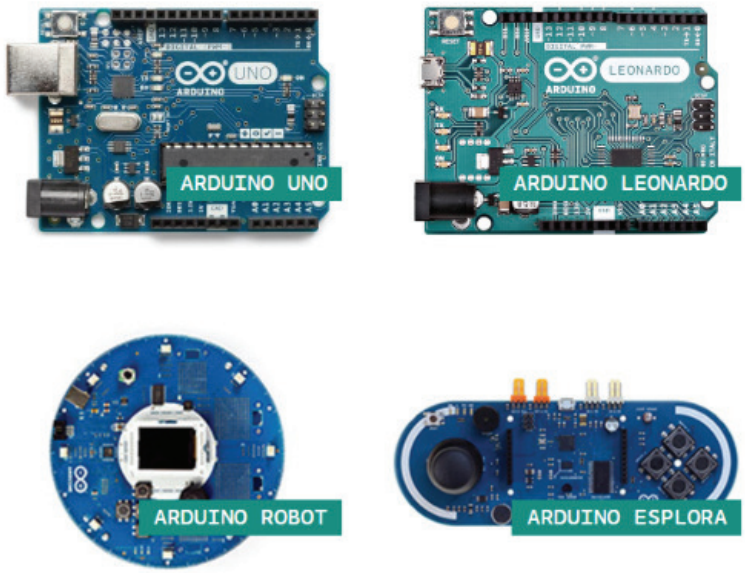


Figura 1.20 Arduino Entry Level [13]

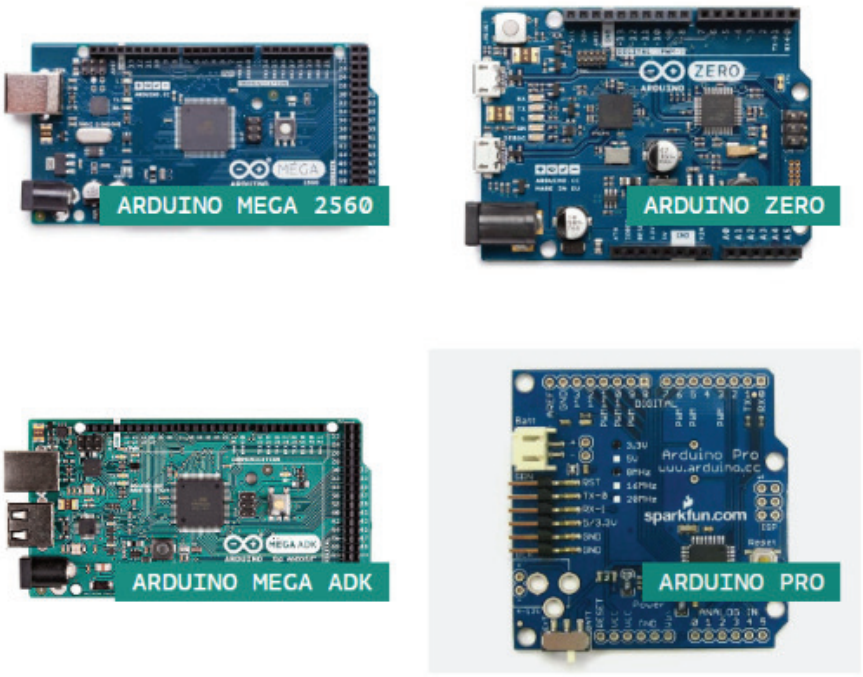


Figura 1.21 Arduino Enhance Features [13]

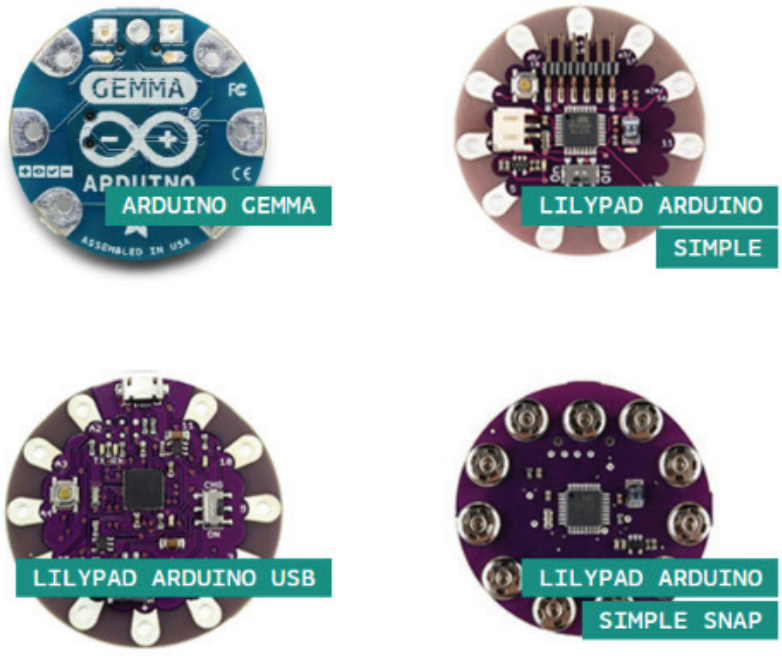


Figura 1.22 Arduino Wearables [13]

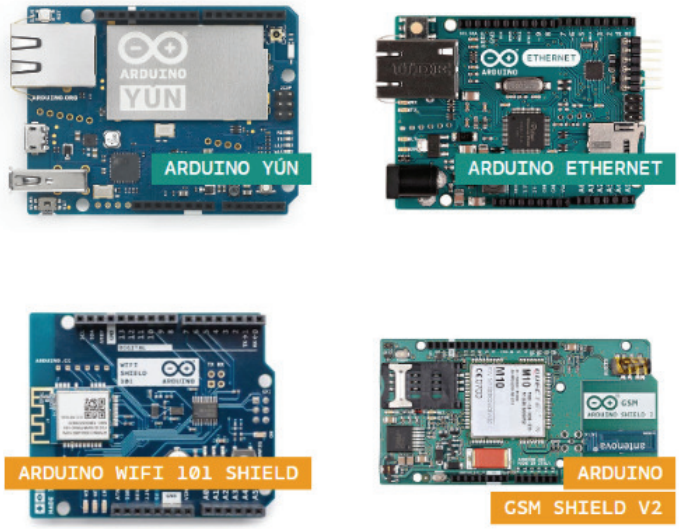


Figura 1.23 Arduino Internet of things [13]

Tabla 1.2 Comparación de las placas Arduino [14]

Name	Processor	Operating/Input Voltage	CPU Speed	Analog In/Out	Digital IO/PWM	EEPROM [kB]	SRAM [kB]	Flash [kB]	USB	UART
101	Intel® Curie	3.3 V/ 7-12V	32MHz	6/0	14/4	-	24	196	Regular	-
Gemma	ATtiny85	3.3 V / 4-16 V	8 MHz	1/0	3/2	0.5	0.5	8	Micro	0
LilyPad	ATmega168V ATmega328P	2.7-5.5 V / 2.7-5.5 V	8MHz	6/0	14/6	0.512	1	16	-	-
LilyPad SimpleSnap	ATmega328P	2.7-5.5 V / 2.7-5.5 V	8 MHz	4/0	9/4	1	2	32	-	-
LilyPad USB	ATmega32U4	3.3 V / 3.8-5 V	8 MHz	4/0	9/4	1	2.5	32	Micro	-
Mega 2560	ATmega2560	5 V / 7-12 V	16 MHz	16/0	54/15	4	8	256	Regular	4
Micro	ATmega32U4	5 V / 7-12 V	16 MHz	12/0	20/7	1	2.5	32	Micro	1
MKR1000	SAMD21 Cortex-M0+	3.3 V/ 5V	48MHz	7/1	8/4	-	32	256	Micro	1
Pro	ATmega168 ATmega328P	3.3 V / 3.35-12 V 5 V / 5-12 V	8 MHz 16 MHz	6/0	14/6	0.512 1	1 2	16 32	-	1
Pro Mini	ATmega328P	3.3 V / 3.35-12 V 5 V / 5-12 V	8 MHz 16 MHz	6/0	14/6	1	1	32	-	1
Uno	ATmega328P	5 V / 7-12 V	16 MHz	6/0	14/6	1	2	32	Regular	1
Zero	ATSAMD21G18	3.3 V / 7-12 V	48 MHz	6/1	14/10	-	32	256	2 Micro	2

En la tabla 1.3 se muestra una comparación de las características técnicas de hardware de las placas Arduino.

1.4.1.2 Placas de expansión (*shields*)

Los shields son placas que se incorporan a manera de módulos a una placa Arduino como indica la figura 1.24 permitiendo añadir funcionalidades extras, esta se apila sobre el Arduino [15].

Figura 1.24 *Shields* apilados sobre una placa Arduino [15]

Entre los *shields* más comunes se tiene:

1.4.1.2.1 Ethernet Shield

Incorpora al Arduino un puerto LAN RJ-45 para conectar la placa a una red de datos, figura 1.25.

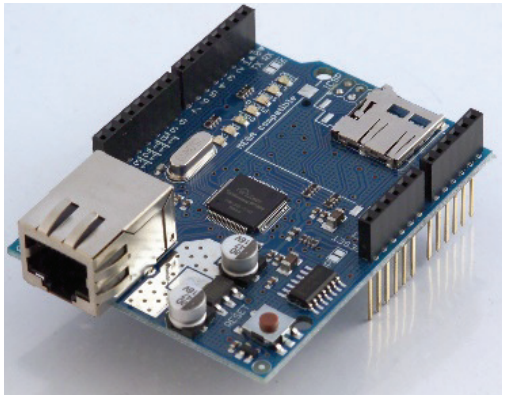


Figura 1.25 Shield Ethernet [16]

1.4.1.2.2 Arduino Wifi Shield

Permite conectar al Arduino a internet o una red inalámbrica a través de una conexión Wi-Fi. La figura 1.26 muestra una placa con conexión Wi-Fi y tarjeta SDCard.

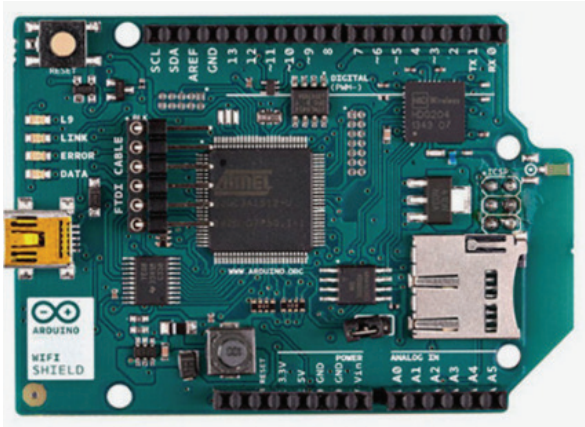


Figura 1.26 Shield Wi-Fi [17]

1.4.1.2.3 Arduino GSM Shield

Conecta Arduino a red de telefonía móvil a través de GSM/GPRS, como indica la figura 1.27 este módulo incluye la ranura para tarjeta SIM necesaria para su funcionamiento.

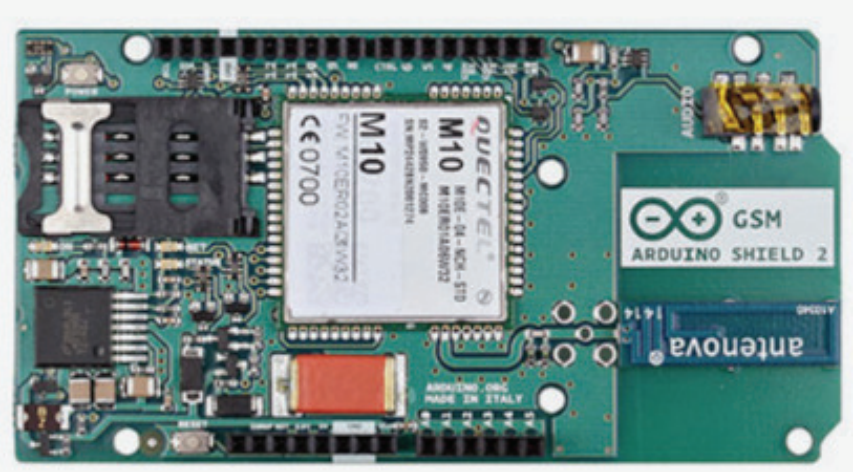


Figura 1.27 Shield GSM/GPRS [18]

1.4.1.2.4 Arduino Motor Shield

Permite controlar dos motores DC, su dirección y velocidad. En la figura 1.28 se observa una placa compatible con Arduino y sus respectivas conexiones para un motor DC.

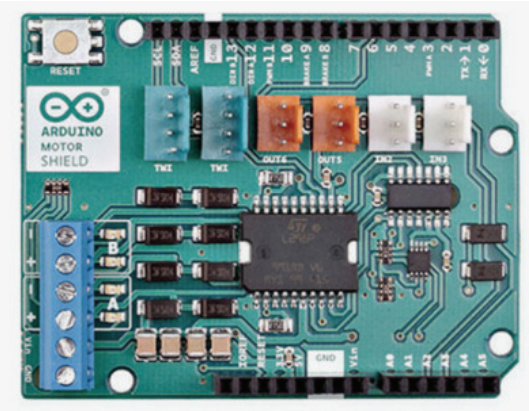


Figura 1.28 Shields para control motores DC [19]

1.4.1.2.5 GPS Shield

Un *shield* que permite almacenar la ubicación geográfica y ser procesada por la placa Arduino. En la figura 1.29 se observa un ejemplo de placa apilable GPS con la ranura de almacenamiento SDCard y área de conexiones para añadir funcionalidades extras.

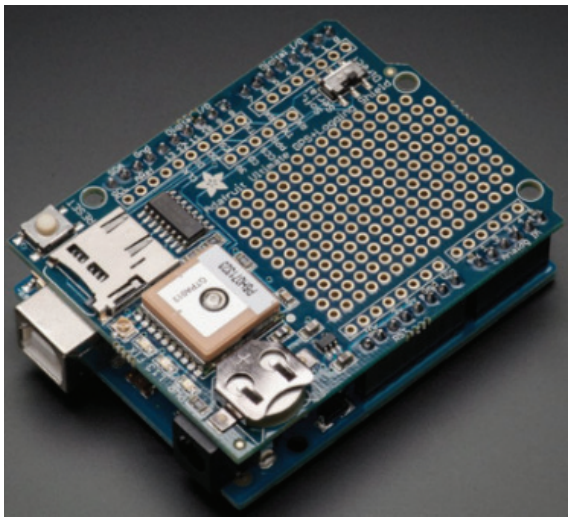


Figura 1.29 Shield GPS [20]

1.4.1.2.6 Xbee Shield

Como indica la figura 1.30 conecta el Arduino inalámbricamente a otros dispositivos que utilicen la tecnología Xbee.

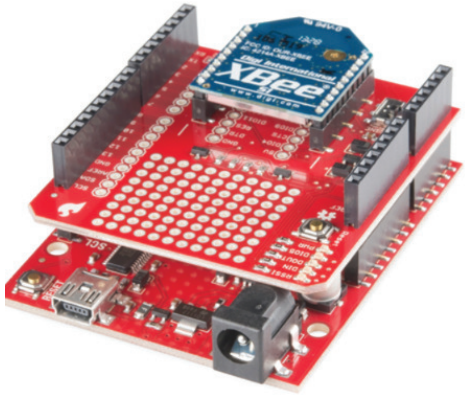


Figura 1.30 Shield para comunicación con Xbee [21]

1.4.1.2.7 Bluetooth Shield

Permite la comunicación mediante la tecnología Bluetooth en inalámbricas de área personal. En la figura 1.31 se observa un módulo Bluetooth compatible con Arduino.

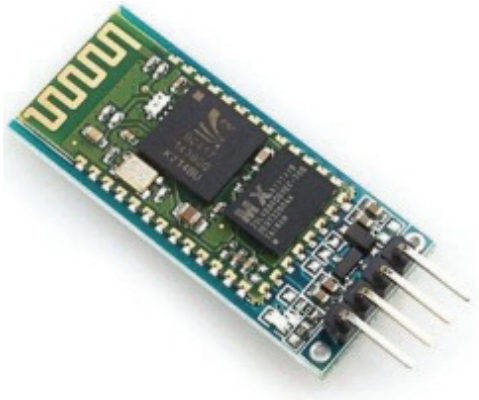


Figura 1.31 Shield Bluetooth [22]

1.4.1.3 Software Arduino

Es un software de código abierto que permite escribir el código para programar el funcionamiento de la placa Arduino y subirlo al microcontrolador [23].

1.4.1.3.1 IDE de desarrollo

Funciona bajo los sistemas operativos Windows, Mac OS X y Linux. El ambiente de desarrollo de Arduino está escrito en Java y está basado en lenguaje de código abierto Processing [24] como se indica en la figura 1.32.

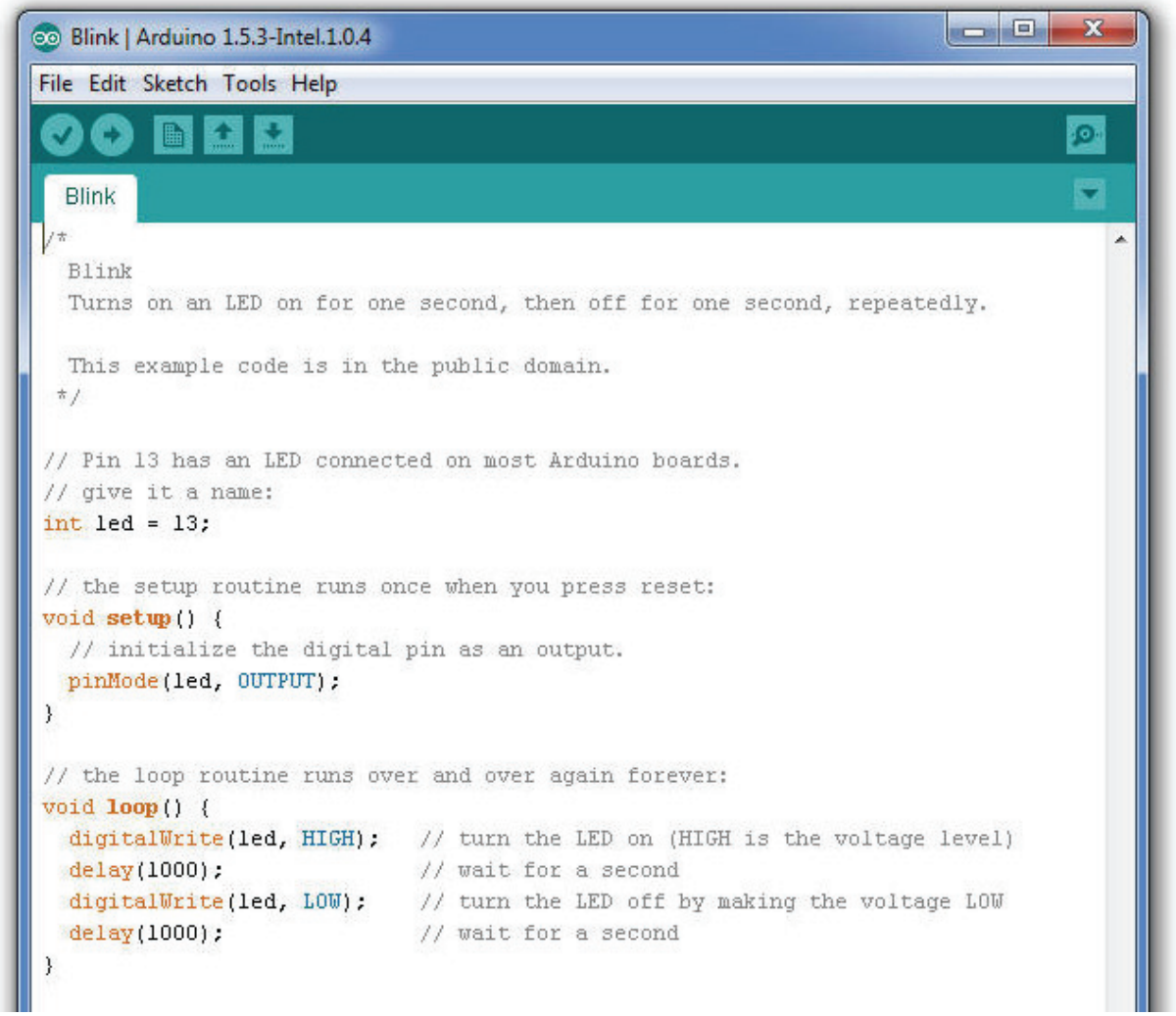


Figura 1.32 Software de desarrollo Arduino para Windows [25]

1.4.1.4 Plataforma de Web desarrollo Arduino

El editor web Arduino (figura 1.33 y 1.34) permite escribir código y subir los *scripts* a cualquier tarjeta Arduino o Genuino, también permite almacenar los proyectos en la nube y acceder desde cualquier dispositivo.

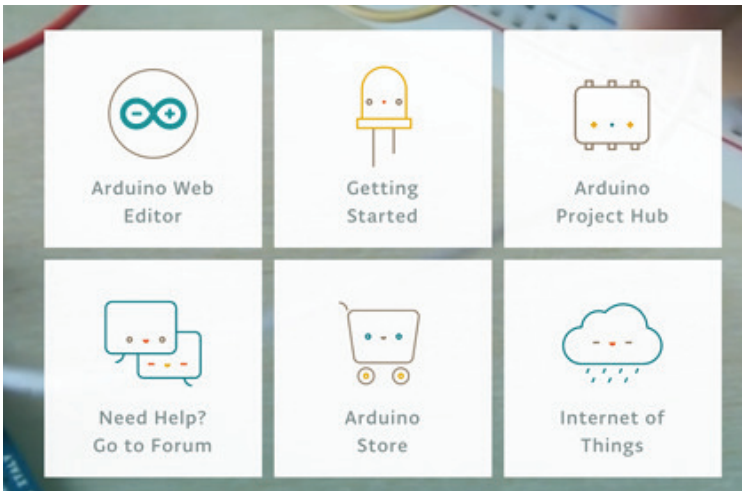


Figura 1.33 Plataforma de desarrollo Web para Arduino [26]

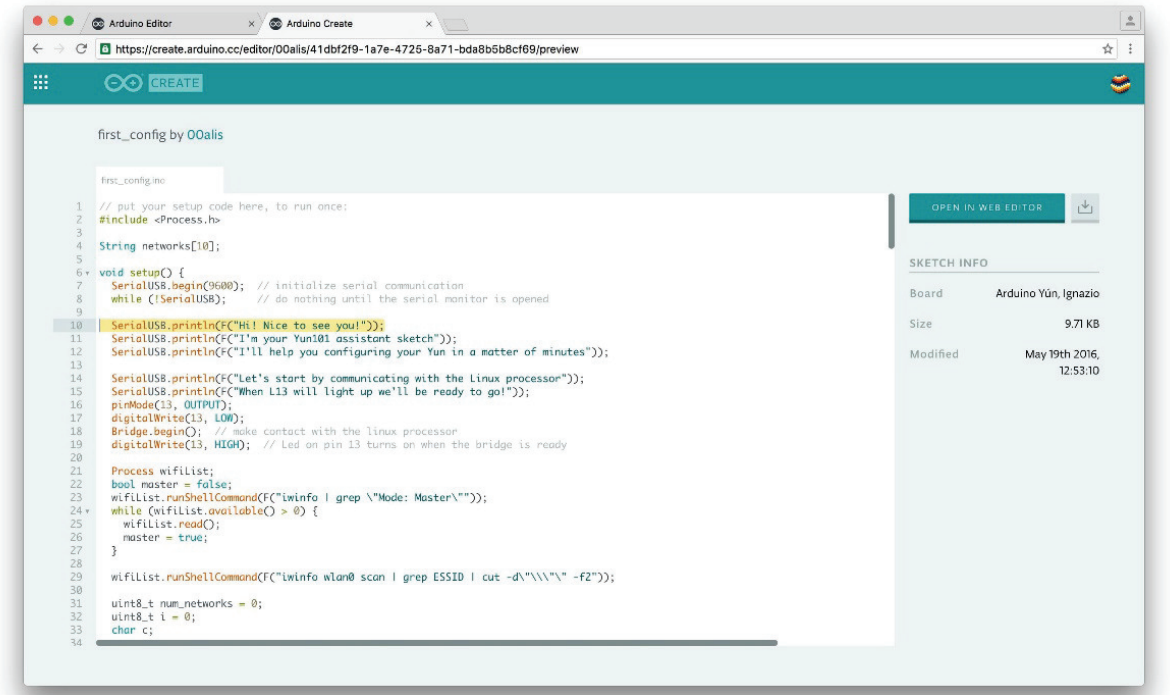


Figura 1.34 Interfaz Web de desarrollo para Arduino [27]

1.4.2 SISTEMA OPERATIVO ANDROID

1.4.2.1 Definición

Android es un *stack* de software de código abierto (figura 1.35) fue creado para funcionar en un amplio rango de dispositivos con diferentes funciones. El principal

propósito de Android es crear una plataforma disponible para proveedores de servicios de telecomunicaciones, fabricantes y desarrolladores que permita realizar ideas innovadoras en productos reales para mejorar la experiencia de los usuarios en terminales móviles.

Android fue desarrollado por un grupo de empresas conocidas como *The Open Handset Alliance*, liderada por Google con el objetivo de invertir en recursos de ingeniería y colaborativos para llevar Android al mercado de dispositivos móviles [28].

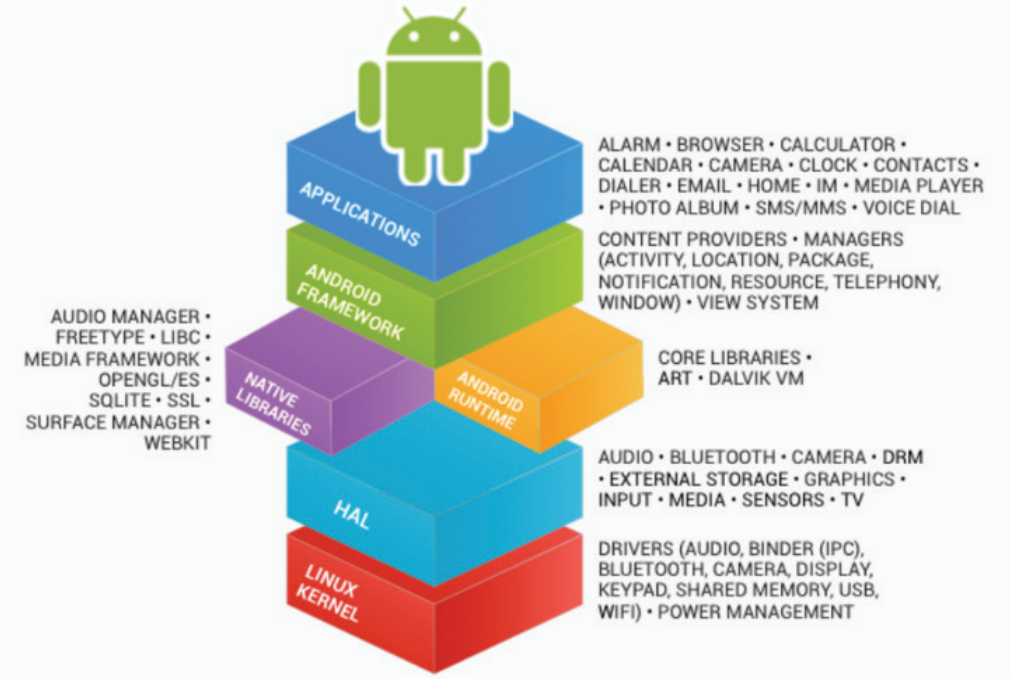


Figura 1.35 Stack Android [28]

1.4.2.2 Arquitectura del sistema operativo Android

Android provee la libertad de implementar un dispositivo con una amplia variedad de especificaciones y controladores. La capa de abstracción de hardware provee métodos estandarizados para crear software que permita interactuar el stack de Android con el hardware del dispositivo.

Se detallan las diferentes capas de la arquitectura del sistema operativo Android en la figura 1.36.

1.4.2.2.1 Capa Aplicación

Esta capa es utilizada por los desarrolladores de aplicaciones, los cuales deben estar al tanto de las API (Interfaz de programación de aplicaciones) que se asignan directamente a las interfaces HAL (Capa de abstracción de hardware) y proporcionar información sobre la implementación de los controladores.

1.4.2.2.2 Capa de Inter Proceso de Comunicación

Es un mecanismo que permite a la capa aplicación intercambiar mensajes y llamadas de procesos dentro del código de servicios del sistema Android, este procedimiento se realiza de manera transparente al desarrollador.

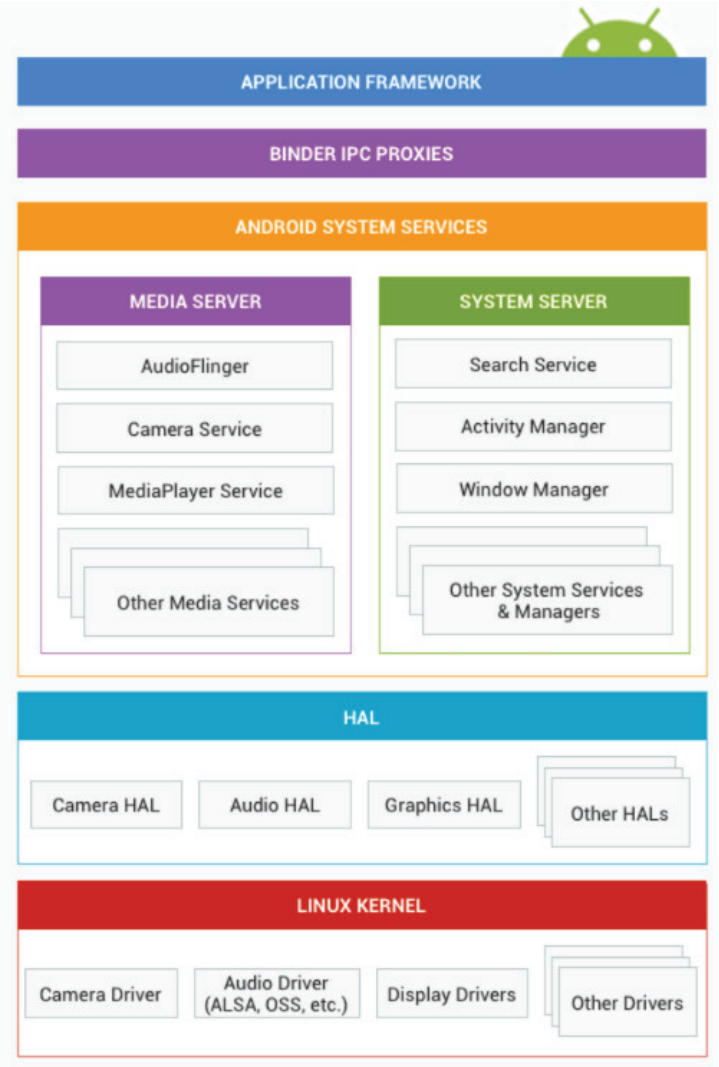


Figura 1.36 Arquitectura del Sistema Operativo Android [28].

1.4.2.2.3 Sistemas de servicios

Funcionalidad que permite a las API´s de la capa aplicación comunicarse con el Sistema de servicios para acceder al hardware subyacente. Estos servicios son modulares, enfocados al administrador de ventanas, servicios de búsqueda, o administrador de notificación.

Android incluye dos grupos de servicios: sistema (servicios como Administrados de ventana y notificaciones) y medios (servicios correspondientes a reproducción y grabación de medios)

1.4.2.2.4 Capa de Abstracción de Hardware (HAL)

La capa de abstracción de hardware (HAL) define un estándar de interfaces para proveedores de hardware para implementar y permitir a Android el funcionamiento de controladores de bajo nivel.

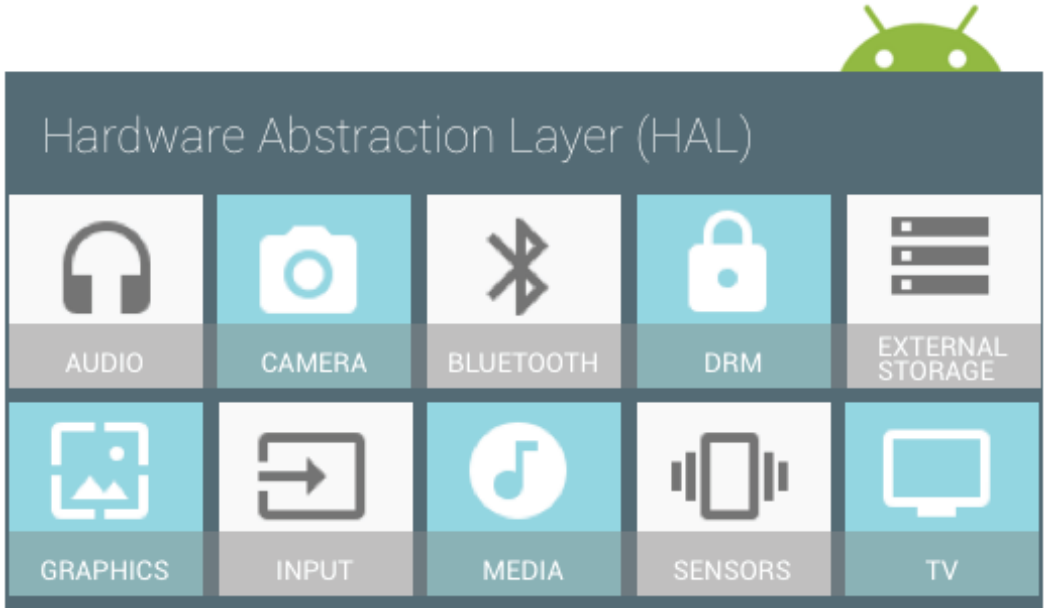


Figura 1.37 Componentes de la capa de abstracción HAL [28]

La HAL permite implementar funcionalidades sin afectar o modificar el sistema de alto nivel. Las implementaciones de HAL son paquetes dentro de módulos (.so) y son cargados por el sistema en el momento requerido. La figura 1.37 indica los módulos que contiene la HAL.

1.4.2.2.5 Kernel de Linux

El kernel permite desarrollar controladores de hardware para un dispositivo. Android usa una versión del kernel de Linux modificada, con funciones añadidas como *wake locks* (Sistema de administración de preservación de memoria) y otras características importantes para una plataforma embebida móvil.

Estas funciones adicionales son primariamente para la funcionalidad del sistema y no afectan al desarrollo de controladores [28].

1.5 METODOLOGÍA DE DESARROLLO KANBAN

La metodología de desarrollo Kanban fue creada por Toyota y ha sido utilizada para controlar el avance sistemático del trabajo enfocado en líneas de producción. Su término significa en japonés “tarjetas visuales” y su propósito es conseguir una mejora continua.

Kanban es la forma en que los equipos y organizaciones visualizan su trabajo, identifican y eliminan los cuellos de botella y consiguen mejoras operacionales con un mayor rendimiento y calidad.

Tal ha sido su éxito que, fuera de la industria automovilística se ha convertido en un popular método de trabajo que los equipos de software y las empresas lo emplean para trabajar a través de proyectos de cualquier dimensión.

1.5.1 DEFINICIÓN DE KANBAN

Kanban es un sistema de gestión de procesos en el que se determina la cantidad exacta de trabajo que el sistema es capaz de asumir.

Sirve para asegurar una producción continua y sin sobrecargas en el equipo de producción multimedia bajo el sistema de trabajo *just in time* lo que significa que evita sobre *stock*, que en la gestión de proyectos multimedia significa inversión innecesaria de tiempo y esfuerzo que se va a utilizar evitando sobrecargar al equipo.

En el mundo de software, este término ha sufrido su propia evolución considerándose como un método ágil para gestionar y mejorar la prestación de servicio en software de forma gradual y evolutiva [29].

1.5.2 PRINCIPIOS FUNDAMENTALES DEL MÉTODO KANBAN

Existen tres principios fundamentales del método Kanban que son:

- ✓ Iniciar con lo que se tiene ahora - proceso actual
- ✓ Perseguir un enfoque evolutivo para el cambio y mejora.
- ✓ Respetar las funciones y responsabilidades de equipo u organización.

Sobre la base de estos principios, el método describe las siguientes tres reglas básicas que se observan continuamente en organizaciones que experimentaron el éxito bajo este método.

1.5.2.1 Visualizar el trabajo y sus fases de ciclo de producción o flujo de trabajo

Kanban se basa en el desarrollo incremental dividiendo el trabajo en partes y describiendo cada una de estas actividades en un *post-it* colocándolos sistemáticamente en una pizarra. Cada uno de estos *post-it* contiene información variada referente a la descripción del trabajo y los tiempos estimados en la realización del mismo. El número de columnas como estados va a depender de la dimensión de la tarea como se puede ver en la figura 1.38.

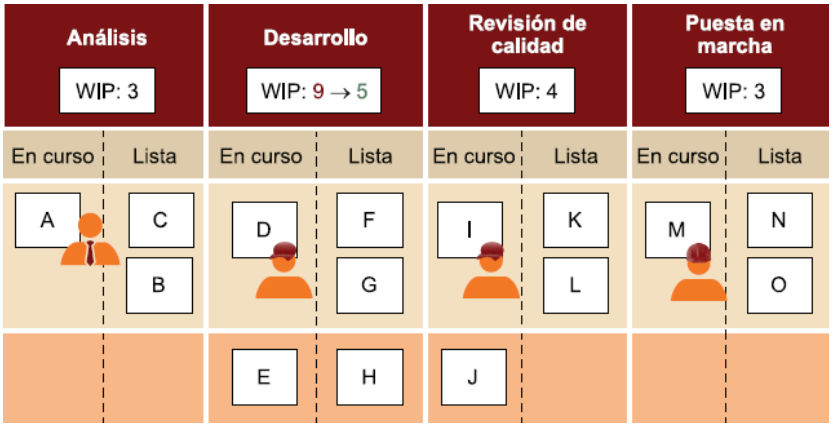


Figura 1.38 Ilustración Pizarra de Kanban [29]

1.5.2.2 Delimitar el Trabajo en Progreso (WIP)

Una de las principales ideas de Kanban es que el trabajo en progreso (*Work In Progress* o WIP) debe estar delimitado, es decir, el número de tareas a realizarse en cada fase del proceso debe ser un dato conocido.

Kanban debe determinar el número de tareas máximo a realizarse por fase, a ese número se lo conoce como límite del *Work in Progress* tomando en cuenta que, para iniciar una nueva tarea, alguna otra tarea previa debe haber finalizado [30]. En la figura 1.39 se puede observar el funcionamiento de un tablero de tareas Kanban.

Figura 1.39 Límites WIP [30]

1.5.2.3 Optimizar el flujo de trabajo

Esta regla consiste en determinar el tiempo que se tarda en terminar cada tarea, este tiempo se conoce *como lead time y* es contado desde que se realiza la petición hasta que se hace la entrega.

Es recomendable anotar las fechas de entrada y salida de cada tarea por cada fase, de este modo podemos obtener una serie de gráficos del tiempo que tardan cada una de las tareas y cuáles son las fases que demandan mayor tiempo.

La utilización del gráfico acumulativo como se indica en la figura 1.40 proporcionará mucha información a ser observada ya que a simple vista podemos determinar las

tareas pendientes al día a día y cuantas hay aproximadamente en producción. Además, se puede ver cual tarea tiene una pendiente más pronunciada ya que si crecen las tareas pendientes frente a las entregadas al cliente se habrá generado un problema.

Además del *lead time* existe otra métrica normalmente utilizada conocida como *cycle time,* la primera mide lo que ven los clientes mientras que la segunda mide el rendimiento del proceso [31].

Al dividir el *cycle type* por el WIP se obtiene el rendimiento de trabajo denominado *Throughput* que es la cantidad de ítems que un equipo puede finalizar en un tiempo dado [32].

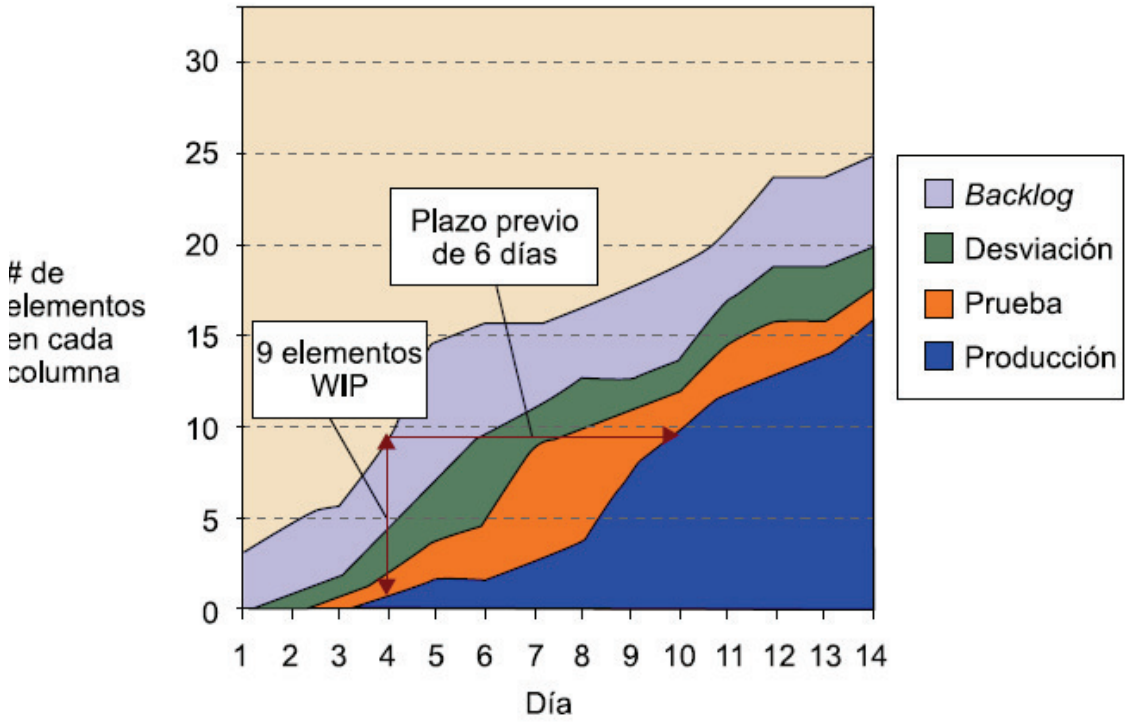


Figura 1.40 Gráfico de la optimización del flujo de trabajo [32]

CAPÍTULO 2 - DISEÑO DE LA APLICACIÓN

2.1 INTRODUCCIÓN

En este capítulo se realizará el diseño del prototipo estableciendo los sensores adecuados para la detección del ángulo ciego y la detección de velocidad de circulación del vehículo mediante una aplicación en un teléfono Android.

Se describe el funcionamiento del prototipo de la siguiente manera:

- El sistema entrará en funcionamiento al momento de encender el vehículo.
- El módulo Arduino realizará la búsqueda del teléfono Android.
- Establecerá la conexión entre ambos.

Una vez terminado el procedimiento anterior el sistema permitirá lo siguiente:

- Detectar objetos en los laterales del vehículo que se encuentren a una distancia entre 1.50 m y 2.50 m.
- Enviar los datos del sensor para ser analizados y procesados por la aplicación en Android.
- Mostrar y analizar la velocidad de circulación del vehículo.

Para el desarrollo del prototipo se utilizarán dos sensores ultrasónicos que permitan detectar objetos mediante reflexión de ondas emitidas desde el vehículo en movimiento, estos se ubicarán en los laterales del vehículo de tal forma que ofrezcan cobertura en su ángulo ciego.

La comunicación entre los sensores y la aplicación Android se realizará usando una placa Arduino la cual transmitirá la información mediante Bluetooth al teléfono Android para que esta sea procesada. Ver figura 2.1.

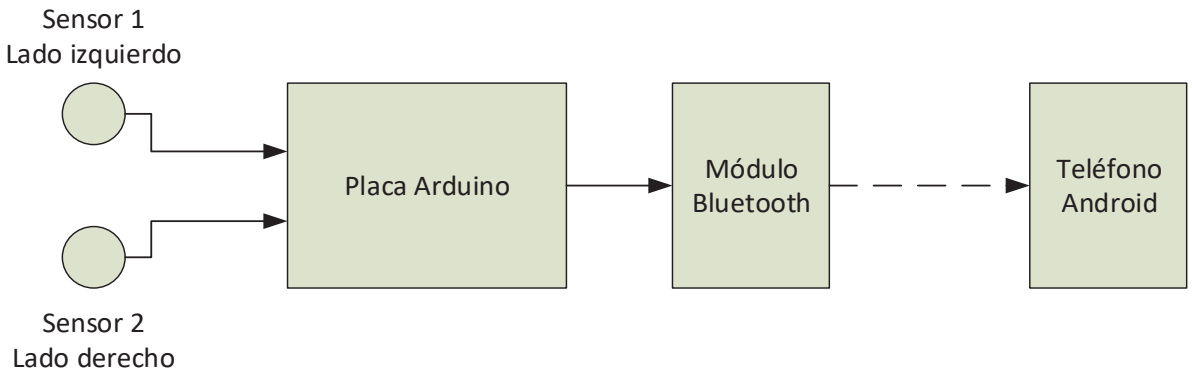


Figura 2.1 Descripción gráfica del diseño que se implementará.

2.2 REQUERIMIENTOS DE SOFTWARE Y HARDWARE

Para el correcto funcionamiento del prototipo se describirá los requerimientos mínimos tanto de software como de hardware.

2.2.1 SOFTWARE DE DESARROLLO

- ANDROID STUDIO V2.3.2.
- Librerías de soporte para Bluetooth 2.3 o superior y GPS.
- ARDUINO IDE 1.8.2.
- Librerías de soporte para sensor ultrasónico Maxbotix.

2.2.2 HARDWARE

- Teléfono con Sistema Operativo Android 4.3 Jelly Bean o superior, recibe y procesa la información de ángulo ciego, además calcula la velocidad de circulación del vehículo para emitir las diferentes alertas según sea el caso indicado.
- Tarjeta de desarrollo Arduino UNO, recibe y procesa la información de distancia de los sensores ultrasónicos.

- Sensores ultrasónicos con rango de cobertura mínimo 4 m: MB1000 LV-MaxSonar®-EZ0 que proveerán la información relacionada al ángulo ciego.
- Módulo Bluetooth HC-06, se conecta a la placa Arduino para enviar de forma inalámbrica la información hacia el teléfono Android.
- Regulador a 5 VDC con salida USB.
- Cables de conexión.

2.3 SENSOR ULTRASÓNICO

En base a lo consultado en el capítulo primero sobre los sensores disponibles en el mercado se ha decidido utilizar el sensor ultrasónico Maxbotix MB1000 (figura 2.2) en base a las siguientes consideraciones:

Figura 2.2 Sensor Maxbotix MB1000 [33]

2.3.1 BENEFICIOS

- Permite la detección de objetos sobre superficies transparentes, brillantes o con presencia de lluvia.
- Condiciones tales como polvos, suciedad o niebla no afectan su funcionamiento.
- Su detección es estable.
- Bajo costo.

- Bajo consumo de energía.
- El sensor procesa directamente la información liberando ciclos de trabajo al procesador.
- Medición de rápida respuesta, mediante disparos internos o externos.
- Alcance 15 cm hasta 645 cm con resolución de 3 cm.

2.3.2 CARACTERÍSTICAS

- Voltaje de alimentación de 2.5 V - 5.5 V @ 2 mA.
- Lectura de muestreo cada 50 ms (20 Hz).
- Transmisión serial, 0 a Vcc, 9600 baudios, 8-1-N.
- Ancho de pulso 142 us/pulgada.
- El sensor opera a 42 KHz.

2.3.3 DESCRIPCIÓN DEL ALCANCE DE DETECCIÓN

Los resultados de la muestra para el patrón de haz medido se muestran en una cuadrícula de 30 cm. El patrón de detección se muestra para los lóbulos de varios diámetros que se genera delante del sensor como indica la figura 2.3.

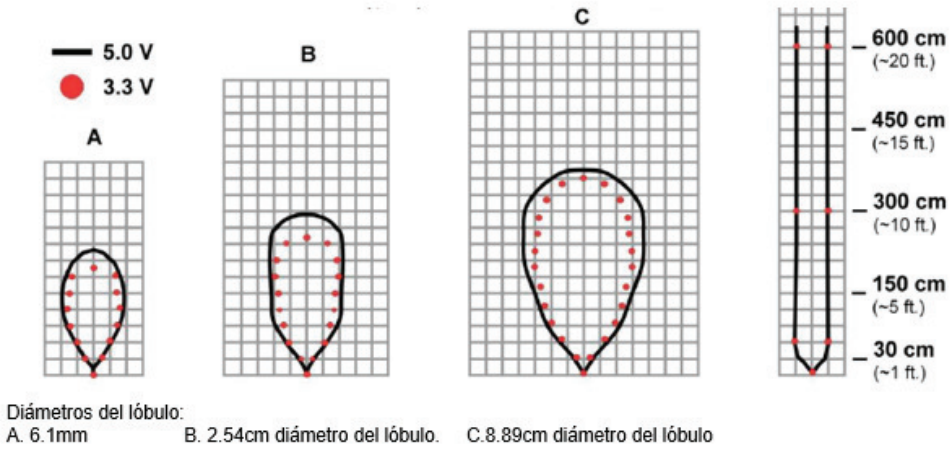


Figura 2.3 Lóbulos de alcance del sensor Maxbotix MB1000 [34]

2.3.4 APLICACIONES Y USOS

- Detección de proximidad en zonas.
- Detección de personas.
- Navegación automática.
- Arreglo de múltiples sensores.
- Medición de distancias.
- Detección de objetos de gran tamaño.

2.3.5 DESCRIPCIÓN DE PINES

La figura 2.4 muestra la descripción de pines para el sensor ultrasónicos MB1000 y su forma física.

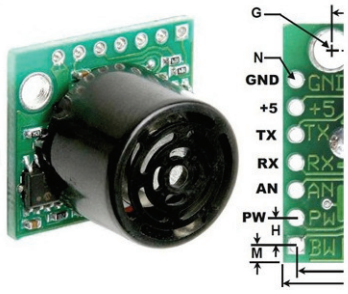


Figura 2.4 Descripción de pines del sensor Maxbotix MB1000 LV [34]

Pin 1 – BW: Dejar abierto el pin o mantener en 0L para modo de retransmisión serial al utilizar TX como salida.

Pin 2 – PW: Este pin de salida utiliza el ancho de pulso para representar la distancia la cual puede calcularse utilizando el factor de escala de 147 uS por pulgada.

Pin 3 – AN: Salida analógica de voltaje con factor de escala de (Vcc/512) por pulgada.

Pin 4 – RX: Este pin está configurado internamente en alto.

Pin 5 – TX: Cuando el pin BW está abierto o en bajo (0L), la salida TX se entrega los datos en formato serial RS232. La salida es un ASCII “R” seguido por 3 caracteres que representan el rango de pulgadas hasta un máximo de 255, seguido por un carácter de retorno (ASCII 13). La velocidad de transmisión es 9600, 8 bits, no paridad, con 1 bit de parada.

Pin 6 - +5V Vcc: Opera con voltajes entre 2.5 V – 5.5 V. Se recomienda una corriente con capacidad de 3 mA para 5 V y 2 mA para 3 V.

Pin 7 – GND: Retorno de la alimentación DC, GND y Vcc debe ser una señal libre de rizado y ruido para una mejor operación del sensor.

2.3.6 CONEXIÓN DEL SENSOR CON EL MÓDULO ARDUINO

Este sensor se puede conectar de 3 formas hacia la interfaz Arduino:

- Interfaz análoga.
- Interfaz por modulación de ancho de pulsos.
- Interfaz serial.

2.3.6.1 Interfaz análoga

La interfaz análoga envía los datos del sensor de forma directamente proporcional entre la distancia y la salida de voltaje. La conexión con la placa Arduino se puede observar en la figura 2.5.

Salida de voltaje análoga con factor de escala de (Vcc/512) por pulgada

Ejemplo:

- Con alimentación de 5 V, tenemos una salida de 9.8 mV/pulgada.
- Con alimentación de 3.3 V, tenemos una salida de 6.4 mV/pulgada.

La exactitud de voltaje de salida analógico presenta menor exactitud que la conexión en modo PWM y Serial [35]

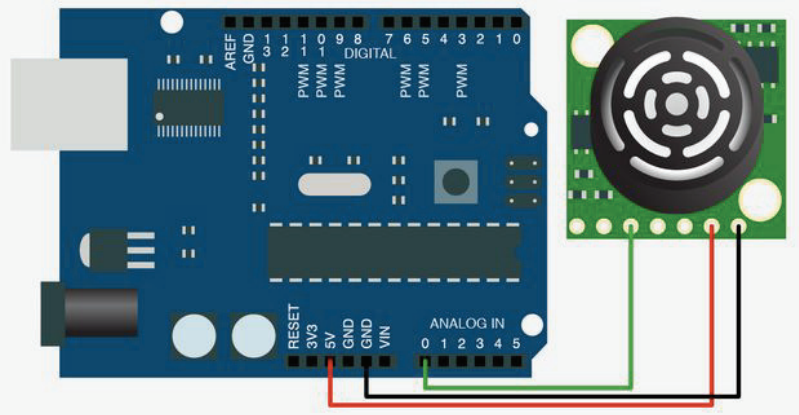


Figura 2.5 Conexiones mediante interfaz de salida análoga [35].

2.3.6.2 Interfaz por modulación de ancho de pulsos

Esta interfaz por ancho de pulso “PW” es otra manera de recuperar información del sensor. La conexión con la placa Arduino se indica en la figura 2.6.

La salida del pin es un pulso donde el ancho del mismo representa la distancia, puede ser calculada utilizando el factor de escala de 147 µs por pulgada.

Para ser utilizada esta conexión en Arduino es necesario utilizar la librería pulseIn().

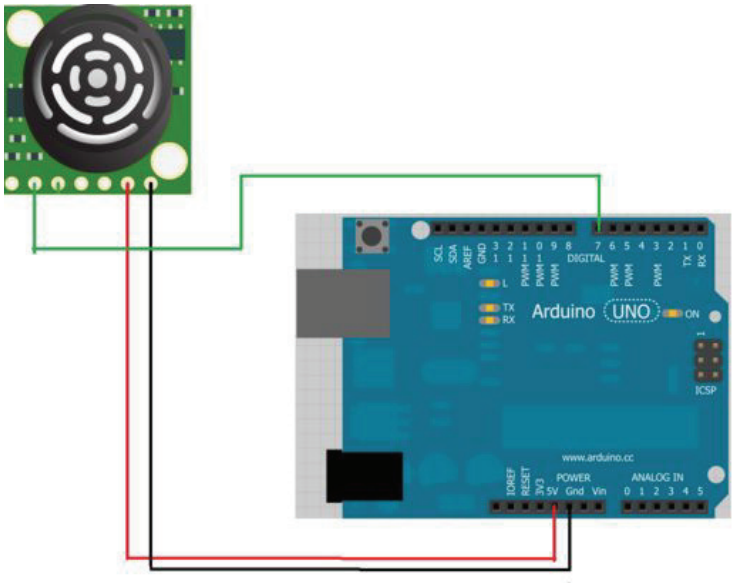


Figura 2.6 Conexiones mediante interfaz de salida PW [35]

2.3.6.3 Interfaz Serial

En este método se utiliza la librería software serial para escribir el código del sensor y una de las salidas digitales del Arduino, dejando abierto o en 0 lógico para la salida Tx del Arduino. La conexión con la placa Arduino se indica en la figura 2.7.

La salida es un carácter ASCII “R” (0x52) seguido por 3 dígitos ASCII que representan el rango en pulgadas hasta un máximo de 255, seguido por un retorno de línea (ASCII 13 o 0x0D).

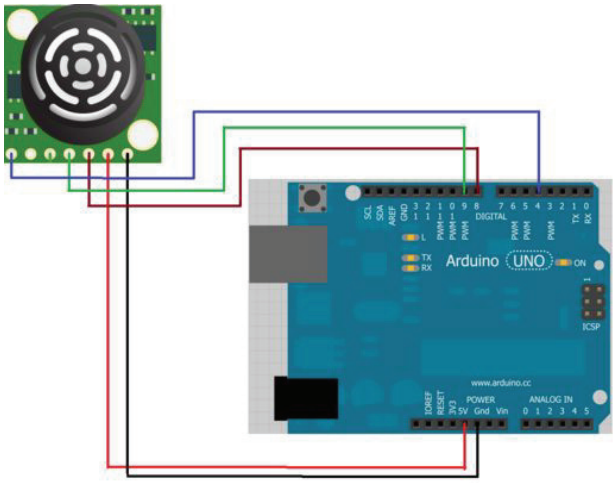


Figura 2.7 Conexiones mediante interfaz de salida serial [35]

2.3.7 CONEXIÓN MÚLTIPLE DE SENSORES POR MODULACIÓN ANCHO DE PULSO (PW)

La conexión de múltiples sensores se puede realizar de 3 maneras diferentes:

- Libre conexión
- Operación simultánea
- Lectura secuencial

2.3.7.1 Libre Conexión

Si se conectan varios sensores a una misma placa Arduino dejando el Pin 4 (RX) sin conexión, los sensores estarán fuera de sincronía lo que causará interferencia

en la mayoría de aplicaciones, ver figura 2.8. No se recomienda este tipo de conexión ya que regularmente enviará información errónea sobre la lectura de la distancia.

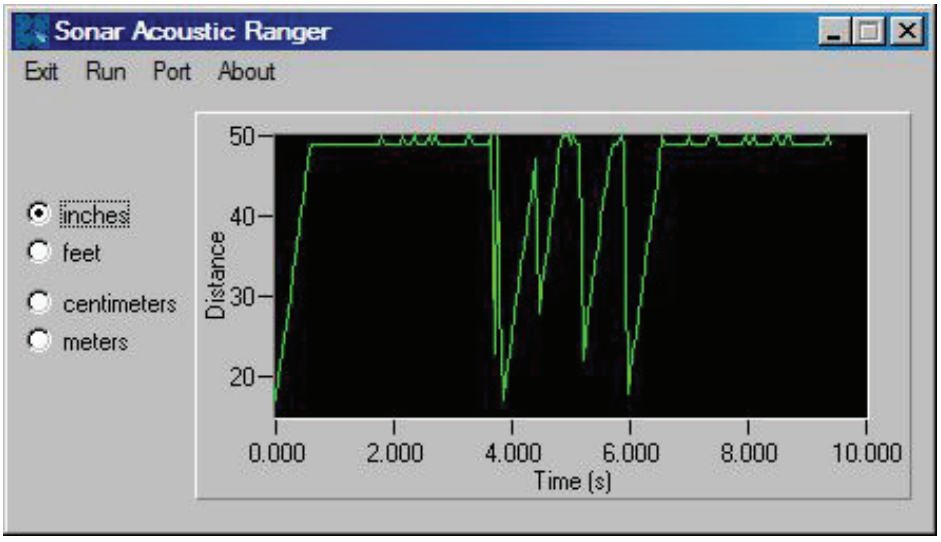


Figura 2.8 Interferencia de los sensores en libre conexión.

2.3.7.2 Operación Simultánea

Conectar los sensores Maxbotix maxsonar con el común al RX permite que la lectura se realice en todos los sensores a la vez como se indica en la figura 2.9 este método es conveniente para utilizar la salida de voltaje análoga. La lectura se realizará cada 50 ms.

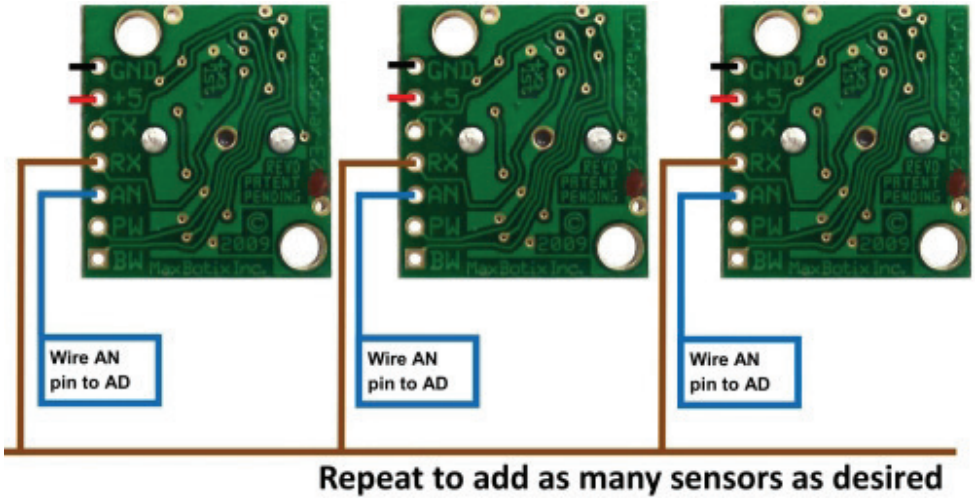


Figura 2.9 Conexión simultánea de los sensores.

2.3.7.3 Lectura Secuencial

Para la lectura secuencial se conecta el dispositivo de disparo al pin 4 (RX) del primer sensor, entonces el pin 5 (TX salida) del primer sensor se conecta al pin RX del siguiente sensor, así hasta completar hasta el último sensor, este permite encadenar hasta 10 sensores, de manera que las lecturas se realizan secuencialmente cada 50 ms. Ver figuras 2.10 y 2.11.

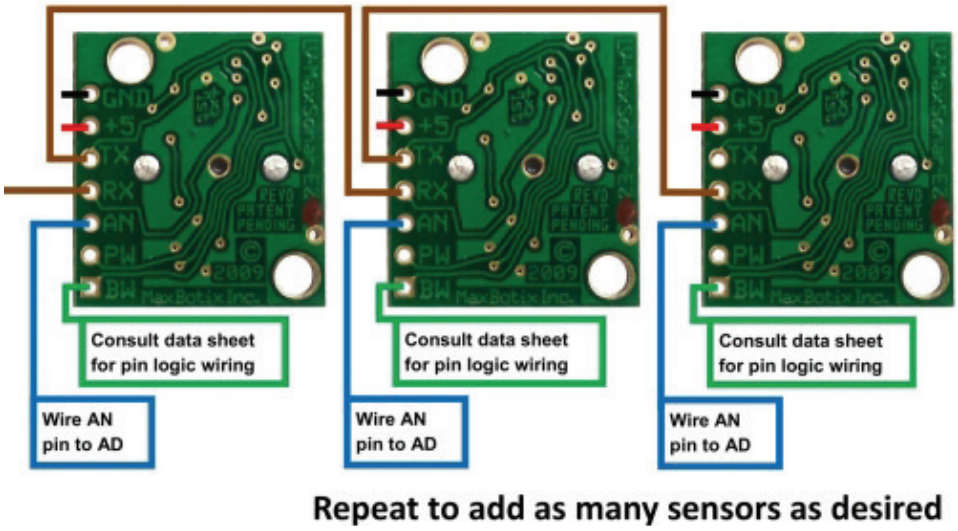


Figura 2.10 Conexión de funcionamiento secuencial de los sensores.

Una variante de este método es el encadenamiento secuencial continuo en el cual el pin TX del último sensor se conecta al Pin Rx del primer sensor a través de un resistor de 1 KΩ.

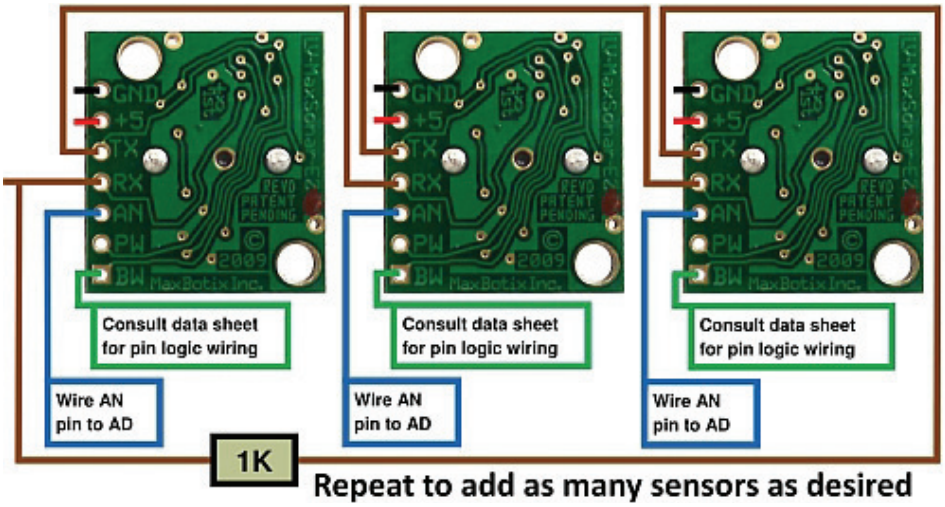


Figura 2.11 Conexión con encadenamiento continuo.

2.4 PLACA DE DESARROLLO ARDUINO UNO

Arduino Uno es una placa de hardware libre basada en el microcontrolador ATmega328.

Se detallan las características de la placa Arduino Uno en la tabla 2.1.

Tabla 2.1 Características de la placa Arduino Uno

ARDUINO UNO	
Microcontrolador	ATmega328
Arquitectura	AVR
Voltaje de operación	5 V
Memoria Flash	32 KB de la cual 0.5 KB son usados por el bootloader
SRAM	2 KB
Velocidad de reloj	16 MHz
Pines I/O Análogos	6
EEPROM	1 KB
Corriente DC de pines I/O	40 mA on I/O Pins; 50 mA on 3,3 V Pin
Voltaje de entrada	7-12 V
Pines Digitales I/O	20 (de los cuales 6 proveen salidas PWM)
Salida PWM	6
Tamaño PCB	53.4 x 68.6 mm
Peso	25 g

2.4.1 DESCRIPCIÓN DE PINES

Como se puede observar en la Figura 2.12 algunos de ellos cumplen doble función la cual puede ser configurada desde el IDE de desarrollo Arduino según sea el uso que necesitemos para nuestra aplicación.

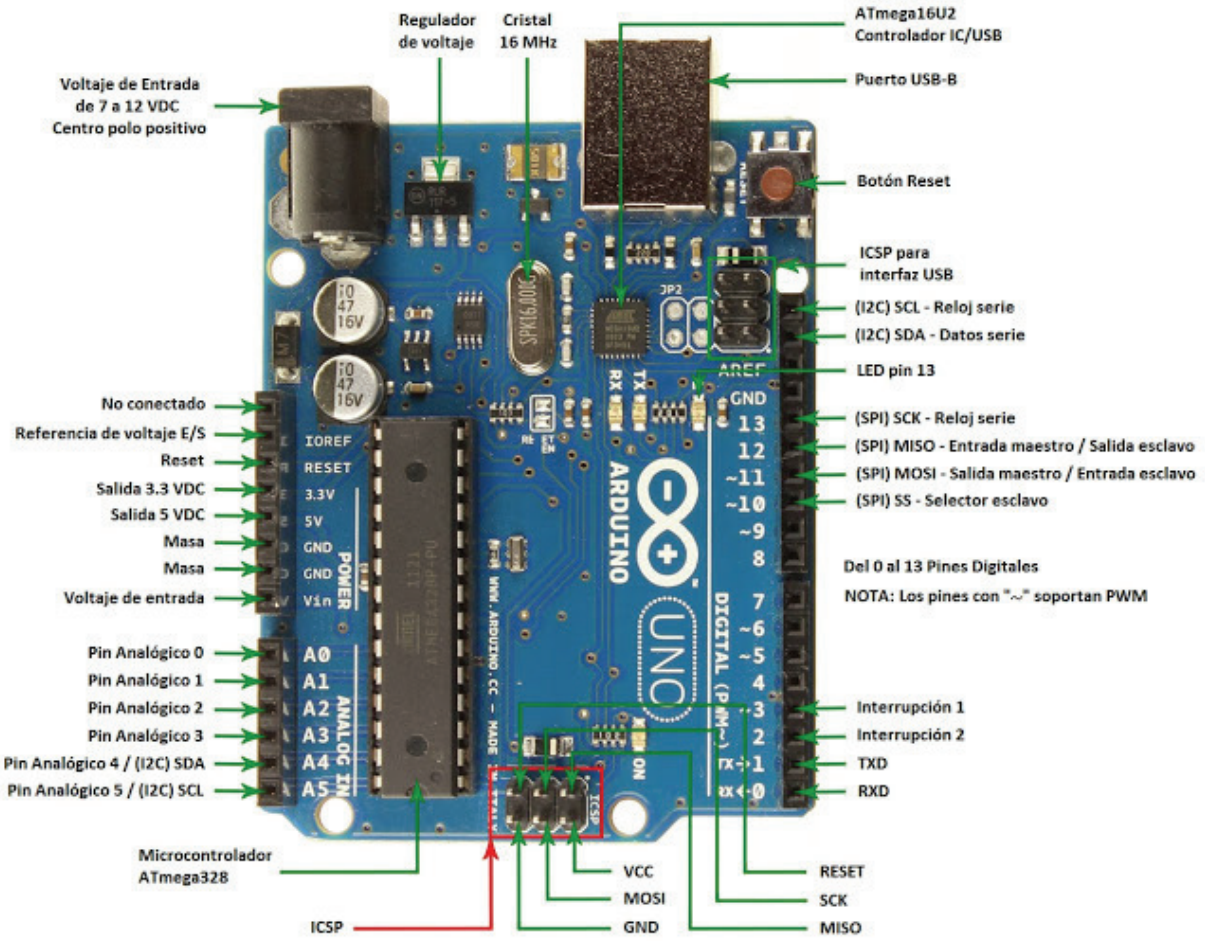


Figura 2.12 Descripción de pines – Arduino Uno [36]

2.5 MÓDULO BLUETOOTH HC-06

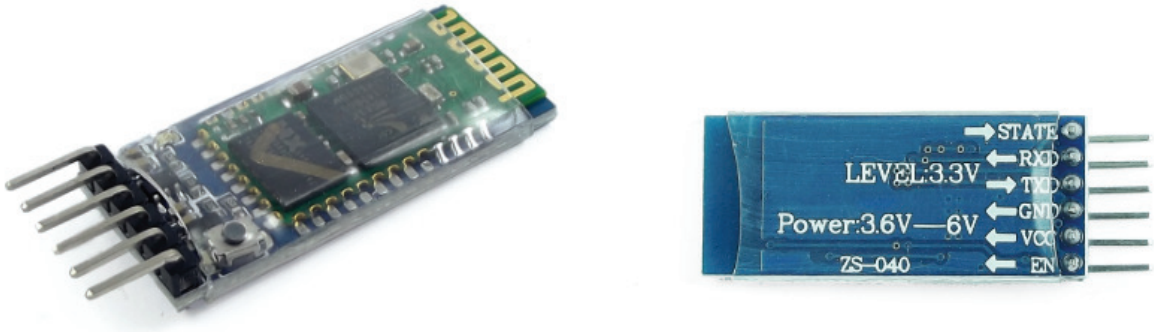


Figura 2.13 Presentación física y descripción de pines del módulo Bluetooth HC-06 [37]

El módulo Bluetooth HC-06 (figura 2.13) es una placa que mantiene compatibilidad de voltajes y velocidad de transmisión para trabajar con las placas Arduino, viene de fábrica con la configuración ESCLAVO, mediante el uso de comandos AT se puede cambiar la configuración como el nombre, velocidad de transmisión, contraseña de vinculación y otros parámetros más.

La configuración por defecto del módulo HC-06 se muestra en la tabla 2.2.

Tabla 2.2 Configuración por defecto del módulo HC-06

HC-06	
MODO	ESCLAVO
NOMBRE	HC-06
VELOCIDAD DE TRANSMISIÓN	9600
CONTRASEÑA DE VINCULACIÓN	1234

2.5.1 ESTADOS DE FUNCIONAMIENTO DEL MÓDULO HC-06

2.5.1.1 Desconectado

Es el estado en el que se enciende cuando se conecta a la alimentación el módulo, en este estado el LED parpadea rápidamente a una frecuencia de 100 ms. En este estado se permite enviar comandos AT.

2.5.1.2 Conectado

Al establecer una conexión el módulo entra a este estado, el LED permanece encendido, el pin RX se encarga de recibir los datos y se envían por el pin TX. En este estado el módulo no puede interpretar comandos AT [22].

2.6 DIAGRAMA DE CONEXIÓN ELÉCTRICA DEL VEHÍCULO

En el diagrama de la Figura 2.14 se muestra la conexión eléctrica a realizarse en el vehículo, es necesario tomar la alimentación de la batería de 12 VDC a la cual se conecta el regulador de voltaje a 5 VDC con el cual se energiza los diferentes dispositivos del prototipo, adicional es necesario instalar el cableado que permita la ubicación de los sensores en los laterales del vehículo.

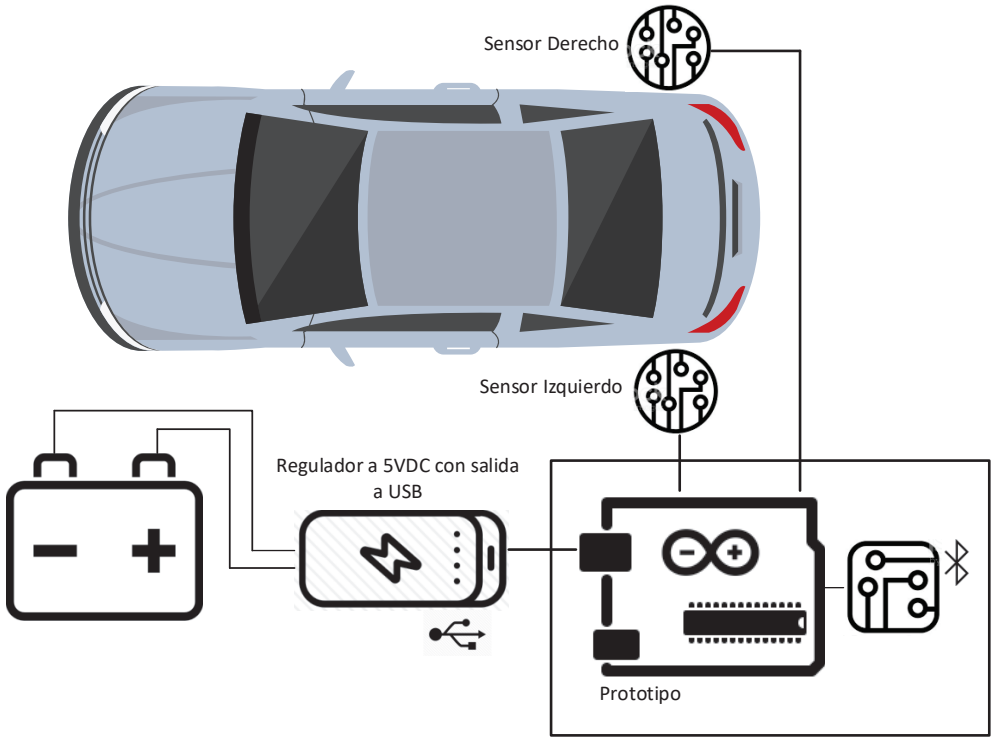


Figura 2.14 Diagrama de conexión eléctrica del vehículo.

2.7 APLICACIÓN ARDUINO

En la figura 2.15 se detallan las diferentes acciones a realizar entre los sensores, la placa Arduino y el módulo Bluetooth para ser enviados hacia el teléfono Android.

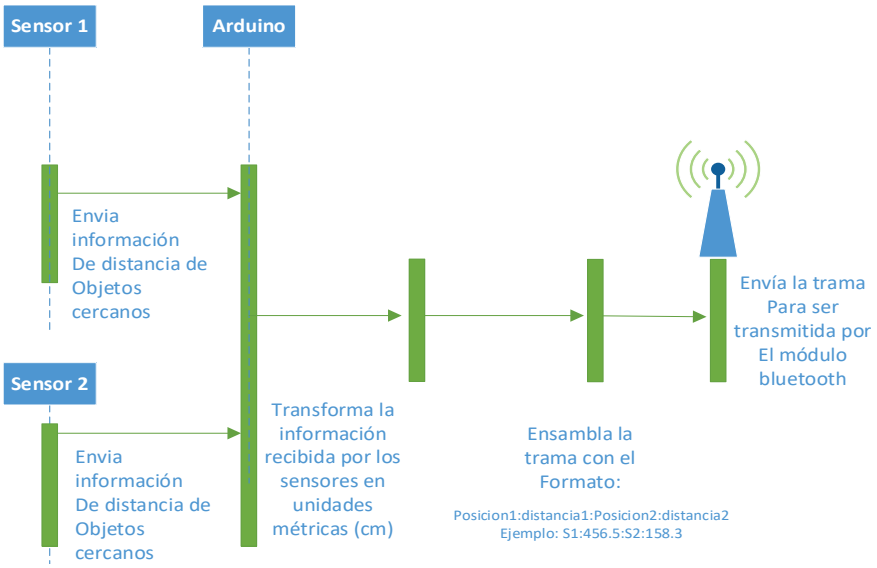


Figura 2.15 Diagrama de funcionamiento entre Sensores-Arduino-Bluetooth

2.7.1 CÓDIGO DE LA APLICACIÓN

En el siguiente código se explica las funciones necesarias para implementar el script en la placa Arduino y procesar la información entregada por los sensores ultrasónicos.

- Definición de variables que asigna los pines de lectura para cada sensor: (Código 2.1)

```
const int pwPin1 = 3;
const int pwPin2 = 5;
```

Código 2.1 Definición de pines Arduino

- Definición de pin de disparo, ver código 2.2.

```
const int triggerPin1 = 13;
```

Código 2.2 Pin de disparo Arduino

- Variables para almacenar lecturas, ver código 2.3

```
long pulse1, pulse2, sensor1, sensor2;
```

Código 2.3 Variables para almacenar distancia

- Configuración del estado inicial de la transmisión y estado de pines asignados a los sensores como indica el código 2.4.

```
void setup () {
  Serial.begin(9600);
  pinMode(pwPin1, INPUT);
  pinMode(pwPin2, INPUT);
  pinMode(triggerPin1,OUTPUT);
}
```

Código 2.4 Inicialización de enlace serial

- Lectura de sensores y transmisión de datos

  El tiempo de retardo de la función delay() dependerá del número de sensores conectados, se recomienda como mínimo asignar por cada sensor 50 ms, para el ejemplo del código 2.5, el retardo se asignaría de 100 ms.

```
void loop () {
  start_sensor();
  read_sensor();
```

```
  printall();
  delay(100);
}
```

Código 2.5 Lectura de datos del sensor ultrasónico

- Funciones:

  Realiza la lectura del valor registrado por el sensor y los almacena en una variable, ver código 2.6.

```
void read_sensor(){
 pulse1 = pulseIn(pwPin1, HIGH);
 pulse2 = pulseIn(pwPin2, HIGH);
 sensor1 = pulse1/147*2.54;
 sensor2 = pulse2/147*2.54;
}
```

Código 2.6 Lectura de datos del sensor

En el código 2.7 se indica cómo se activa el pin de disparo para iniciar la lectura de los sensores.

```
void start_sensor(){
  digitalWrite(triggerPin1,HIGH);
  delay(1);
  digitalWrite(triggerPin1,LOW);
}
```

Código 2.7 Inicio de Lectura del sensor

En el código 2.8 se ensambla la cadena para ser transmitida mediante el pin de comunicación serial a través del módulo Bluetooth.

```
void printall(){
  Serial.print("S1");
  Serial.print(":");
  Serial.print(sensor1);
  Serial.print(":");
  Serial.print("S2");
  Serial.print(":");
  Serial.print(sensor2);
  Serial.println(" ");
}
```

Código 2.8 Estructura de la cadena de texto a enviarse por Bluetooth

2.7.2 CONFIGURACIÓN DE LA CONEXIÓN ANDROID – ARDUINO

El envío de información se realizará utilizando los pines de transmisión (TX, pin 1) y recepción (RX, pin 0) del Arduino, como se indica en la tabla 2.3 [38]:

Tabla 2.3 Conexión de pines entre Arduino y módulo Bluetooth

Pines	
Arduino	Bluetooth
RX (Pin 0)	TX
TX (Pin 1)	RX
5 V	VCC
GND	GND

En la Figura 2.16, se indica las conexiones a realizarse entre el módulo Bluetooth y Arduino.

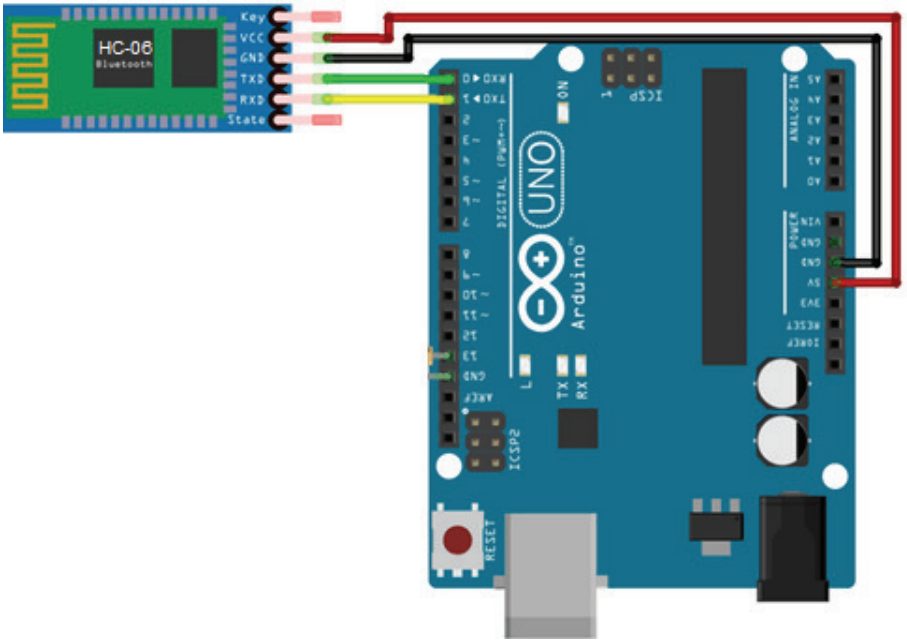


Figura 2.16 Diagrama de conexión eléctrica entre Arduino y el módulo Bluetooth HC-06

2.8 APLICACIÓN ANDROID

El desarrollo de la aplicación para el dispositivo Android permitirá al usuario visualizar los datos obtenidos desde los sensores ubicados en el ángulo ciego y la velocidad actual a la que circula el vehículo, además de un menú de configuración para modificar el valor de la velocidad a la que se activan las alertas.

Se describirá los diagramas de funcionamiento de la aplicación.

2.8.1 DIAGRAMAS DE SECUENCIA

Se describirá la secuencia de funcionamiento de los diferentes módulos que componen el prototipo correspondiente a la aplicación en Android.

2.8.1.1 Secuencia de funcionamiento para información de ángulo ciego

La figura 2.17 muestra el proceso para la información enviada por los sensores a través del Arduino y el módulo Bluetooth.

El proceso inicia al lanzar la aplicación en Android, se enciende la conexión Bluetooth el cual permite acceder a la lista de los dispositivos emparejados, luego la aplicación escanea por nuevos dispositivos cercanos y los añade a la lista.

El usuario podrá seleccionar el dispositivo que corresponda al módulo Bluetooth que por defecto se nombra como HC-06. Si la conexión se realiza correctamente la información enviada por los sensores se procesa y visualiza en la interfaz del usuario.

El caso de no realizarse la conexión la aplicación intenta nuevamente emparejar con el dispositivo previamente seleccionado, lo mismo ocurre en caso de pérdida de conexión.

Al terminar la aplicación se cerrará la conexión con el módulo Bluetooth el cual queda en espera hasta iniciar nuevamente la aplicación en Android.

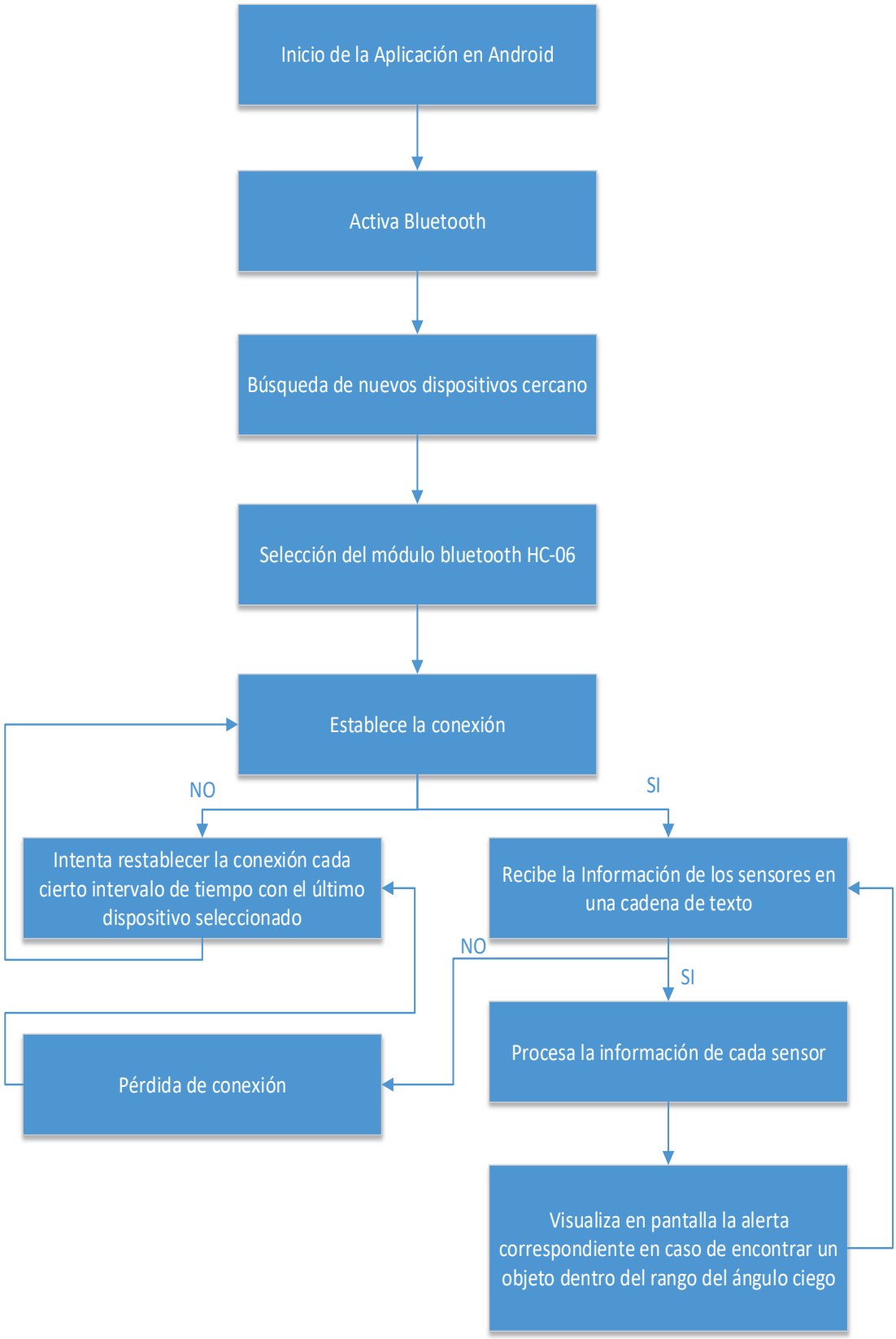


Figura 2.17 Diagrama funcionamiento ángulo ciego

2.8.1.2 Secuencia de funcionamiento para la información de Velocidad de circulación y exceso de límites de velocidad

El siguiente diagrama (figura 2.18) indica el proceso para determinar la velocidad de circulación del vehículo y la detección al sobrepasar los límites de velocidad establecidos.

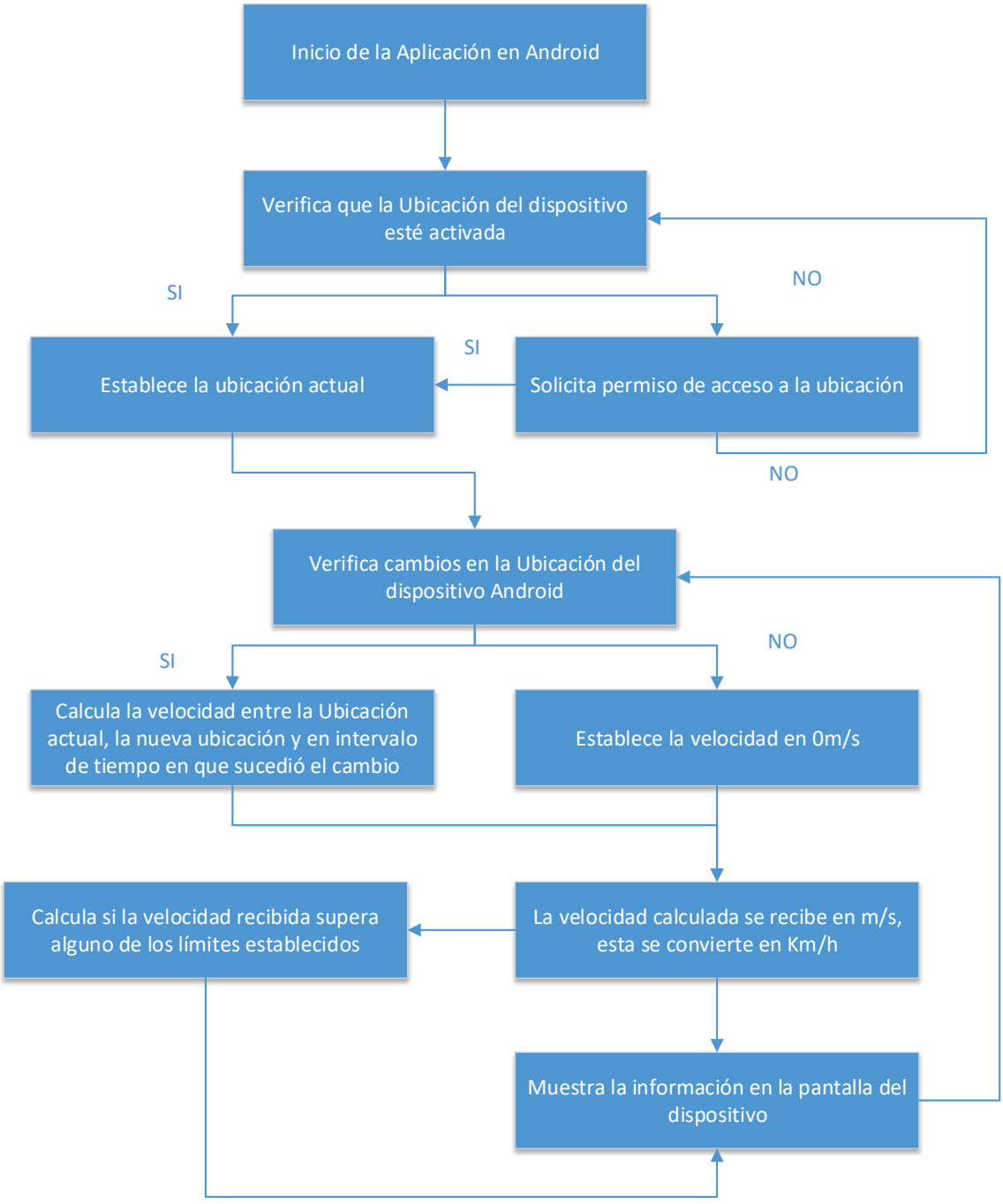


Figura 2.18 Diagrama funcionamiento velocidad y alertas límites de velocidad

2.8.2 DIAGRAMAS DE CLASE

El diagrama que se muestra en la figura 2.19 detalla todos los métodos necesarios para desarrollar e implementar la aplicación que procesará y mostrará la información al usuario.

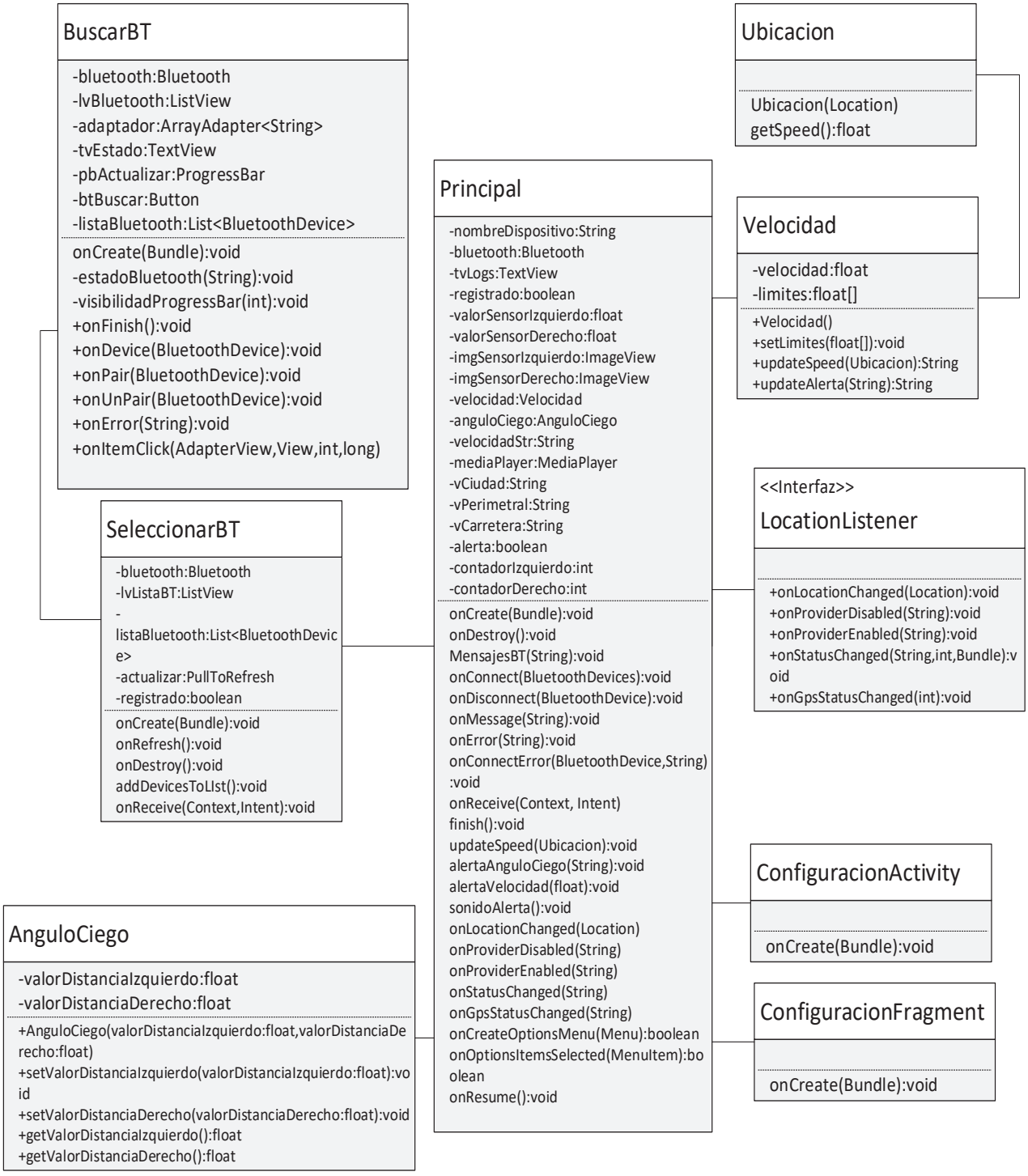


Figura 2.19 Diagrama de clases de la aplicación Android

La interfaz de visualización de datos y configuración se realizará en un teléfono con Sistema Operativo Android la cual constará de los siguientes módulos:

- Módulo Velocidad.
- Módulo Ángulo Ciego.
- Módulo Configuración.

Además, las clases necesarias para realizar la conexión mediante Bluetooth con el dispositivo Arduino.

2.8.3 MÓDULO VELOCIDAD

Este módulo consiste en determinar la velocidad a la que se mueve el vehículo utilizando la posición proporcionada por el sistema de ubicación del teléfono y calculando a diferentes intervalos de tiempo.

Para el funcionamiento de este módulo se establece lo siguiente:

- Los límites de velocidad por defecto asignados a la aplicación para emitir las alertas se establecen de acuerdo al reglamento de la ANT actual para vehículos livianos, estos son 50 Km/h, 90 Km/h y 100 Km/h.
- Las alertas se emitirán cada vez que se sobrepase uno de los límites indicados de la siguiente manera
    - Sobrepasar los 50 Km/h: se emitirá una vez la alerta sonora y se muestra en pantalla el límite que se excedió.
    - Sobrepasar los 90 Km/h: se emitirá dos veces la alerta sonora y se muestra en pantalla el límite que se excedió.
    - Sobrepasar los 100 Km/h: se emitirá de manera continua la alerta sonora y se muestra en pantalla el límite que se excedió.

- Adicionalmente la aplicación contará con un menú de configuración que permitirá modificar estos valores y a su vez activar/desactivar las diferentes alertas según lo requiera el usuario [39].

Para realizar las funciones detalladas anteriormente se utiliza el API Google *Location Services* llamada *Android.location*, la cual nos provee de las librerías y funciones necesarias para automáticamente obtener datos como movimiento, ubicación precisa, orientación geográfica, velocidad de desplazamiento del móvil, entre otras funciones.

Esta API permite mejorar el rendimiento de la batería, ya que el sistema determina la ubicación mediante torres de telefonía móvil, redes Wifi que se encuentren dentro del alcance del teléfono y sistema de satélites, de esta manera se tiene información en ambientes interiores y exteriores, reduciendo el consumo de batería [40].

El procedimiento que utiliza el dispositivo para obtener la ubicación más precisa se realiza como indica la figura 2.20.

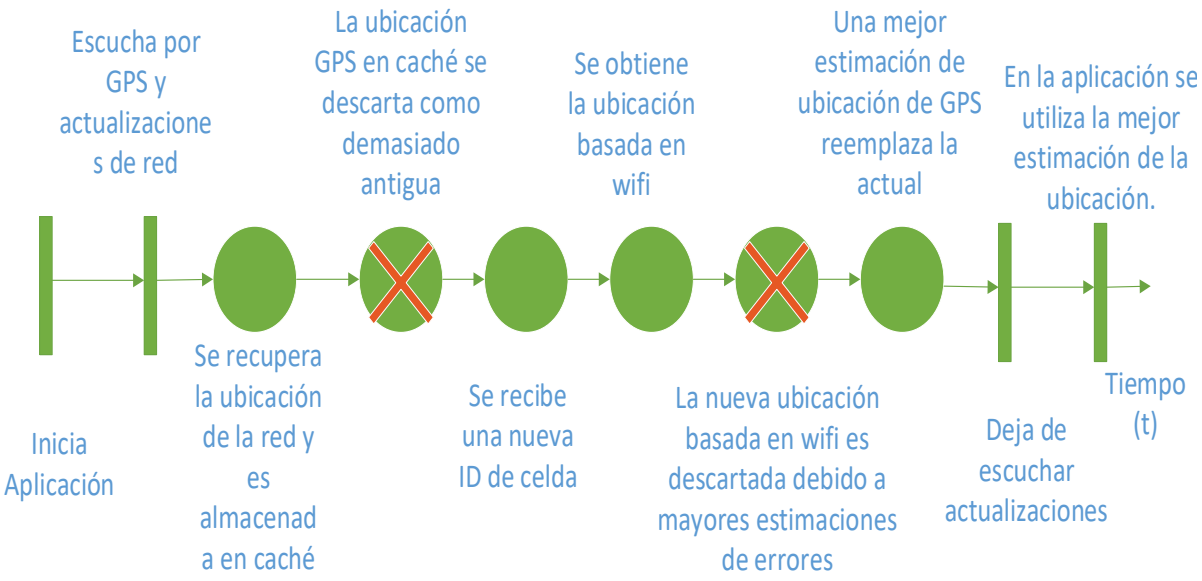


Figura 2.20 Diagrama de Flujo para obtener la ubicación del dispositivo.

Para hacer uso de Android.location es necesario que la aplicación disponga de los siguientes permisos:

- Recibir actualizaciones de ubicación: NETWORK_PROVIDER *o* GPS_PROVIDER

- Si se utiliza NETWORK_*PROVIDER y* GPS_PROVIDER, es necesario únicamente el permiso ACCESS_FINE_LOCATION el cual se incluye en el archivo AndroidManifest.xml. El permiso ACCESS_COARSE_LOCATION solo permite el acceso a la ubicación mediante la red de telefonía móvil.

De esta API se utilizarán las siguientes funciones:

2.8.3.1 getSpeed

Añadida en API *level* 1. Obtiene la velocidad en metros/segundo si se encuentra disponible, en caso de no estar disponible retornará 0.0.

Formato:

```
float getSpeed ()

Returns
Float
```

2.8.3.2 LocationListener

```
public interface LocationListener
android.location.LocationListener
```

Utilizada para recibir notificaciones desde el Administrador de Ubicaciones (**LocationManager**), cuando existe un cambio de ubicación. Estos métodos son llamados si el **LocationListener** ha sido registrado utilizando el servicio **requestLocationUpdates(String, long, float, LocationListener).**

Consta de los siguientes métodos para su implementación:

2.8.3.3 onLocationChanged

Añadido en API level 1

```
void onLocationChanged (Location location)
```

Llamado cuando la ubicación ha cambiado

- Parámetros:

location Location: La nueva ubicación, como un objeto Location.

2.8.3.4 onProviderDisabled

Añadido en API level 1

```
void onProviderDisabled (String provider)
```

Se llama cuando el proveedor es inhabilitado por el usuario. Si requestLocationUpdates llama a proveedor deshabilitado, este método se llama inmediatamente.

Parámetros

provider String: el nombre del proveedor de ubicación asociado a esta actualización.

2.8.3.5 onProviderEnabled

Añadido en API level 1

```
void onProviderEnabled (String provider)
```

Llamado cuando el proveedor es habilitado por el usuario

Parámetros

provider String: el nombre del proveedor de ubicación asociado a esta actualización.

2.8.3.6 onStatusChanged

Añadido en API level 1

```
void onStatusChanged (String provider, int status, Bundle extras)
```

Llamado cuando el proveedor cambia de estado. Este método se llama cuando un proveedor no puede obtener una ubicación o si el proveedor ha estado disponible recientemente después de un período de indisponibilidad.

Parámetros

provider	String: el nombre del proveedor de ubicación asociado a esta actualización.
status	int: OUT_OF_SERVICE si el proveedor está fuera de servicio y no se espera que cambie en un futuro próximo; TEMPORARILY_UNAVAILABLE si el proveedor no está disponible temporalmente, pero se espera que esté disponible en breve. Y AVAILABLE si el proveedor está actualmente disponible.
extras	Bundle: Un paquete opcional que contendrá variables de estado específicas del proveedor. Los proveedores que utilicen cualquiera de las claves de esta lista deben proporcionar el valor correspondiente al número de satélites utilizados para obtener la aproximación.

La declaración de la interfaz se puede observar en el código 2.9.

```
1.  import android.location.GpsStatus;
2.  import android.location.Location;
3.  import android.os.Bundle;
4.
5.  public interface LocationListener extends android.location.LocationListener, GpsStat
    us.Listener
6.  {
7.      public void onLocationChanged(Location location);
8.      public void onProviderDisabled(String provider);
9.      public void onProviderEnabled(String provider);
10.     public void onStatusChanged(String provider, int status, Bundle extras);
11.     public void onGpsStatusChanged(int event);
12. }
```

Código 2.9 Declaración de la interfaz para actualizar ubicación de GPS

2.8.3.7 Clase Ubicación.java

La clase Ubicación.java permite a través de los datos obtenidos mediante los valores de posicionamiento por GPS y su variación en el tiempo. Su declaración se indica en el código 2.10.

```
1.  import android.location.Location;
2.  public class Ubicacion extends Location{
3.      public Ubicacion(Location location)
4.      {
5.          super(location);
6.      }
```

Código 2.10 Declaración de la clase Ubicación.java

En el código 2.11 se calcula la velocidad del dispositivo a partir de los cambios de ubicación del teléfono.

La velocidad se obtiene en m/s, siendo necesario convertirla a Km/h (multiplicar por 3.6).

```
7.      @Override
8.      public float getSpeed()
9.      {
10.         float nSpeed = super.getSpeed()*3.6f;
11.         return nSpeed;
12.     }
13. }
```

Código 2.11 Función que calcula la velocidad a la que se mueve el vehículo

En el código 2.12 se establece los métodos necesarios para procesar la información de velocidad y alertas por sobrepasar los límites establecidos.

```
1.  public class Velocidad extends AppCompatActivity {
2.
3.      float velocidad =0;
4.      float[] limites = {50,90,100};
5.      public Velocidad() {
6.      public void setLimites(float[] limites) {
7.          this.limites = limites;
8.      }
```

Código 2.12 Declaración de clase Velocidad.java

El código 2.13 devuelve la velocidad actual como texto.

```
9.      public String updateSpeed(Ubicacion location) throws Exception
10.     {
11.         try {
12.             velocidad = 0;
13.             if (location != null) {
14.                 velocidad = location.getSpeed();
15.             }
16.             Formatter speed = new Formatter(new StringBuilder());
17.             speed.format(Locale.US, "%.0f", velocidad);
18.             String strCurrentSpeed = speed.toString();
19.         return strCurrentSpeed;
20.         }catch (Exception e) {
21.             Toast toast = Toast.makeText(this, e.getMessage(), Toast.LENGTH_SHORT);
22.             toast.show();
23.         }
24.         return "0";
25.     }
```

Código 2.13 Conversión a texto del valor de velocidad obtenido mediante GPS

El código 2.14 devuelve el valor de la alerta detectada que sobrepasa el límite de velocidad.

```
26.     public String updateAlerta(String valorVelocidad){
27.
28.         velocidad = Float.parseFloat(valorVelocidad);
29.
30.         float valorAlerta=0;
31.         if(velocidad<limites[0]){
32.             valorAlerta=0;
33.
34.         }else if (velocidad >= limites[0] && velocidad <limites[0]+10){
35.             valorAlerta = limites[0];
36.
37.         } else if(velocidad >= limites[1] && velocidad <limites[1]+10) {
38.             valorAlerta = limites[1];
39.         }else if(velocidad >= limites[2] ){
40.             valorAlerta = limites[2];
41.         }
42.
43.         Formatter alerta = new Formatter(new StringBuilder());
```

```
44.         alerta.format(Locale.US, "%5.0f", valorAlerta);
45.         String strAlerta = alerta.toString();
46.         return strAlerta;
47.     }
48. }
```

Código 2.14 Detección de alertas a partir de la velocidad.

2.8.4 MÓDULO ÁNGULO CIEGO

Para la detección del ángulo ciego se desarrollará las clases de conexión mediante bluetooth y el procesamiento de la información recibida desde el dispositivo Arduino para emitir la alerta en caso de la detección de un objeto en el umbral del ángulo ciego.

Se establecen las siguientes funciones:

- El sistema mostrará alertas visuales cada vez que un objeto ingrese a la zona de Angulo ciego.
- Las alertas sonoras se emitirán cuando un objeto se encuentre en el ángulo ciego y esté activada la luz intermitente de cambio de carril respectiva a ese sensor.
- La conexión para obtener los datos se realizará a través de Bluetooth hacia la placa arduino, este proceso implica:
    - Búsqueda de los dispositivos Bluetooth conectados.
    - Selección del dispositivo.
    - Establecimiento de la conexión.
    - Transmisión de datos.
    - Cierre de la conexión.

2.8.4.1 Clase BuscarBT.java

Esta clase realiza una búsqueda de todos los dispositivos Bluetooth cercanos y que se encuentren en modo visible.

Para buscar los dispositivos Bluetooth disponibles para realizar la conexión se detalla en el código 2.15, 2,16, 2,17. Este código consiste en iniciar la búsqueda de los dispositivos emparejados, agregar a la lista los nuevos dispositivos encontrados y emparejar con el dispositivo Bluetooth seleccionado.

```
1.      @Override
2.      protected void onCreate(Bundle savedInstanceState) {
3.          super.onCreate(savedInstanceState);
4.          setContentView(R.layout.scan);
5.
6.          lvBluetooth = (ListView)findViewById(R.id.lvLista);
7.          tvEstado = (TextView) findViewById(R.id.tvEstado);
8.          pbActualizar = (ProgressBar) findViewById(R.id.pbScan);
9.          btBuscar = (Button) findViewById(R.id.btnRepetirBusqueda);
```

Código 2.15 Inicializa los controles de la interfaz de búsqueda de Bluetooth

```
10.     @Override
11.     public void onDevice(final BluetoothDevice dispositivo) {
12.         final BluetoothDevice temp = dispositivo;
13.         listaBluetooth.add(dispositivo);
14.
15.         runOnUiThread(new Runnable() {
16.             @Override
17.             public void run() {
18.                 adaptador.add(temp.getAddress()+" - "+temp.getName());
19.             }
20.         });
21.     }
```

Código 2.16 Permite interactuar con los datos de una lista en una vista

```
22.     @Override
23.     public void onPair(BluetoothDevice dispositivo) {
24.         visibilidadProgressBar(View.INVISIBLE);
25.         estadoBluetooth("Emparejado");
26.         Intent i = new Intent(BuscarBT.this, SeleccionarBT.class);
27.         startActivity(i);
28.         finish();
29.     }
30.
```

```
31.     @Override
32.     public void onUnpair(BluetoothDevice dispositivo) {
33.         visibilidadProgressBar(View.INVISIBLE);
34.         estadoBluetooth("Emparejado!");
35.     }
```

Código 2.17 Proceso para emparejar el dispositivo Bluetooth con el teléfono

2.8.4.2 Clase SeleccionarBT.java

Esta clase permite seleccionar uno de los dispositivos previamente recocidos por la clase BuscarBT.java y establece la conexión con el dispositivo.

De acuerdo a las API de Google y los ejemplos que proporciona Android Studio el código 2.18 actualiza la lista de dispositivos Bluetooth disponibles.

```
1.      @Override
2.      public void onRefresh() {
3.          List<String> names = new ArrayList<String>();
4.          for (BluetoothDevice d : bluetooth.getPairedDevices()){
5.              names.add(d.getName());
6.          }
7.          String[] array = names.toArray(new String[names.size()]);
8.          final ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
9.                  android.R.layout.simple_list_item_1, android.R.id.text1, array);
10.        this.runOnUiThread(new Runnable() {
11.             @Override
12.             public void run() {
13.                 lvListaBT.removeViews(0, lvListaBT.getCount());
14.                 lvListaBT.setAdapter(adapter);
15.                 listaBluetooth = bluetooth.getPairedDevices();
16.             }
17.         });
18.         actualizar.refreshComplete();   }
```

Código 2.18 Actualiza la lista de dispositivos emparejados

El código 2.19 añade a la lista los dispositivos emparejados con el teléfono

```
19.     private void addDevicesToList(){
20.         listaBluetooth = bluetooth.getPairedDevices();
```

```
21.         List<String> names = new ArrayList<>();
22.         for (BluetoothDevice d : listaBluetooth){
23.             names.add(d.getName());
24.         }
25.
26.         String[] array = names.toArray(new String[names.size()]);
27.
28.         ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
29.                 android.R.layout.simple_list_item_1, android.R.id.text1, array);
30.
31.         lvListaBT.setAdapter(adapter);
32.         btnBuscarBluetooth.setEnabled(true);
33.     }
```

Código 2.19 Añade un nuevo dispositivo Bluetooth

Una vez realizada la conexión Bluetooth es necesario verificar si ocurren cambios en el estado de la conexión con el dispositivo emparejado, como se muestra en el código 2.20, el método implementado permite detectar si se encuentra en estado apagado, encendido o ha ocurrido un error en la conexión.

```
34.     private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
35.         @Override
36.         public void onReceive(Context context, Intent intent) {
37.             final String action = intent.getAction();
38.
39.         if (action.equals(BluetoothAdapter.ACTION_STATE_CHANGED)) {
40.
41.             final int state = intent.getIntExtra(BluetoothAdapter.EXTRA_STATE,
42.                     BluetoothAdapter.ERROR);
43.
44.             switch (state) {
45.                 case BluetoothAdapter.STATE_OFF:
46.                     runOnUiThread(new Runnable() {
47.                         @Override
48.                         public void run() {
49.                             lvListaBT.setEnabled(false);
50.                         }
51.                     });
52.                     Toast.makeText(SeleccionarBT.this, "Encender Bluetooth",
53.                             Toast.LENGTH_LONG).show();
54.                     break;
```

```
                case BluetoothAdapter.STATE_ON:
                    runOnUiThread(new Runnable() {
                        @Override
                        public void run() {
                            addDevicesToList();
                            lvListaBT.setEnabled(true);
                        }
                    });
                    break;
            }
        }
        }
    };
}
```

Código 2.20 Verifica el estado de la conexión Bluetooth

2.8.4.3 Clase AnguloCiego.java

Esta clase procesa la información enviada por el dispositivo Arduino y detecta si se debe habilitar la alerta por un objeto en el ángulo ciego. La declaración de la clase AnguloCiego.java se puede revisar en el código 2.21, 2.22 la cual consiste en variables para almacenar la distancia enviada por los sensores y variables lógicas para establecer el estado de activación del sensor.

```
public class AnguloCiego {

    public AnguloCiego(boolean sensorIzquierdo, boolean sensorDerecho, float valorDi
stanciaIzquierdo, float valorDistanciaDerecho) {
        this.sensorIzquierdo = sensorIzquierdo;
        this.sensorDerecho = sensorDerecho;
        this.valorDistanciaIzquierdo = valorDistanciaIzquierdo;
        this.valorDistanciaDerecho = valorDistanciaDerecho;
    }
```

Código 2.21 Declaración de la clase AnguloCiego y Constructor

El código 2.22 permite asignar y modificar valores a un objeto AnguloCiego

```
    public void setSensorIzquierdo(boolean sensorIzquierdo) {
        this.sensorIzquierdo = sensorIzquierdo;
    }
```

```
12.    public void setSensorDerecho(boolean sensorDerecho) {
13.        this.sensorDerecho = sensorDerecho;
14.    }
15.
16.    public void setValorDistanciaIzquierdo(float valorDistanciaIzquierdo) {
17.        this.valorDistanciaIzquierdo = valorDistanciaIzquierdo;
18.    }
19.
20.    public void setValorDistanciaDerecho(float valorDistanciaDerecho) {
21.        this.valorDistanciaDerecho = valorDistanciaDerecho;
22.    }
23. }
```

Código 2.22 Accesorios para asignar valores a un objeto AnguloCiego

2.8.5 MÓDULO CONFIGURACIÓN

Consiste en un menú que permita seleccionar un rango de valores para activar las alertas por sobrepasar el límite de velocidad.

2.8.5.1 Clase ConfiguraciónActivity.java

La aplicación permitirá al usuario modificar el valor en el cual detecte el límite y dispare la alerta, en el código 2.23 se implementa el código necesario para la interfaz de configuración.

```
1.  public class ConfiguracionActivity extends Activity {
2.      @Override
3.      protected void onCreate(Bundle savedInstanceState) {
4.          super.onCreate(savedInstanceState);
5.          getFragmentManager().beginTransaction()
6.                  .replace(android.R.id.content, new ConfiguracionFragment())
7.                  .commit();
8.      }
9.  }
```

Código 2.23 Inicializa el código para modificar la Configuración de la actividad.

2.8.5.2 Clase ConfiguracionFragment.java

Esta clase permite guardar los cambios las configuraciones de la aplicación, se indica en el código 2.4

```
24. public class ConfiguracionFragment extends PreferenceFragment {
25.
26.     @Override
27.     public void onCreate(Bundle savedInstanceState) {
28.         super.onCreate(savedInstanceState);
29.         addPreferencesFromResource(R.xml.settings);
30.     }
31. }
```

Código 2.24 Inicialización del Fragment para guardar las configuraciones

2.8.6 CLASE PRINCIPAL.JAVA

La clase Principal.java inicia la actividad invocando los métodos específicos para el funcionamiento de la interfaz que se muestra al usuario.

La inicialización de las variables se indica en el código 2.25 al código 2.

- La inicialización de las imágenes para mostrar el ángulo ciego se configura en negro para el sensor derecho e izquierdo. Ver código 2.25.

```
1.      @Override
2.      protected void onCreate(Bundle savedInstanceState) {
3.          super.onCreate(savedInstanceState);
4.          setContentView(R.layout.activity_main);
5.
6.          imgSensorIzquierdo=(ImageView)findViewById(R.id.imageIzquierda);
7.          imgSensorIzquierdo.setBackgroundColor(Color.BLACK);
8.
9.          imgSensorDerecho=(ImageView)findViewById(R.id.imageDerecha);
10.         imgSensorDerecho.setBackgroundColor(Color.BLACK);
11.         tvLogs = (TextView)findViewById(R.id.tvLogs);
```

Código 2.25 Inicialización de las variables para ángulo ciego

- Se activa la comunicación Bluetooth previamente establecida como se indica en el código 2.26, donde se obtiene el nombre del dispositivo y se inicia la recepción de mensajes desde el módulo Bluetooth conectado al Arduino.

```
12.         bluetooth = new Bluetooth(this);
13.         bluetooth.enableBluetooth();
14.         bluetooth.setCommunicationCallback(this);
15.
```

```
16.         int pos = getIntent().getExtras().getInt("pos");
17.         nombreDispositivo = bluetooth.getPairedDevices().get(pos).getName();
18.
19.         MensajesBT("Conectando...");
20.         bluetooth.connectToDevice(bluetooth.getPairedDevices().get(pos));
21.         IntentFilter filter = new IntentFilter(BluetoothAdapter.ACTION_STATE_CHANGED
    );
22.         registerReceiver(RecibirBT, filter);
23.         registrado =true;
```

Código 2.26 Comunicación Bluetooth

- Las verificaciones de los permisos para acceder a la ubicación del dispositivo se verifican al iniciar la aplicación como se indica en el código 2.27, estos permisos también deben añadirse en el archivo AndroidManifest.xml.

```
24. LocationManager locationManager = (LocationManager)
25.                 this.getSystemService(Context.LOCATION_SERVICE);
26.
27.         if (ActivityCompat.checkSelfPermission(
28.                 this, Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PE
    RMISSION_GRANTED
29.                 && ActivityCompat.checkSelfPermission(
30.                         this,
31.                 Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMIS
    SION_GRANTED) {
32.
33.             return;
34.         }
```

Código 2.27 Permisos para acceder a la ubicación del dispositivo

- Carga del archivo de sonido para emitir la alerta. Ver código 2.28

```
33.         mediaPlayer = MediaPlayer.create(this, R.raw.alert1);
34.         TextView textAlerta = (TextView) this.findViewById(R.id.tvAlerta);
35.         textAlerta.setVisibility(View.INVISIBLE);
```

Código 2.28 Archivo de sonido de alerta

La actualización de la alerta de ángulo ciego consiste en verificar que la información enviada por los sensores se encuentre dentro del rango establecido a partir de las pruebas, si coincide este rango se cambia la imagen de color negro por una de color naranja en el lado del sensor que detecta el objeto. Ver código 2.29.

```
36.     public void MensajesBT(final String mensaje){
37.         this.runOnUiThread(new Runnable() {
38.             @Override
39.             public void run() {
40.      //           tvLogs.setText(mensaje);
41.                 tvLogs.setText("Sensores: " + valorSensorDerecho + " " + valorSensor
    Izquierdo
42.                         + " Limites:"
43.                         + " " + velocidad.limites[0]
44.                         + " " + velocidad.limites[1]
45.                         + " " + velocidad.limites[2]);
46.
47.                 if(valorSensorIzquierdo<150 && valorSensorIzquierdo>90) {
48.                    contadorIzquierdo++;
49.                    if (contadorIzquierdo>=3){
50.                         imgSensorIzquierdo.setBackgroundResource(R.drawable.relleno_
    degradado2);
51.                     }
52.                 }else{
53.                     imgSensorIzquierdo.setBackgroundColor(Color.BLACK);
54.                     contadorIzquierdo=0;
55.                 }
56.                 if(valorSensorDerecho<150 && valorSensorDerecho>90) {
57.                    contadorDerecho++;
58.
59.                     if (contadorDerecho>=3){
60.                         imgSensorDerecho.setBackgroundResource(R.drawable.relleno_de
    gradado);
61.                     }
62.                }else{
63.                     imgSensorDerecho.setBackgroundColor(Color.BLACK);
64.                     contadorDerecho=0;
65.                 }
66.             }
67.         });
68.     }
```

Código 2.29 Actualización de sensores de ángulo ciego

La función que actualiza la velocidad se activa cada vez que ocurre un cambio de posición del vehículo, con este valor se verifica si se encuentra dentro de los límites

establecidos y emite la alerta visual y sonora de ser el caso. Este procedimiento se indica en los códigos 2.30, 2.31 y 2.32.

```
69.     public void updateSpeed(Ubicacion ubicacion) throws Exception
70.     {
71.         try {
72.             velocidadStr = velocidad.updateSpeed(ubicacion);
73.             float miVelocidadValor= Float.parseFloat(velocidadStr);
74.             TextView txtCurrentSpeed3 = (TextView) this.findViewById(R.id.tvVelocida
    dValor);
75.             txtCurrentSpeed3.setText(velocidadStr);
76.             String strAlerta = velocidad.updateAlerta(velocidadStr);
77.            alertaVelocidad(miVelocidadValor);
78.         }catch (Exception e) {
79.             Toast toast = Toast.makeText(this, e.getMessage(), Toast.LENGTH_SHORT);

80.             toast.show();
81.         }
82.     }
```

Código 2.30 Actualización de la velocidad de circulación del vehículo

```
83.     public void alertaVelocidad(float valorVelocidad){
84.
85.         TextView txtAlerta = (TextView) this.findViewById(R.id.tvAlerta);
86.         if(valorVelocidad>= velocidad.limites[0]-
    3 && valorVelocidad<= velocidad.limites[0]+9
87.                 && alerta==true){
88.            txtAlerta.setText(vCiudad);
89.             txtAlerta.setVisibility(View.VISIBLE);
90.             sonidoAlerta();
91.             alerta=false;
92.
93.         } else if(valorVelocidad>= velocidad.limites[1]-3
94.                 && valorVelocidad<= velocidad.limites[1]+9 && alerta==true){
95.
96.             txtAlerta.setText(vPerimetral);
97.             txtAlerta.setVisibility(View.VISIBLE);
98.             sonidoAlerta();
99.             alerta=false;
100.
```

Código 2.31 Verifica si se sobrepasó el límite de circulación en ciudad

```
101.      public void sonidoAlerta()
102.      {
103.          if(mediaPlayer.isPlaying()==false){
104.
105.              if (mediaPlayer != null) {
106.                  mediaPlayer.release();
107.              }
108.             mediaPlayer = MediaPlayer.create(this, R.raw.alert1);
109.              mediaPlayer.start();
110.          }
111.      }
112.
```

Código 2.32 Función que emite la alerta sonora

CAPÍTULO 3 – IMPLEMENTACIÓN DEL PROTOTIPO

3.1 IMPLEMENTACIÓN DEL PROTOTIPO

En este capítulo se explica la instalación de software en las dos plataformas utilizadas: Android y Arduino de tal manera que los distintos componentes del prototipo se comuniquen entre sí, entre las conexiones se implementará: el envío de datos entre los sensores y Arduino.

La comunicación Bluetooth entre Arduino y Android y el procesamiento de la información recibida para ser mostrada al usuario, además la configuración del hardware y la instalación eléctrica en un vehículo para realizar las pruebas necesarias de funcionamiento.

3.2 REQUERIMIENTOS PARA LA IMPLEMENTACIÓN DEL PROTOTIPO

Las funcionalidades y características del prototipo son las siguientes:

- Detectar objetos en el ángulo ciego.
- Determinar la velocidad de circulación del vehículo.
- Alertar al conductor cuando se sobrepase la velocidad máxima establecida en la aplicación.

3.3 HERRAMIENTAS UTILIZADAS PARA EL DESARROLLO

3.3.1 ARDUINO IDE 1.8.3

El Software Arduino (IDE) permite escribir programas/*scripts* y subir el código compilado a la tarjeta de esta manera modificar el funcionamiento del Arduino.

3.3.1.1 INSTALACIÓN DEL SOFTWARE ARDUINO IDE

Se recomienda descargar la última versión del software disponible de la página del fabricante en el cual está disponible para los siguientes Sistemas Operativos: Windows, Linux, Mac OsX.

- Una vez descargado, ejecutar el instalador.

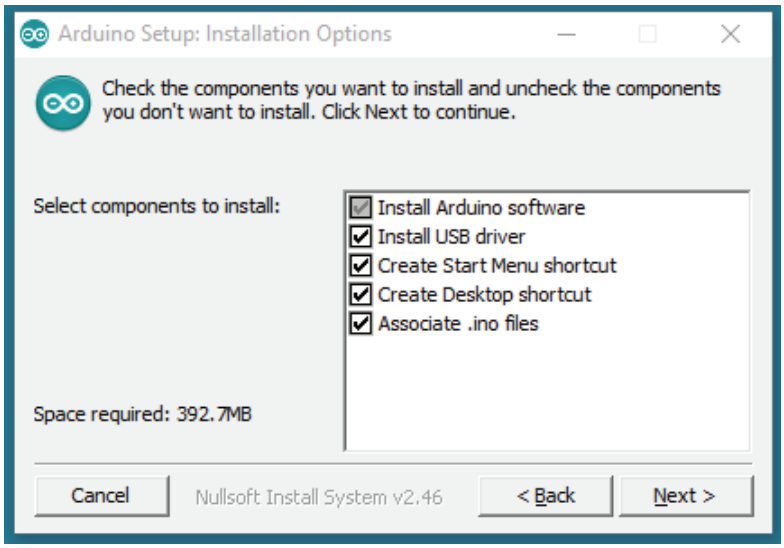


Figura 3.1 Instalación Arduino IDE - Componentes

- Seleccionar los componentes a instalar.

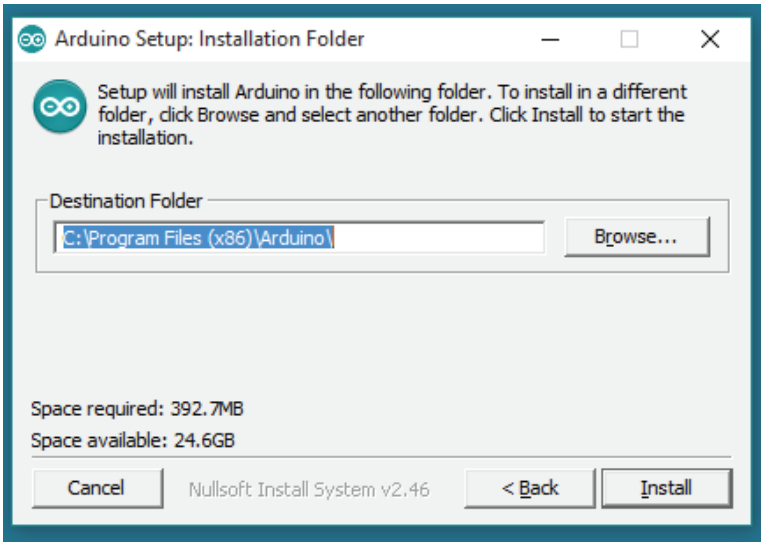


Figura 3.2 Instalación Arduino IDE - Ubicación de la instalación

- Seleccionar el directorio de instalación (Recomendación: seleccionar el directorio por defecto)

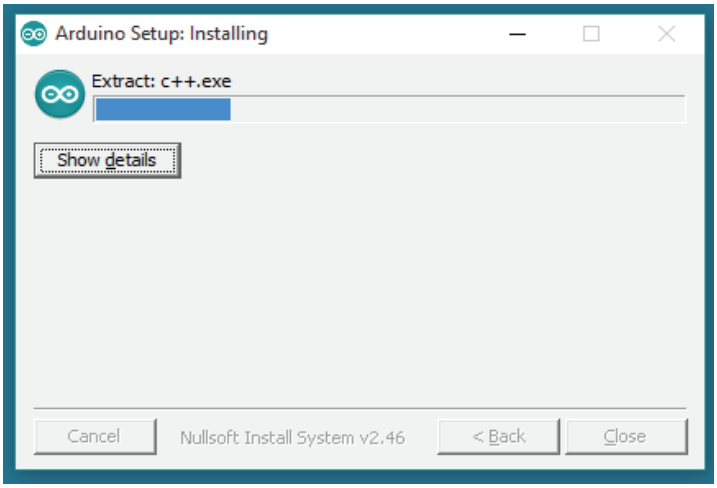
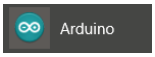


Figura 3.3 Instalación Arduino IDE - Instalación

- El instalador, extraerá e instalará los archivos necesarios para el correcto funcionamiento de programa.
- Una vez instalado ejecutar el programa 

- El siguiente paso es seleccionar la placa a utilizar desde el menú **Herramientas > Placa> [Seleccionar placa].**

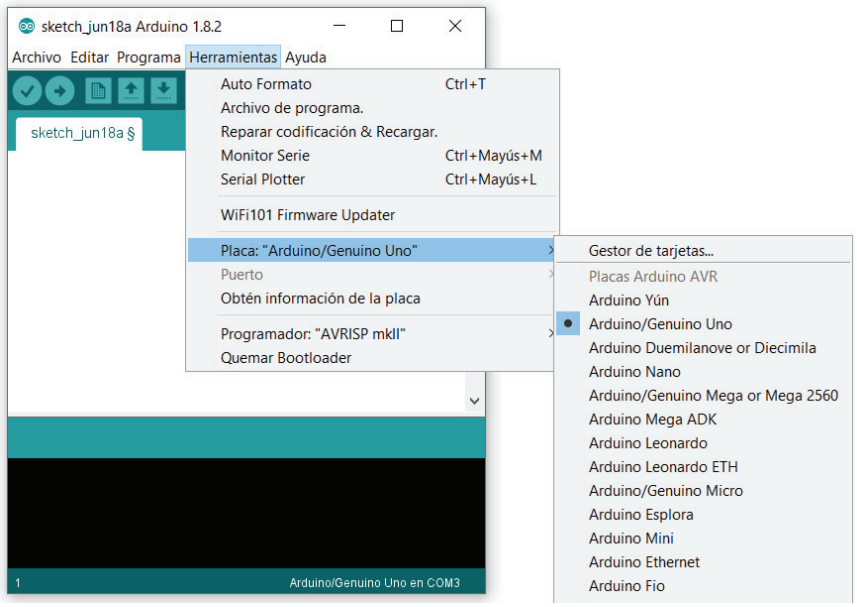


Figura 3.4 Configuración de la placa Arduino

3.3.1.2 INSTALACIÓN DE LIBRERIAS EN ARDUINO IDE

Para el desarrollo de scripts, en ocasiones es necesario la instalación de las librerías que provee el fabricante de sensores o incluidas en el SDK DE ARDUINO.

Para instalar descargar la librería desde la página del fabricante. Es un archivo comprimido de extensión **.zip**.

Una vez descargado el archivo, se debe agregar al IDE la librería desde el menú **Programa > Incluir Librería > Añadir librería .zip.** Ver Figura 3.5.

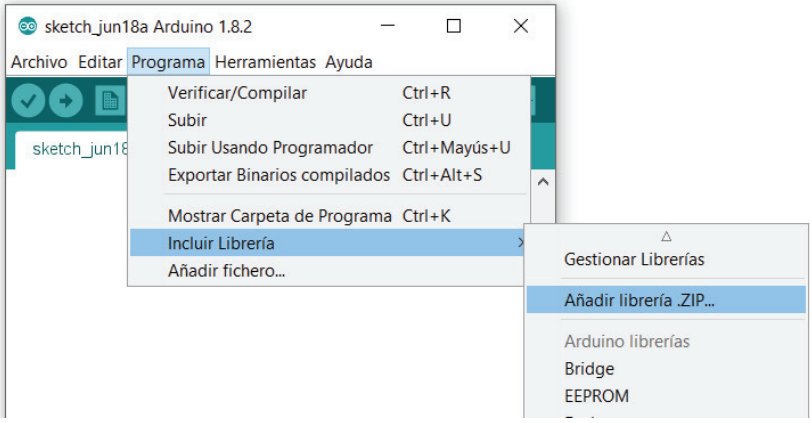


Figura 3.5 Instalación de Librería Arduino

Abrir el archivo que contiene la librería **.zip.** Como indica la figura 3.6.

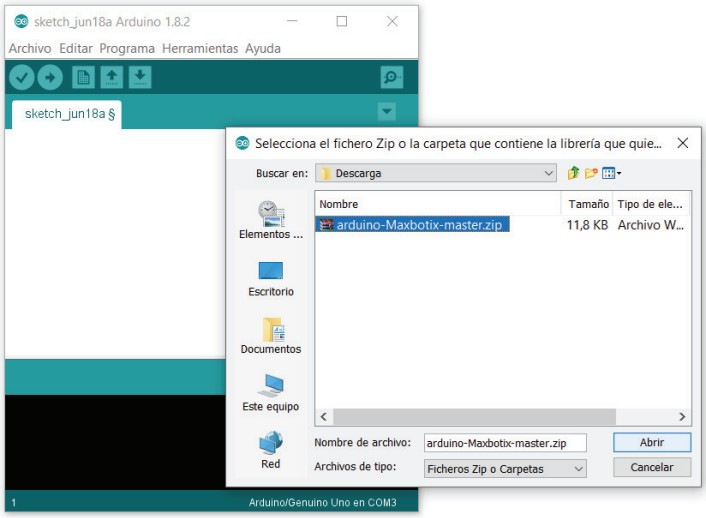


Figura 3.6 Instalación de librería Arduino - Ubicación de archivo

Para indicar al IDE que se va a utilizar la librería instalada, se utilizará el siguiente código

```
#include "Maxbotix.h"
```

3.3.2 ANDROID STUDIO

Para la programación de la aplicación se utilizará el software de desarrollo Android Studio en su versión 2.2.3. Este IDE (*Integrated Development Environment*) desarrollado por Google permite compilar el código y acceder a las diferentes librerías que provee Google para controlar los diferentes recursos de hardware y software que dispone el sistema Android, además dispone de un emulador mediante software que nos da la posibilidad de compilar el código y simular su ejecución.

3.3.2.1 INSTALACIÓN DE ANDROID STUDIO

Antes de la instalación es necesario tener instalado el software de Java JDK 7 o versiones superiores. Una vez verificado que tenemos instalado Java se procede a descargar el instalador de Android Studio desde la página web de *Android Developers* (Figura 3.7)

Figura 3.7 Página web de *Android Developers* para descargar Android Studio

Se ejecuta el código que se ha descargado y se continúa el asistente de instalación hasta completarlo. Ver figura 3.8.

Figura 3.8 Asistente de instalación de Android Studio

3.4 IMPLEMENTACIÓN DEL SOFTWARE EN ARDUINO

Para la implementación del software en Arduino es necesario realizar las conexiones que se indican en la tabla 3.1 y figura 3.9.

Tabla 3.1 Descripción de pines de conexión entre el sensor ultrasónico y Arduino

SENSOR 1	**ARDUINO**
PIN 7 – GND	GND (Ground)
PIN 6 – +5 V Vcc	VCC 5 V
PIN 5 – Tx	Sin conexión
PIN 4 – Rx	PIN 5 Tx Sensor 2
PIN 3 – AN	Sin Conexión
PIN 2 – PW	PIN 5 SALIDA DIGITAL PWM
PIN 1 – BW	VCC 5 V

SENSOR 2	**ARDUINO**
PIN 7 – GND	GND (Ground)
PIN 6 – +5 V Vcc	VCC 5 V
PIN 5 – Tx	PIN 4 Rx Sensor 1
PIN 4 – Rx	PIN 2
PIN 3 – AN	Sin Conexión
PIN 2 – PW	PIN 3 SALIDA DIGITAL PWM
PIN 1 – BW	VCC 5 V

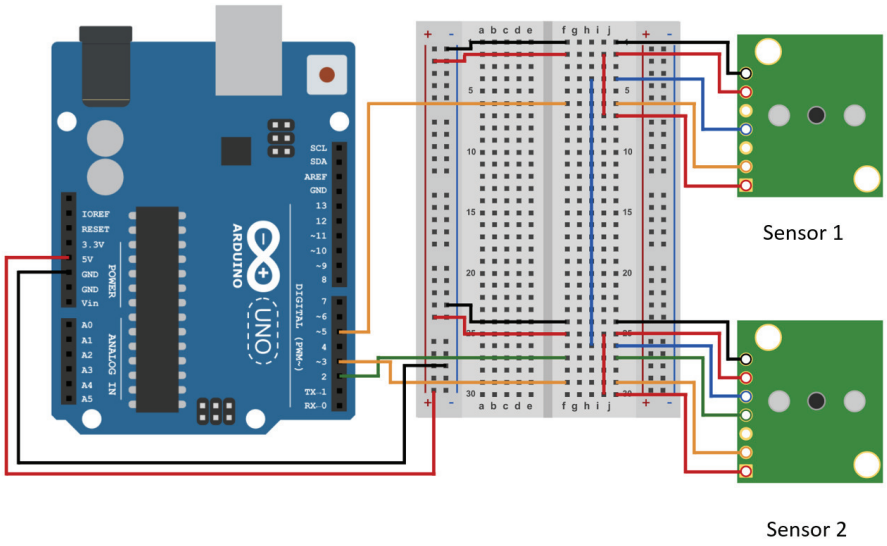


Figura 3.9 Conexiones entre sensor y Arduino.

3.4.1 IMPLENTACIÓN DEL CÓDIGO

El código que permitirá controlar y obtener la información de los sensores será adecuado a partir del *datasheet* del fabricante Maxbotix. Siguiendo las recomendaciones de implementación como se indica en el código 3.1

```
/*
Código sensores ultrasónicos Maxbotix
Medida en cm
Formato: PW
*/
const int pwPin1 = 3;
const int pwPin2 = 5;

const int triggerPin1 = 13;
long pulse1, pulse2, sensor1, sensor2;

void setup () {
  Serial.begin(9600);
  pinMode(pwPin1, INPUT);
  pinMode(pwPin2, INPUT);

  pinMode(triggerPin1,OUTPUT);
}

void read_sensor1(){
 pulse1 = pulseIn(pwPin1, HIGH);
 pulse2 = pulseIn(pwPin2, HIGH);

 sensor1 = pulse1/147*2.54;
 sensor2 = pulse2/147*2.54;
}
```

```
void start_sensor1(){
  digitalWrite(triggerPin1,HIGH);
  delay(1);
  digitalWrite(triggerPin1,LOW);
}

void printall(){
  Serial.print("S1");
  Serial.print(":");
  Serial.print(sensor1);
  Serial.print(":");
  Serial.print("S2");
  Serial.print(":");
  Serial.print(sensor2);
  Serial.println(" ");
}

void loop () {
  start_sensor1();
  read_sensor1();

  printall();
  delay(500);
}
```

Código 3.1 Programación de la placa Arduino

3.5 IMPLEMENTACIÓN DEL SOFTWARE EN ANDROID

La implementación del código en Android consiste en desarrollar una aplicación con interfaz para el usuario que permita visualizar la velocidad y objetos cercanos en la zona del ángulo ciego del vehículo, también alertar mediante sonidos al sobrepasar los límites establecidos en la aplicación

3.5.1 INTERFAZ DE LA APLICACIÓN EN ANDROID

La interfaz del proyecto permitirá una vista al usuario con la información de velocidad y alertas máximas, además conocer el estado de los sensores en caso de activarse por objetos en el ángulo ciego.

Adicionalmente dispondrá de una vista de configuración para establecer, activar o desactivar los tres diferentes límites de velocidad referentes a ciudad, vías perimetrales y carretera, por defecto los valores serán 50 Km/h (Ciudad), 90 Km/h (vías perimetrales) y 100 Km/h (carretera), también, en la vista configuración la

aplicación permitirá activar/desactivar las alertas sonoras y la detección del ángulo ciego.

Se describirá las funciones de las vistas de la aplicación.

3.5.1.1 VISTA PRINCIPAL:

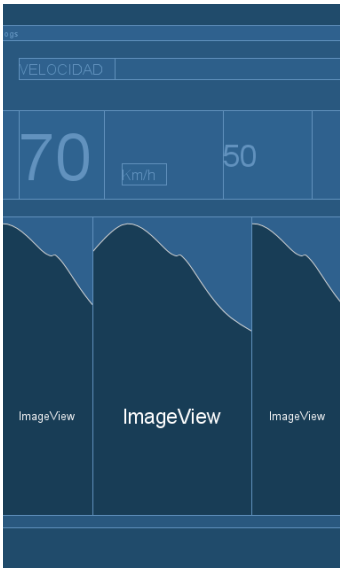


Figura 3.10 Vista principal de la Aplicación

En las figuras 3.10 y 3.11, la interfaz de usuario se divide en dos secciones:

- La parte superior, corresponde a la detección de velocidad indicando el valor en Km/h a la que se mueve el vehículo y las alertas que se sobrepasen del límite establecido
- La parte inferior indicará que lado del vehículo detecta objetos en el ángulo ciego mientras está en circulación.

3.5.1.2 VISTA DE CONFIGURACIÓN

La vista de configuración permitirá seleccionar si el usuario desea que la alerta de exceso de velocidad se active antes o después de que el velocímetro marque el valor establecido. La figura 3.12 indica cómo se muestra al usuario la configuración de la aplicación.

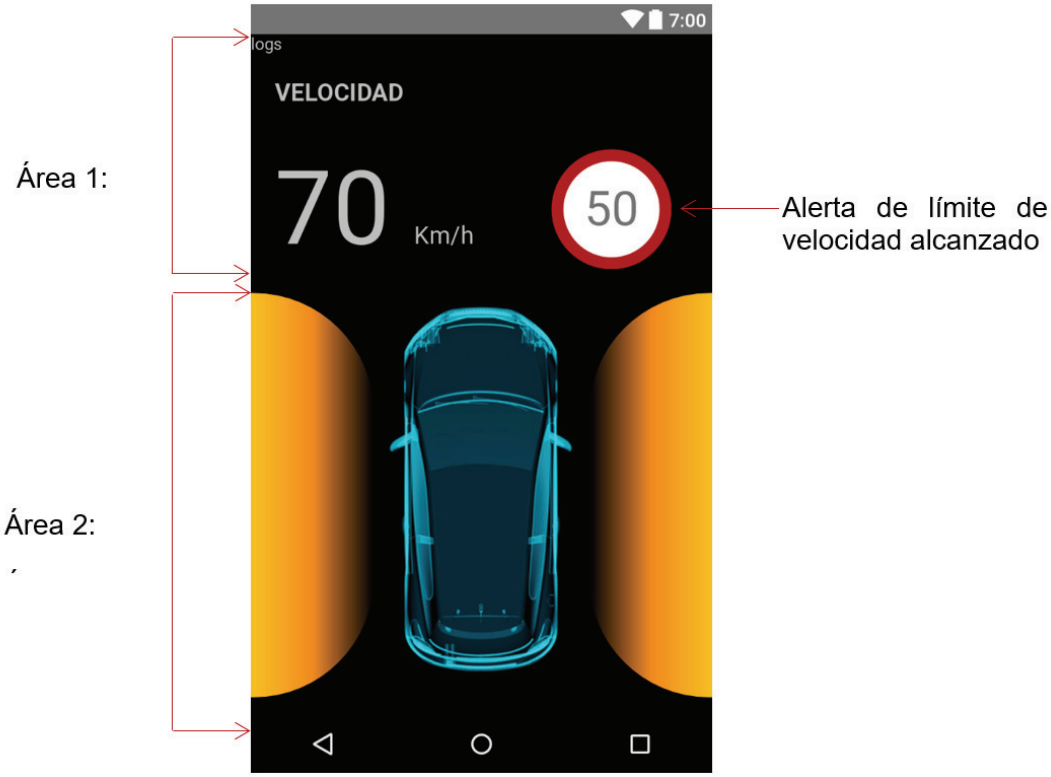


Figura 3.10 Vista principal de la aplicación en Android

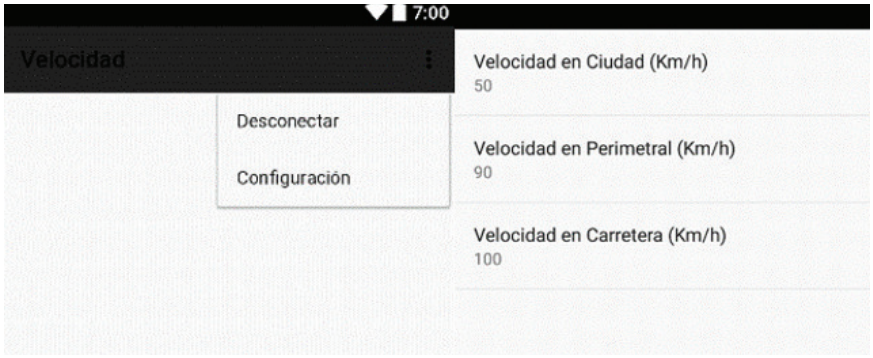


Figura 3.11 Opciones de la vista de configuración

3.5.1.3 PROTOTIPO ARDUINO

Las pruebas iniciales se realizaron en un *protoboard* como se puede ver en las figuras 3.13 y 3.14 con los componentes necesarios para el funcionamiento del prototipo como son: dos sensores ultrasónicos, el módulo Bluetooth y la placa Arduino.

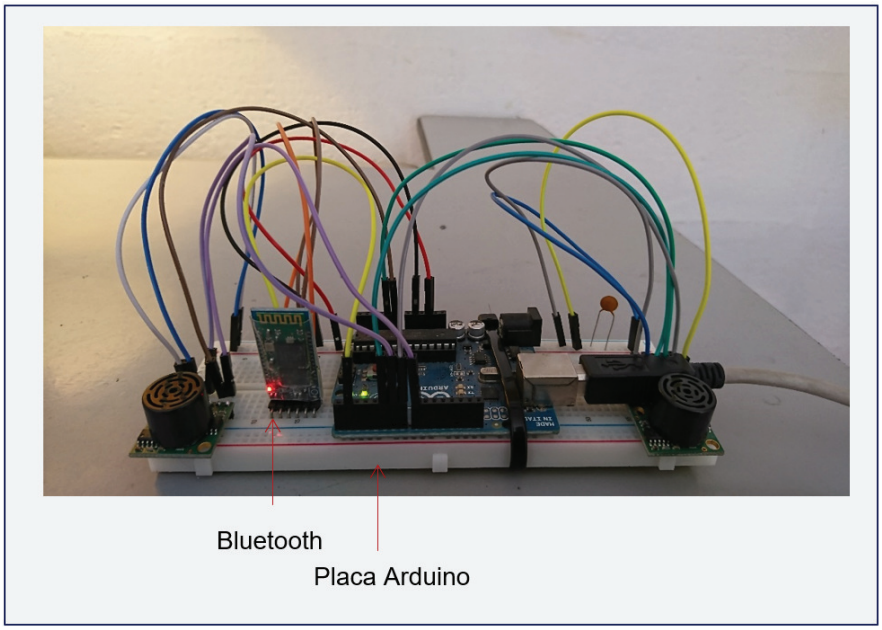


Figura 3.12 Prototipo para pruebas de funcionamiento

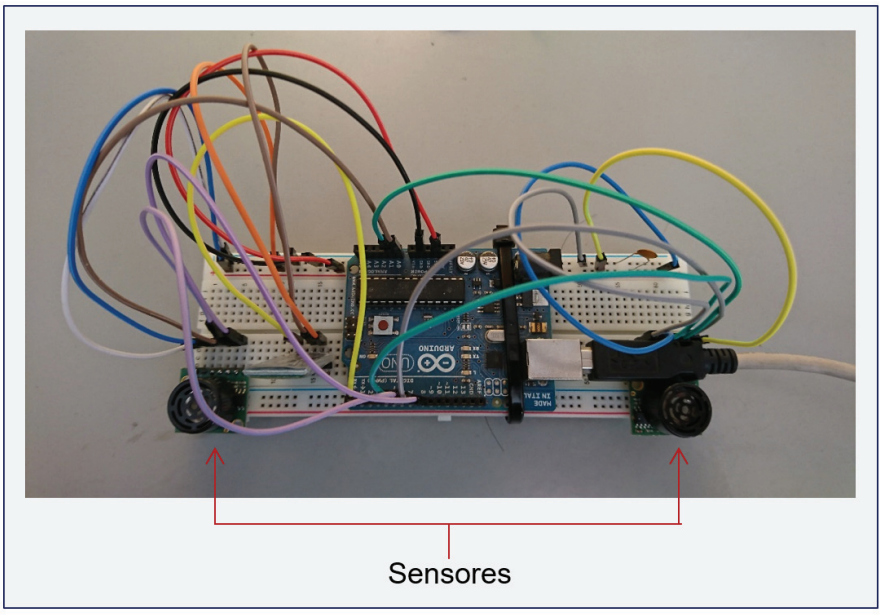


Figura 3.13 Prototipo de pruebas en funcionamiento

3.6 INSTALACIÓN EN EL VEHÍCULO

El proceso de conexión eléctrica en el vehículo permitirá que se active el dispositivo al momento de encender el auto, para este proceso el dispositivo Arduino se

conectará a la salida de 12 VDC mediante un regulador de voltaje de 5 VDC para alimentar la placa Arduino, el transmisor Bluetooth y los sensores.

3.6.1 UBICACIÓN DE SENSORES

Los sensores se ubicarán en los laterales del vehículo en la parte trasera como se muestra en la figura 3.15.

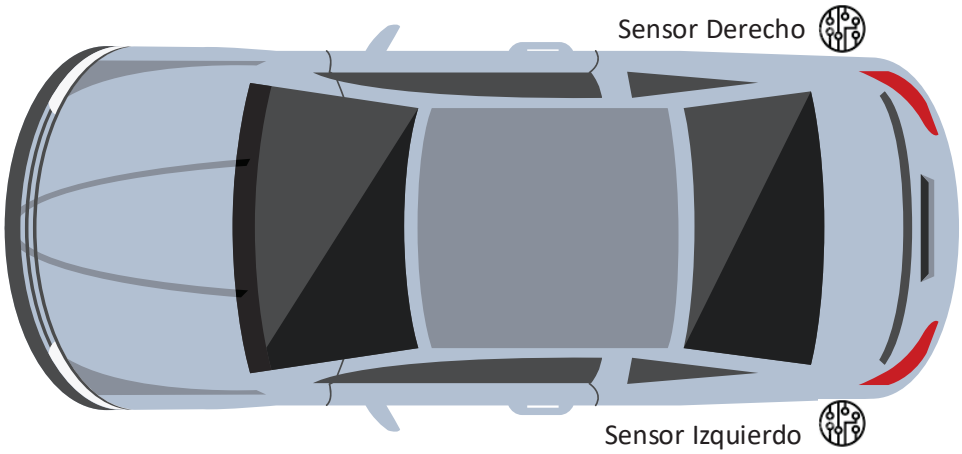


Figura 3.14 Ubicación de los sensores en el vehículo.

Figura 3.15 Ubicación física de los sensores ultrasónicos

En la figura 3.16 se puede observar los sensores ubicados en los laterales del vehículo de tal manera que el foco de detección de los sensores cubra el área del ángulo ciego del vehículo.

CAPÍTULO 4 - PRUEBAS DEL PROTOTIPO

En este capítulo se describe el correcto funcionamiento de la aplicación por medio de la realización de pruebas al vehículo.

Las pruebas irán enfocadas al análisis de la correcta medición de velocidad entre el velocímetro del vehículo y el valor mostrado en la aplicación utilizando el GPS; la activación de las alertas del límite de velocidad excedido de los valores previamente configurados y la detección de objetos que se ubican en el ángulo ciego del vehículo en circulación.

Además, en este capítulo se describirán las conclusiones y recomendaciones de este proyecto, las cuales permitirán determinar de mejor manera los procedimientos y métodos a utilizar con el fin de obtener resultados óptimos, sistemáticos y confiables del uso de la aplicación.

4.1 ACTIVACIÓN DE ALERTA AL SOBREPASAR EL LÍMITE DE VELOCIDAD

- Prueba de alerta de velocidad – Límite urbano 50 Km/h

En esta prueba se verificará que la alerta sonora y visual se active en la aplicación en Android cuando el vehículo en movimiento sobrepase la velocidad de 50 Km/h.

Figura 4.1 Alerta de límite de velocidad establecido en 50 Km/h

En la Figura 4.1 se verifica la activación de la alerta al sobrepasar el límite urbano de 50 Km/h el cual se muestra en la interfaz de la aplicación Android y coincide con el valor mostrado por el velocímetro al momento de activarse la alerta.

- Prueba de alerta de velocidad – Límite perimetral 90 Km/h

Figura 4.2 Alerta de límite de velocidad establecido en 90Km/h.

En la Figura 4.2 se verifica la activación de la alerta al sobrepasar el límite de 90Km/h en una vía perimetral el cual se muestra en la interfaz de la aplicación Android y coincide con el valor mostrado por el velocímetro al momento de activarse la alerta.

- Prueba de alerta de velocidad – Límite en carretera 100 Km/h.

Figura 4.3 Alerta de límite de velocidad establecido en 100 Km/h.

En la Figura 4.3 se verifica la activación de la alerta al sobrepasar el límite de 100 Km/h en una carretera el cual se muestra en la interfaz de la aplicación Android y coincide con el valor mostrado por el velocímetro al momento de activarse la alerta.

4.2 ÁNGULO CIEGO

Las pruebas de funcionamiento de las alertas del ángulo ciego se realizaron con un vehículo en movimiento recopilando la información de los sensores para determinar el rango correcto en el que se detecta objetos en el ángulo ciego del vehículo.

En la figura 4.4 se puede ver la prueba de funcionamiento del sensor derecho e izquierdo cuando un objeto entra en la zona del ángulo ciego y el vehículo en movimiento.

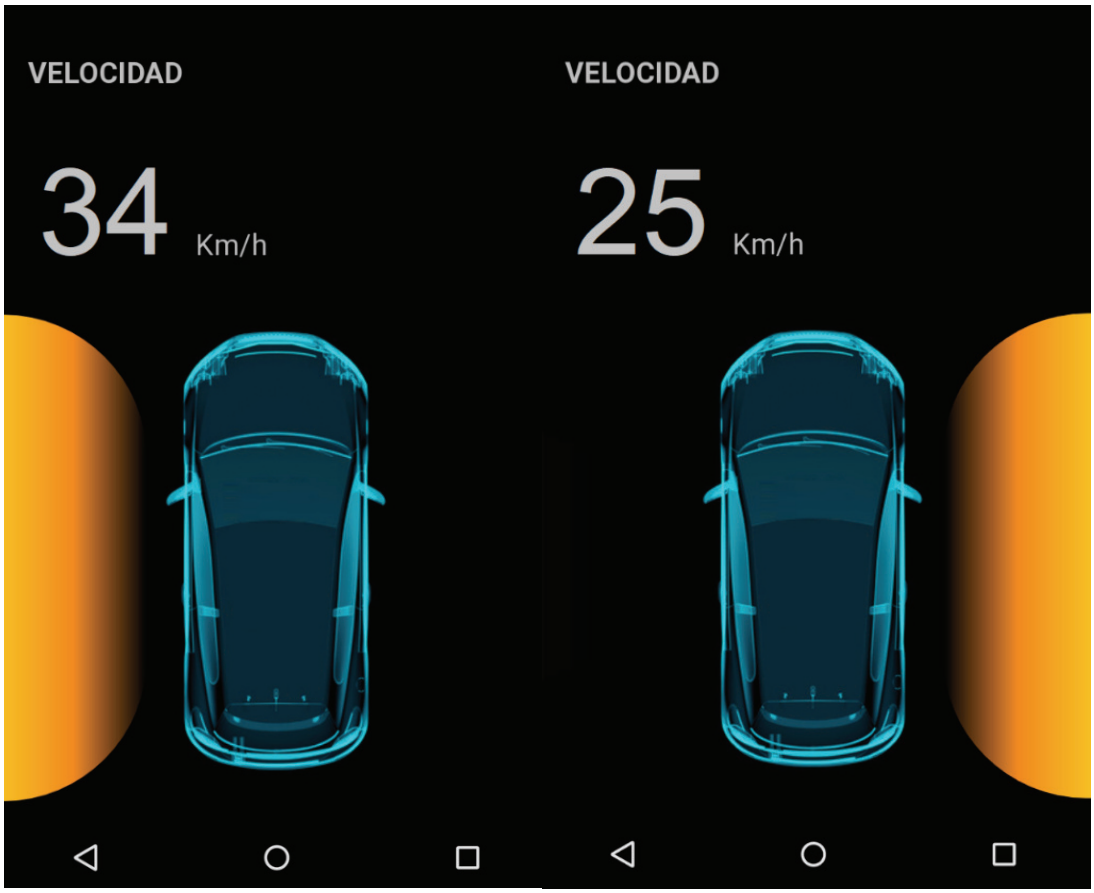
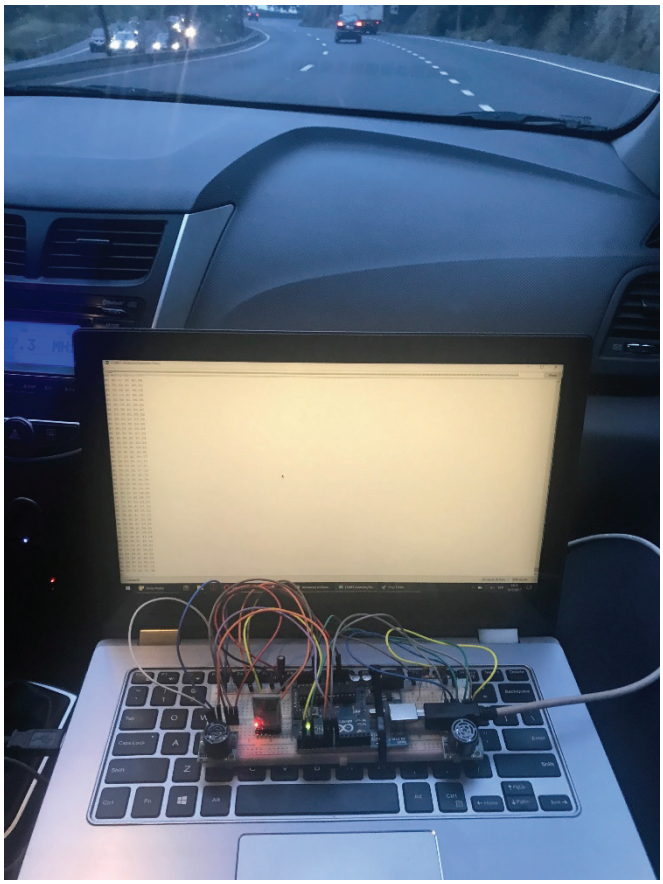


Figura 4.4 Interfaz de alerta para ángulo ciego en el lado derecho e izquierdo del vehículo.

4.2.1 PRUEBAS DE FUNCIONAMIENTO DE LOS SENSORES



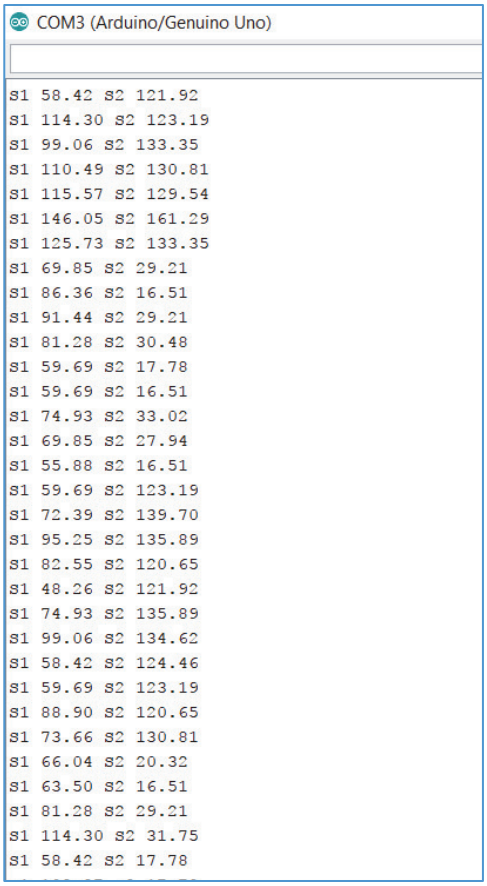


Figura 4.5 Recopilación de datos del sensor en un vehículo en movimiento

En esta prueba de funcionamiento (figura 4.5) se realizó con el vehículo en movimiento de tal manera que se pueda determinar la distancia a la que se detecta un objeto en el área del sensor cuando un vehículo se encuentra en el área de ángulo ciego.

La prueba consistió en posicionar los dos sensores en el lateral del vehículo para ir recopilando los datos de la distancia que genera los dos sensores, las medidas se realizaron para un intervalo de 200 ms para cada muestra.

La recopilación de los datos se hizo por medio del terminal serial incorporado en el IDE de desarrollo de Arduino como se puede observar en la figura 4.6.

De los datos obtenidos se identificó que la distancia de acercamiento de un vehículo que se ubica en el ángulo ciego se encuentra entre 1,5 m y 2,75 m.

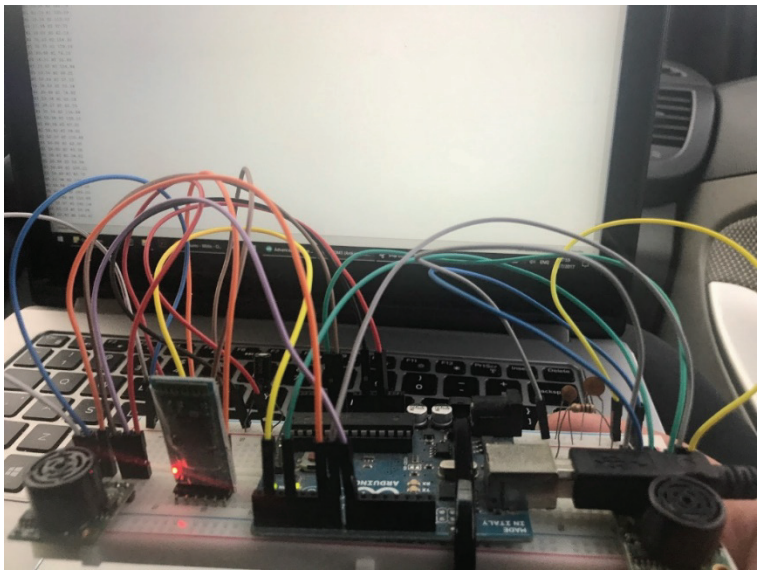


Figura 4.6 Pruebas de funcionamiento de los sensores para detectar el ángulo ciego.

4.2.2 DETECCIÓN DE OBJETOS CON EL VEHÍCULO EN MOVIMIENTO

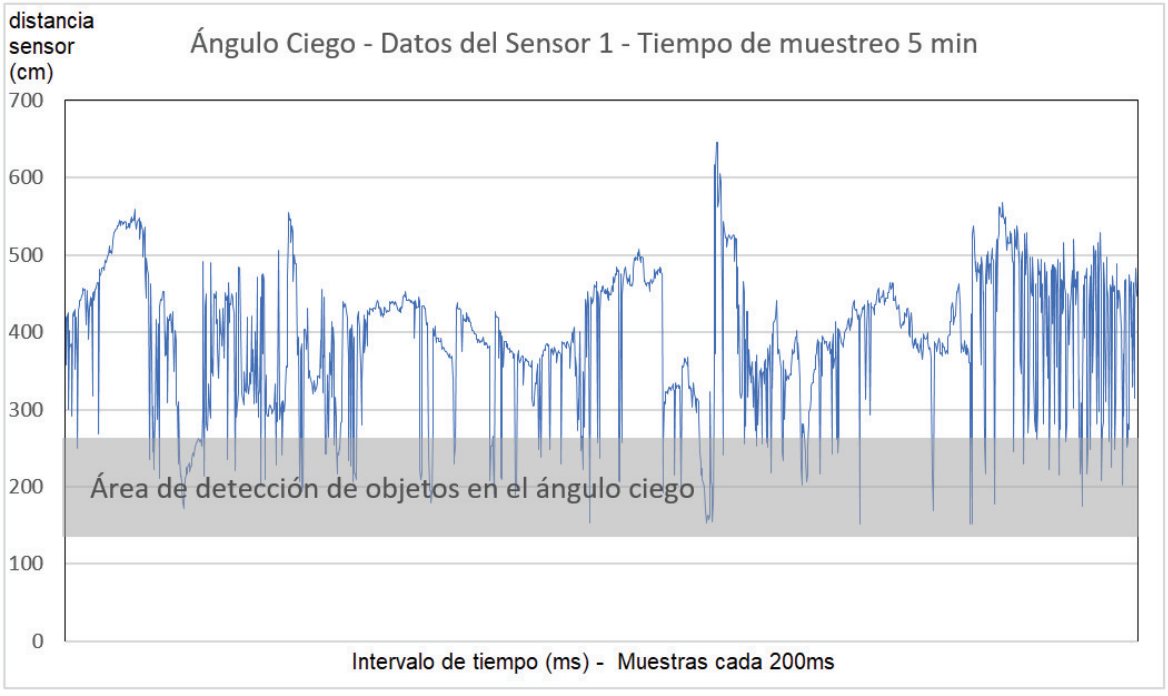


Figura 4.7 Valores de distancia obtenidos y detección de ángulo ciego.

En la figura 4.7, se muestra la recopilación de valores de distancia detectados por el sensor durante un intervalo de 5 minutos, tomando muestras cada 200 ms, ahí se verifica que cada vez que se ubica un vehículo en el ángulo ciego este se encuentra a la distancia entre 1,5 m y 2,75 m.

En la tabla 4.1, se señala las detecciones de objetos en el ángulo ciego, realizando medidas simultáneas por los dos sensores.

Tabla 4.1 Valores de distancia y ángulo ciego

S1	423	S2	446	S1	456	S2	408	S1	288	S2	314	S1	403	S2	462
S1	392	S2	391	S1	321	S2	357	S1	251	S2	273	S1	405	S2	450
S1	442	S2	437	S1	371	S2	358	S1	217	S2	436	S1	226	S2	413
S1	433	S2	432	S1	368	S2	403	S1	244	S2	262	S1	409	S2	460
S1	432	S2	432	S1	446	S2	417	S1	241	S2	258	S1	373	S2	465
S1	433	S2	436	S1	243	S2	395	S1	253	S2	296	S1	206	S2	207
S1	433	S2	437	S1	243	S2	250	S1	249	S2	279	S1	394	S2	461
S1	446	S2	425	S1	243	S2	290	S1	283	S2	392	S1	218	S2	216
S1	439	S2	425	S1	345	S2	378	S1	284	S2	406	S1	216	S2	232
S1	222	S2	225	S1	264	S2	391	S1	343	S2	425	S1	210	S2	215
S1	210	S2	208	S1	274	S2	347	S1	439	S2	424	S1	231	S2	342
S1	220	S2	215	S1	373	S2	410	S1	439	S2	425	S1	419	S2	146
S1	434	S2	429	S1	364	S2	396	S1	433	S2	422	S1	427	S2	442
S1	434	S2	429	S1	344	S2	394	S1	437	S2	428	S1	418	S2	417
S1	434	S2	433	S1	414	S2	424	S1	437	S2	437	S1	405	S2	460
S1	431	S2	437	S1	316	S2	432	S1	427	S2	438	S1	391	S2	465
S1	432	S2	438	S1	406	S2	404	S1	418	S2	445	S1	357	S2	470
S1	428	S2	434	S1	385	S2	418	S1	405	S2	455	S1	279	S2	442
S1	410	S2	432	S1	254	S2	462	S1	364	S2	457	S1	340	S2	456
S1	413	S2	445	S1	316	S2	447	S1	234	S2	212	S1	382	S2	450

Los valores de la tabla 4.1 se registraron mediante la consola que recibe los datos del puerto serie del Arduino, estos valores se encuentran en cm. En las pruebas realizadas se verificó que la distancia de detección de objetos en el ángulo ciego se encontraba en el intervalo de 150 cm a 250 cm, una medida válida se considera cuando se reciben 3 o más valores en este rango los cuales se representan en la tabla resaltados en color naranja.

Esta prueba se realizó de tal manera que permita apreciar cómo se comportan los sensores y comparar su funcionamiento frente al mismo evento, determinando que las medidas realizadas presentan aproximaciones muy cercanas entre el sensor 1 y 2, siendo confiable los datos que permitan ser procesados para su emitir la posterior alerta en la aplicación Android.

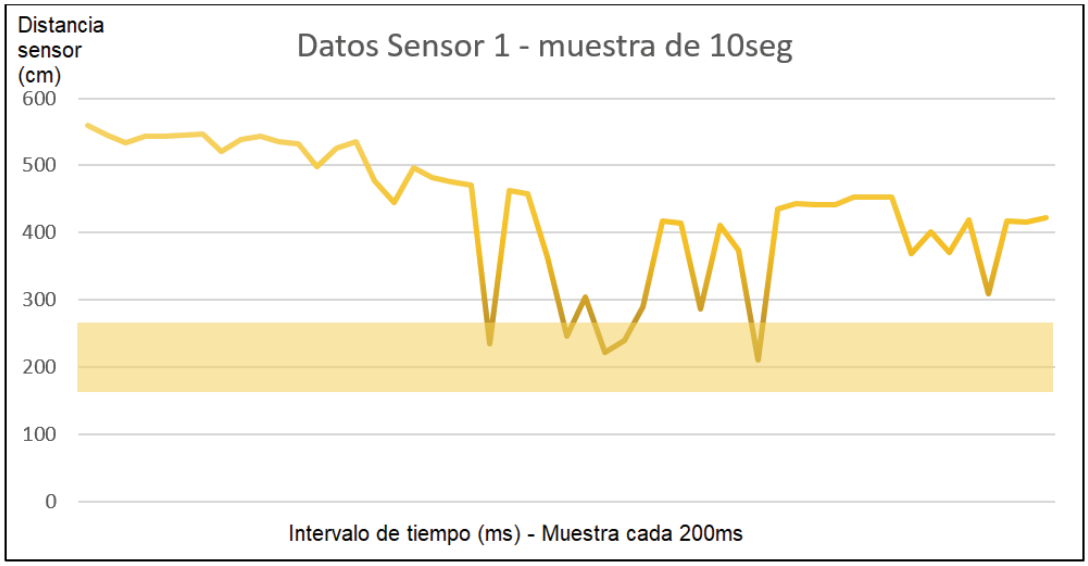


Figura 4.8 Detección de un objeto en el ángulo ciego en una muestra de 10 ms para el sensor 1.

En la Figura 4.8 se indica los valores para el sensor 1 que detecta el sensor durante un intervalo de 10 segundos tomando muestras cada 20 ms al momento que se encuentra un vehículo en el ángulo ciego.

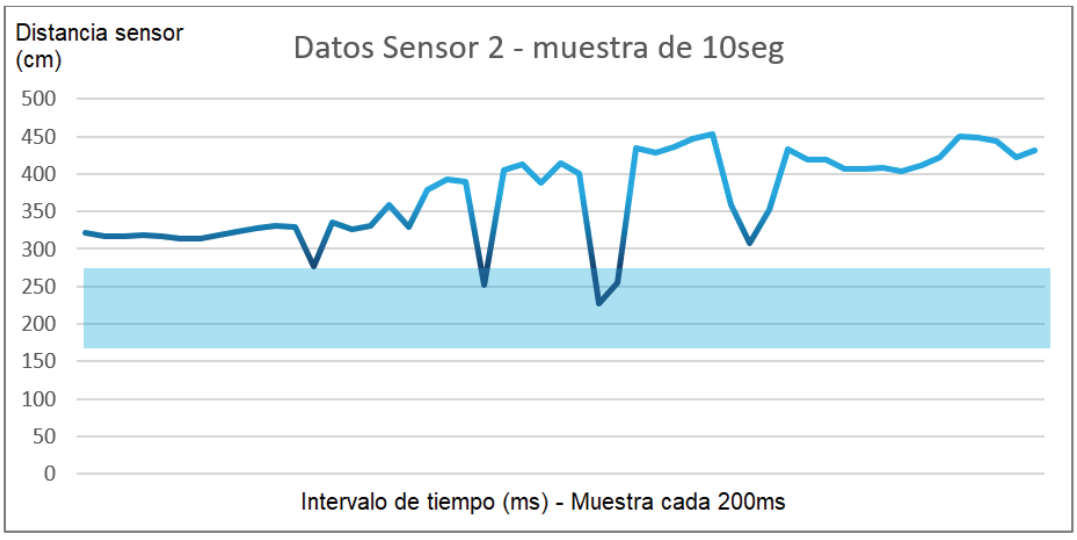


Figura 4.9 Detección de un objeto en el ángulo ciego en una muestra de 10 ms para el sensor 2.

En la Figura 4.9 se indica los valores para el sensor 2 que detecta el sensor durante un intervalo de 10 segundos tomando muestras cada 20 ms al momento que se encuentra un vehículo en el ángulo ciego.

De las pruebas anteriores se establece el intervalo que mejor se ajusta cuando un vehículo en movimiento se encuentra en el ángulo ciego.

4.3 INSTALACIÓN FINAL EN EL VEHÍCULO

Para la instalación del prototipo en el vehículo se cambió las conexiones cableadas por una placa de circuito impreso que permita conectar sensores, módulo Bluetooth y Arduino.

En las figuras 4.10 y 4.11 se muestra el diseño de la placa de circuito impreso y la ubicación de los componentes, esta placa se realizó de tal manera que se conecte como un módulo para la placa Arduino facilitando la instalación de cables de conexión hacia los sensores y fijación del prototipo en el vehículo.

El resultado final de la placa de cobra donde se colocarán los cables de conexión hacia los sensores se puede observar en la figura 4.12.

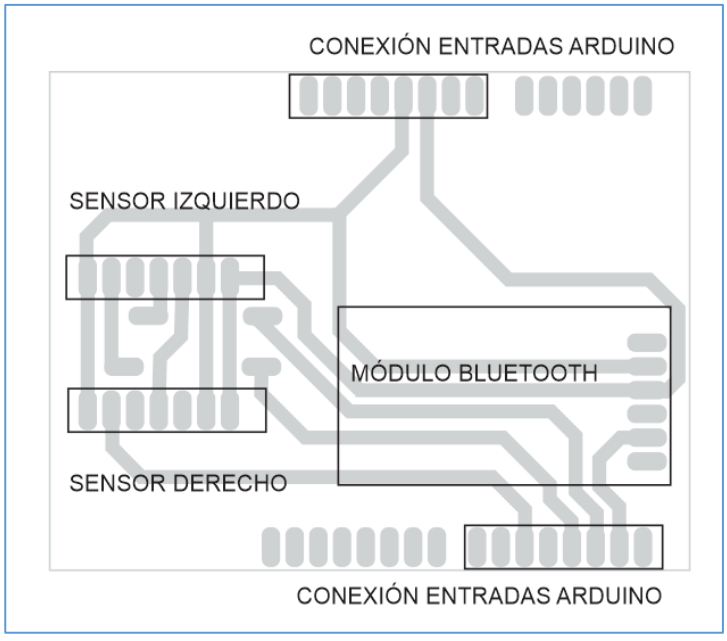


Figura 4.10 Ubicación de componentes en la placa de circuito impreso

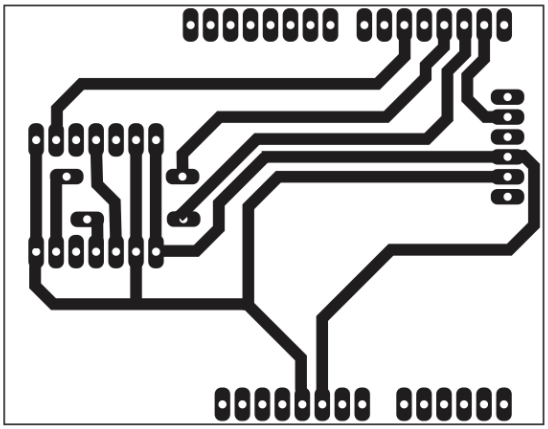


Figura 4.11 Placa de circuito impreso

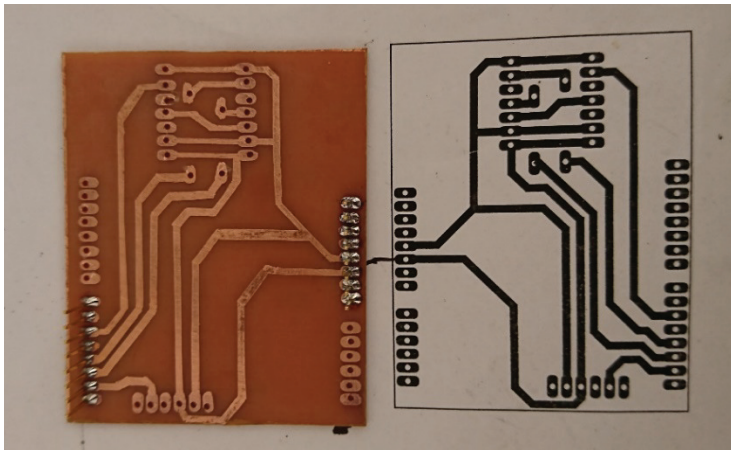


Figura 4.12 Placa de cobre para conexión de sensores y módulo Bluetooth

La placa de cobre se diseñó de tal manera que funcione como un *shield* apilable para la placa Arduino Uno como se puede obseevar en las figuras 4.13 y 4.14.

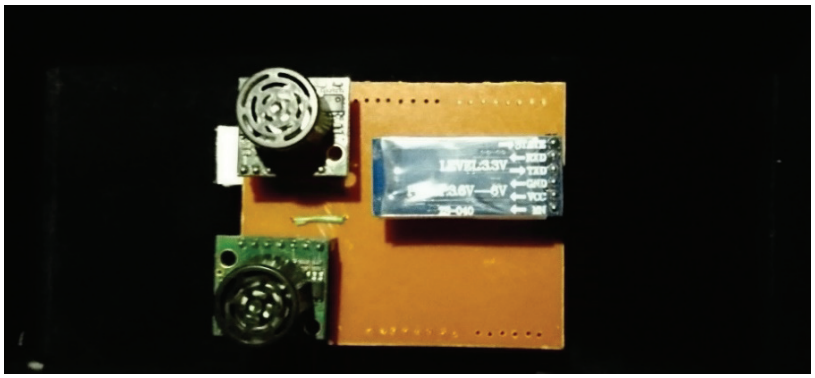


Figura 4.13 Instalación de sensores y módulo Bluetooth

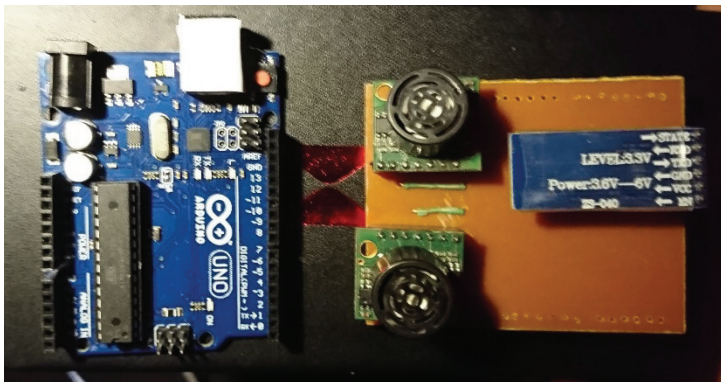


Figura 4.14 Prototipo final para instalar en el vehículo

Para la instalación en el vehículo se ubicó el prototipo ensamblado en el portaequipaje del vehículo (ver figura 4.15) permitiendo un fácil acceso para mantener las conexiones cableadas hacia los sensores, la alimentación del circuito y la señal que transmite el módulo Bluetooth pueda llegar hasta el panel de instrumentos donde se coloca el teléfono Android.

Una vez instalado todos los componentes se procedió a verificar su funcionamiento realizando las pruebas necesarias en los diferentes escenarios de velocidad y ángulo ciego que se presentan en un vehículo en circulación por la ciudad.

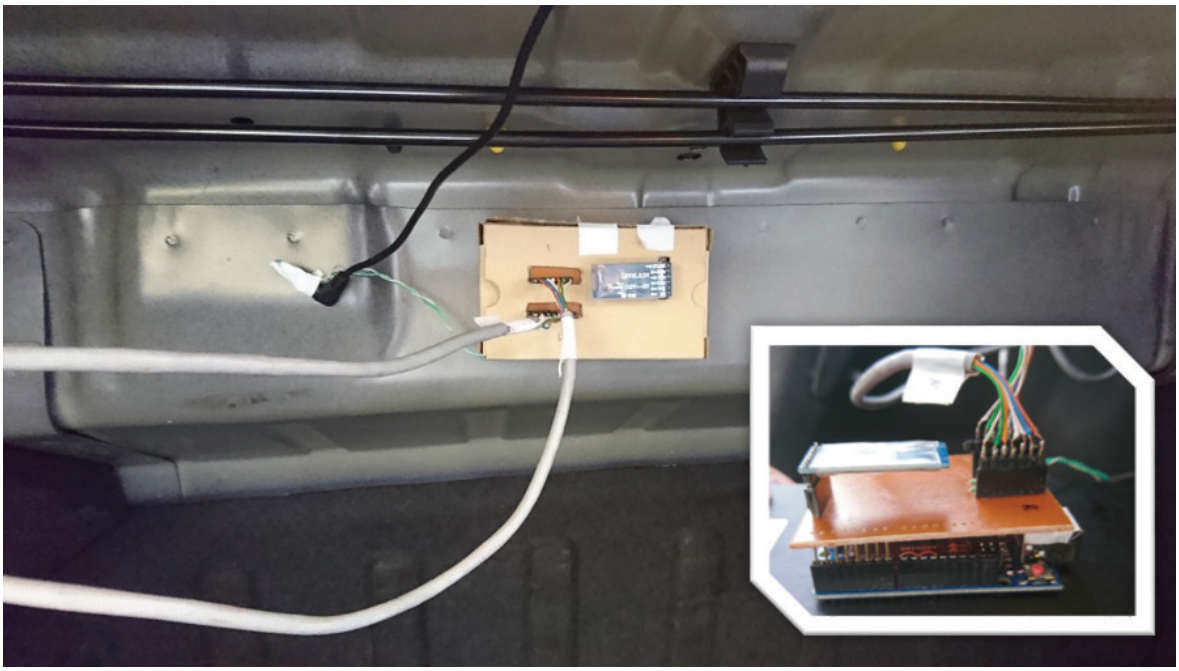


Figura 4.15 Prototipo instalado en el portaequipaje del vehículo

CAPITULO 5 - CONCLUSIONES Y RECOMENDACIONES

5.1 CONCLUSIONES

- Se determinó que la detección por ultrasonido es la más viable considerando la aplicabilidad, instalación y costo de implementación. Su funcionamiento por reflexión de ondas permite cubrir una mayor área detectando los objetos que no son fácilmente observables por el conductor a través de los retrovisores laterales.

- Los sistemas de seguridad vehicular han ido en crecimiento a través de los años y han sido de mayor interés para los sectores automotrices debido a las ventajas que éstos ofrecen sin embargo han estado limitados para la clase media o baja por tanto el sistema propuesto viene a ser de gran utilidad ya que cubriría sectores vulnerables que no disponen de medios económicos para este beneficio global.

- El proyecto realizado ofrece al usuario ventajas como detección de objetos en el ángulo ciego y alerta de límites de velocidad excedidos, que de acuerdo con el análisis realizado es útil para reducir o mitigar los accidentes de tránsito causados por falta de visibilidad, distracción o conducción a altas velocidades de manera práctica, asequible, económica y segura. De este modo disminuye la cantidad heridos, y posibles accidentes consecuentes que cobren vidas inocentes y el daño al entorno lo que significaría gastos municipales.

- El diseño del prototipo realizado cumple las expectativas para el cual fue concebido. Es un sistema integral de fácil instalación e implementación que cuenta con componentes existentes en el mercado a precio razonable lo cual permitirá a futuro implementar el servicio en cualquier tipo de automóvil considerándolo como un proyecto económico, innovador, rentable y útil.

- La culminación de este proyecto da la apertura para implementar un estudio de mercado y análisis de costos con el fin de comercializar el servicio generando alzas en la producción y comercio nacional lo que significa una mejora en la economía de sectores pequeños y el posterior desarrollo del país.

- La aplicación diseñada es de fácil manejo, e indica de manera clara la información requerida. Cuenta con dos secciones, en la parte superior el usuario podrá observar los límites máximos, si son excedidos y sus respectivas alertas y en la parte inferior se podrá visualizar la presencia de objetos en el ángulo ciego.

- Debido al avance tecnológico, el proyecto propuesto tiene gran acogida en el mercado ya que la gran mayoría de la población utiliza un teléfono inteligente con sistema Bluetooth y GPS incorporados, herramientas necesarias para el correcto funcionamiento de la aplicación. Estas son herramientas gratuitas y sin consumo de datos lo que permitirá su funcionamiento en cualquier momento.

5.2 RECOMENDACIONES

- Debido avance tecnológico y la variedad de sensores disponibles en el mercado es necesario realizar un estudio detallado de la viabilidad de su uso, instalación y resultados de esta manera mejorar el desempeño de la aplicación y agregar nuevas funciones de ayuda a la seguridad del conductor y ocupantes.

- Antes de elegir el sensor verificar que el área de funcionamiento del sensor cubra la mayor parte del ángulo ciego del vehículo.

- Para disminuir el consumo de batería de la aplicación para Android, es recomendable aumentar el tiempo de retardo de las peticiones realizadas al GPS.

- El prototipo al ser un dispositivo de seguridad pasiva su función sólo es alertar al conductor el cual debe tomar las acciones necesarias para evitar algún percance.

- En las pruebas se verificó que la señal de GPS se pierde al ingresar en entornos cerrados como túneles, como mejora se podría reemplazar el uso del GPS por un escáner OBD que envíe los datos del velocímetro al vehículo.

- Para mantener la conectividad entre la aplicación Android y Arduino es necesario configurar que la aplicación se mantenga en funcionamiento en segundo plano, caso contrario se cerrará la conexión Bluetooth si en algún momento se cambia a otras aplicaciones del teléfono.

- Se recomienda mantener como máximo 3 dispositivos Bluetooth conectados al teléfono (Ejemplo: radio del vehículo, manos libres y Arduino) ya que en las pruebas se verificó que al sobrepasar este número de dispositivos la aplicación Android dejaba de recibir los datos del Arduino.

- Revisar que el voltaje de alimentación y el voltaje que se entrega a las diferentes entradas del Arduino se mantenga en 5 V y no superior para aumentar la vida útil del dispositivo.

BIBLIOGRAFÍA

[1] IBÁÑEZ, “Sistemas de detección en los coches para evitar accidentes,” 2011. [Online]. Available: https://www.xataka.com/automovil/sistemas-de-deteccion-en-los-coches-para-evitar-accidentes.

[2] T. Kon, “Collision Warning and Avoidance System for Crest Vertical Curves,” p. 41.

[3] “IR sensor principles,” 2015. [Online]. Available: http://education.rec.ri.cmu.edu/content/electronics/boe/ir_sensor/images/409px-IR_Sensor_Principles. [Accessed: 14-Jul-2017].

[4] “Panasonic совместно с МТ-Систем представляет GRID-EYE,” 2015. [Online]. Available: http://www.mt-system.ru/news/panasonic/grid-eye-pervyj-v-mire-infrakrasnyj-sensor-poverhnostnogo-montazha-na-osnove-massiva-. [Accessed: 14-Jul-2017].

[5] “ComoHacerTuRobot.com - Taller: Sensor e Interruptores de Proximidad Para Robots,” 2013. [Online]. Available: http://www.comohacerturobot.com/Taller/taller-sensorProximidad.htm. [Accessed: 14-Jul-2017].

[6] C. E. Canto, “Sensores Ultrasónicos,” p. 10, 2006.

[7] J. D. Pascual, “Prueba Lexus Ls 600h (Parte 3): Seguridad y Conducción,” 2008. [Online]. Available: https://www.motor.es/pruebas-coches/prueba-lexus-ls-600h-parte-3-seguridad-y-conduccion.html. [Accessed: 14-Jul-2017].

[8] “Así funcionan los sensores LIDAR de los coches autónomos,” 2016. [Online]. Available: https://hipertextual.com/2016/03/sensores-lidar. [Accessed: 14-Jul-2017].

[9] “What is Arduino?,” 2017. [Online]. Available: https://www.arduino.cc/en/Guide/Introduction.

[10] D. Kushner and D. Kushner, “The Making of Arduino,” *IEEE Spectr.*, 2011.

[11] “¿Qué es y cómo funciona los wearables? DicZionario,” 2014. [Online]. Available: http://blogginzenith.zenithmedia.es/que-es-y-como-funcionan-los-wearables-diczionario/. [Accessed: 14-Jul-2017].

[12] A. Team, “Arduino Products,” 2017. [Online]. Available: https://www.arduino.cc/en/Main/Products?from=Main.GenuinoProducts.

[13] “Arduino Products,” 2017. [Online]. Available: https://www.arduino.cc/en/Main/Products.

[14] “Compare board specs,” 2017. [Online]. Available: https://www.arduino.cc/en/Products/Compare.

[15] E. Crespo, “Shields para Arduino | Aprendiendo Arduino,” 2017. [Online]. Available: https://aprendiendoarduino.wordpress.com/2015/03/23/shields-para-arduino/. [Accessed: 14-Jul-2017].

[16] “ARDUINO ETHERNET SHIELD 2,” 2017. [Online]. Available: https://store.arduino.cc/usa/arduino-ethernet-shield-2.

[17] ArduinoDevelopers, “Arduino Wi-Fi Shield,” 2017. [Online]. Available: https://www.arduino.cc/en/Main/ArduinoWiFiShield.

[18] “ARDUINO GSM SHIELD 2 (INTEGRATED ANTENNA),” 2017. [Online]. Available: https://store.arduino.cc/usa/arduino-gsm-shield-2-integrated-antenna.

[19] ArduinoDevelopers, “Arduino Shields Motors rev2,” 2017. [Online]. Available: https://store.arduino.cc/usa/arduino-motor-shield-rev3.

[20] “Adafruit Ultimate GPS Logger Shield - Includes GPS Module ID : Adafruit Industries, Unique & fun DIY electronics and kits.” [Online]. Available: https://www.adafruit.com/product/1272. [Accessed: 14-Jul-2017].

[21] “XBee Shield V2.0 - Seeed Wiki.” [Online]. Available:

http://wiki.seeed.cc/XBee_Shield_V2.0/. [Accessed: 14-Jul-2017].

[22] J. R. S. A. Zamudio, “Bluetooth HC-05 y HC-06 Tutorial de Configuración,” 2014. [Online]. Available: http://www.geekfactory.mx/tutoriales/bluetooth-hc-05-y-hc-06-tutorial-de-configuracion/.

[23] “Access the Online IDE.” [Online]. Available: https://www.arduino.cc/en/Main/Software.

[24] “WHAT IS ARDUINO CREATE?,” 2017. [Online]. Available: https://create.arduino.cc/.

[25] “Blinking an LED with the Arduino* IDE,” 2016. [Online]. Available: https://software.intel.com/en-us/get-started-arduino-blink. [Accessed: 14-Jul-2017].

[26] “Arduino Blog » Arduino Create is a one stop shop for Makers,” 2016. [Online]. Available: https://blog.arduino.cc/2016/05/20/arduino-create-is-a-one-stop-shop-for-makers/. [Accessed: 19-Jul-2017].

[27] Arduino Team, “Arduino Blog » Arduino Create is a one stop shop for Makers,” 2016. [Online]. Available: https://blog.arduino.cc/2016/05/20/arduino-create-is-a-one-stop-shop-for-makers/. [Accessed: 14-Jul-2017].

[28] “The Android Source Code | Android Open Source Project,” 2017. [Online]. Available: https://source.android.com/source/. [Accessed: 19-Jul-2017].

[29] Marcos Bermejo, “El Kanban,” p. 36, 2014.

[30] “Desarrollo Ágil con Kanban,” 2011. [Online]. Available: https://desarrolloweb.com/articulos/desarrollo-agil-kanban.html. [Accessed: 19-Jul-2017].

[31] “What is Kanban? An Introduction to Kanban Methodology,” 2017. [Online]. Available: https://www.digite.com/kanban-guide/what-is-kanban/. [Accessed: 19-Jul-2017].

[32] Javier Garzás, “Kanban: una explicación del método,” 2011. [Online]. Available: http://www.javiergarzas.com/2011/11/kanban.html. [Accessed: 19-Jul-2017].

[33] Maxbotix, “LV-MaxSonar-EZ,” 2017.

[34] “LV-MaxSonar ® -EZ™ Series LV-MaxSonar ® -EZ™ Series High Performance Sonar Range Finder,” 2015.

[35] Mohannad Rawashdeh, “All About Max Sonar EZ0 and Arduino: 3 Steps,” 2013. [Online]. Available: http://www.instructables.com/id/Max-Sonar-EZ0/. [Accessed: 19-Jul-2017].

[36] M. D. C. Garcia, “Conociendo la placa ARDUINO UNO Rev. 3.,” 2015. [Online]. Available: http://miarduinounotieneunblog.blogspot.com/2015/12/conociendo-la-placa-arduino-uno-rev-3_11.html.

[37] Naylamp, “Configuración del módulo bluetooth HC-05 usando comandos AT,” 2016. [Online]. Available: http://www.naylampmechatronics.com/blog/24_Configuración--del-módulo-bluetooth-HC-05-usa.html.

[38] M. Girish, “Arduino Bluetooth Basic Tutorial.” [Online]. Available: https://create.arduino.cc/projecthub/user206876468/arduino-bluetooth-basic-tutorial-d8b737.

[39] AGENCIA NACIONAL DE TRANSITO - ECUADOR, “Fotoradares de Última Tecnología para el Control de Velocidades,” 2017. [Online]. Available: http://www.ant.gob.ec/index.php/component/content/article/49-boletines/189-agencia-nacional-de-transito-participa-en-la-semana-de-seguridad-vial#.WW_YhIg1_IV. [Accessed: 19-Jul-2017].

[40] “LocationListener | Android Developers,” 2017. [Online]. Available: https://developer.android.com/reference/android/location/LocationListener.ht

ml. [Accessed: 19-Jul-2017].