GPS GUIDED GROUND VEHICLE

Under The Guidance Of :
Prof. P. BHARANI CHANDRA KUMAR
GITAM DEEMED TO BE UNIVERSITY

Submitted By:-

POTNURU SAMITA – VU22EECE0100371
MEHBOOB SAHIL SHARIEF – VU22EEECE0100037
SARVANI RAPETI – VU22EECE0100043
SHIVA TEJA - VU22EECE0200001
EECE Department
Gandhi Institute of Technology and Management (GITAM)
Visakhapatnam
Andhra Pradesh
India

Acknowledgement

I would like to express my sincere gratitude to Prof. P. BHARANI CHANDRA KUMAR sir for providing me the opportunity to work on this internship project and for his guidance throughout the internship. I am grateful for their mentorship which not only enhanced my technical skills.

I would also like to thank Lokesh Sir and Abbas for their guidance and patience throughout the Internship. They played a crucial role in guiding me throughout the project. Their expertise in GPS Guided Ground vehicle was the critical factor in the completion of the project.

Finally, it is a great pleasure to thank one and all who helped us directly and indirectly throughout this Internship.

NAME AND ROLL NO. OF THE STUDENTS

POTNURU SAMITA – VU22EECE0100371
MEHBOOB SAHIL SHARIEF – VU22EEECE0100037
SARVANI RAPETI – VU22EECE0100043
SHIVA TEJA – VU22EECE0200001

INDEX :-

1. Objective:

1. Build a GPS-guided robot.
2. Optimize robot movements based on GPS waypoints using Arduino.
3. Gain experience in autonomous robotic systems with MIT App Inventor integration.

2. Abstract: The Instructables guide on building a GPS(Global Positioning System)-guided robot outlines the process of creating a robot that autonomously navigates using GPS coordinates. The project uses a microcontroller (Arduino), GPS module, and motor controller to direct the robot's movements. With step-by-step instructions, the tutorial covers wiring, coding, and testing, allowing the robot to travel to predetermined waypoints based on GPS signals. It's an ideal project for hobbyists looking to learn about robotics, GPS technology, and autonomous systems[1].

3. Problem Statement: The project addresses the challenge of efficiently controlling ground vehicles remotely by solving the problem of complex manual navigation, allowing users to autonomously send vehicles to a predefined location with a single command, reducing the need for continuous human control and intervention[2].

4. **Introduction**: In recent years, GPS-guided robotic systems have gained increasing attention for their role in autonomous navigation.These systems utilize GPS technology to accurately navigate robots to pre-programmed waypoints, allowing for efficient movement over various landscapes.Central to these systems are microcontrollers, like Arduino, which processing GPS data, while motor controllers translate this data into movement, allowing the robot to adjust its path in real time. Ground vehicles that operated autonomously (such as in warehouses, mines, and agriculture) were often designed to follow fixed paths, either physically marked (like tracks, rails, or wires in the ground) or pre-programmed. They had little or no ability to deviate from these routes. This limited their functionality to highly controlled environments where there was little or a lot of change. This project offers hands-on experience in integrating hardware and software. It demonstrates how GPS signals can control a robot's movement and highlights key skills in programming, circuit design, and problem-solving. Ideal for those interested in robotics and automation, it provides practical insights into the navigation technologies used in applications like autonomous vehicles and drones[3].

4. Literature Review:

1. The article highlights innovations in GPS-guided autonomous navigation systems for robots. It explores how combining GPS for precise tracking with Bluetooth for remote control enhances the robots' adaptability in diverse environments, allowing for both industrial and personal applications. Equipped with additional sensors like magnetometers for direction and ultrasonic sensors for obstacle detection, these robots can autonomously navigate complex settings with impressive accuracy and flexibility. Such capabilities make them valuable for tasks ranging from logistics in manufacturing to personalized load carrying, thus streamlining operations and reducing the need for manual labor[4].
2. The article "Arduino-Based GPS Guided Vehicle" examines various components of GPS-guided vehicles and autonomous navigation systems. It underscores the importance of connecting with at least four satellites to ensure accurate GPS positioning and highlights the role of autonomous delivery robots in minimizing human contact—especially valuable during the COVID-19 pandemic. The use of the Extended Kalman Filter (EKF) is discussed for its effectiveness in state prediction and error correction through sensor integration. The article also describes the GPS system's three main segments—Space, Control, and User—and presents a hybrid tracking method that

combines GPS with dead reckoning for enhanced navigation in urban areas. Additionally, it showcases an economical autonomous guided vehicle (AGV) design, incorporating GPS for route planning, a MIMO antenna array for precise location accuracy, and the potential of GPS-driven driverless cars as a cost-effective alternative to expensive technologies like LiDAR. The review further explores applications like vapor compression systems with ejectors and gesture-based robotic control, emphasizing the broad range of advancements in GPS and autonomous navigation technologies[5].

3. The article "GPS-Based Autonomous Vehicle Navigation in Robotics Along with Directionality" details the successful design of an autonomous vehicle tracking robot platform, specifically aimed at operating in environments where human presence is challenging or unsafe. The platform proved to be fully functional, effectively navigating from its starting point to its destination by relying on GPS data and gyroscope (compass) readings. When tested without a PID controller, the robot experienced deviations from its set path, causing delays. However, by adjusting the motor speeds on each side of the robot based on gyroscope feedback, the system reduced path deviations by 50% and achieved smoother movement. Although this adjustment slightly reduced the robot's speed by 33.3%, it maintained a steady course and reached its destination within a similar timeframe, even without PID control[6].
4. The paper "GPS-Based Autonomous Vehicle Navigation and Control" presents a GPS-driven navigation and control system designed to autonomously route vehicles, such as cabs, to passengers using an Android application. The system centers on an autonomous route calculation and control algorithm that leverages GPS signals to determine the vehicle's position, speed, and timing. To find the shortest route, the navigation system employs Haversine distance calculations, ensuring efficient path planning to the destination. The Android app sends real-time location data to a Central Control Room (CCR), which then dispatches the nearest vehicle to the user's location. Key hardware components include Arduino-based microcontrollers and GSM modules for live data transmission, showcasing the system's potential to improve accuracy and efficiency in urban vehicle dispatch[7].
5. The paper "Unmanned Ground Vehicle Control using IoT" introduces a GPS-based, IoT-enabled navigation system for cab dispatching, aiming to improve the efficiency and accuracy of vehicle routing by replacing traditional staff-dependent voice communication with digital data transmission. This approach leverages satellite tracking for precise location monitoring, which allows for faster dispatching and reduces human error, enhancing customer service in the competitive cab industry. Despite providing a promising solution, the paper's limited review of prior research and related systems weakens its contextual grounding, as it overlooks a thorough discussion of existing technologies that could have highlighted the unique contributions and strengths of their system[8].
6. The article "Design of an Autonomous Controlled and GPS-Guided Experimental Small-Scale Prototype" explores various methodologies and technologies that advance reliable and efficient navigation for autonomous vehicles. A key focus is on integrating GPS with other sensors, such as inertial measurement units (IMUs) and magnetometers, to improve positional accuracy and address GPS limitations like environmental interference and satellite visibility issues. Algorithms like Kalman filters and A* are utilized to refine pathfinding and navigation. The research also emphasizes the use of

commercially off-the-shelf (COTS) components to reduce costs without compromising functionality, making these systems more accessible for experimentation. To tackle real-time obstacle detection, sensor technologies like infrared and ultrasonic sensors are incorporated alongside GPS to enable safe navigation in complex environments. Collectively, these efforts reflect a trend toward integrating diverse technologies and algorithms to enhance autonomous navigation systems' accuracy, efficiency, and affordability[9].

7. The paper "Object Detection and Avoidance in Unmanned Ground Vehicle using Arduino" describes a simple, low-cost model of an unmanned ground vehicle (UGV) designed for obstacle detection using Arduino Uno. The primary component for obstacle detection is an ultrasonic sensor, which emits ultrasonic waves and detects objects or obstacles within range. Upon detecting an obstacle, the sensor sends a signal to the Arduino UNO, which is connected to a motor driver shield programmed to control the vehicle. The motor driver regulates the DC motors based on input from the Arduino, allowing the UGV to navigate around obstacles. This project holds promising potential for advancing affordable robotics applications in India[10].
8. The article "GPS-Based Autonomous Robot Navigation in Robotics Along With Directionality" highlights extensive research on GPS-guided navigation and control in autonomous vehicles. Key areas of exploration include automated luggage systems powered by ultrasonic sensors and DC motors, fuzzy variable structure controllers for path following, vision-based control, and neural network applications in autonomous driving. Practical implementations, such as Volkswagen's work with GPS and visual lane detection in controlled settings, as well as studies on multi-robot coordination and leader-follower algorithms, demonstrate the versatility of GPS in navigation. Integrating technologies like Bluetooth, GPS receivers, and magnetometers has been shown to improve both accuracy and reliability. Notable advancements include reactive path planning for dynamic environments and deliberative planning for static ones, with current trends emphasizing affordable solutions that combine GPS guidance with magnetic compass navigation to enhance directional control in autonomous systems[11].
9. The article "A Cost-Effective GPS-Aided Autonomous Guided Vehicle for Global Path Planning" emphasizes the role of mobile robots as central testbeds in Industry 4.0, highlighting path planning as a fundamental challenge in robotics. It differentiates between two primary architectures: reactive (local) path planning, which generates paths in real-time using sensor data without prior knowledge of the environment, and deliberative (global) path planning, which relies on pre-existing environmental maps to create safe routes. The reactive approach is computationally efficient and well-suited for dynamic environments, while deliberative planning excels in static settings but demands more processing power and may necessitate re-planning when conditions change. The literature showcases various implementations of these methods using diverse sensors and control architectures, each presenting its own advantages and limitations based on the specific application context[12].
10. the technical implementation of an autonomous cab dispatch system. It highlights the use of Arduino microcontrollers, GPS modules, and GSM shields, along with Python libraries. Additionally, the development of the graphical user interface (GUI) incorporates technologies like MySQL, PHP, and Google Maps, showcasing a comprehensive approach to creating an efficient and user-friendly cab dispatch solution[13].

11. The paper "Internet of Things-Based Vehicle Tracking and Monitoring System" outlines the implementation and results of a prototype for an IoT-based vehicle monitoring and tracking system. The system features an onboard module discreetly installed within the vehicle and a base station that monitors the vehicle's longitude and latitude. At its core, the system utilizes an Arduino Uno microcontroller, which connects to a Neo 6M GPS module to track the vehicle's location. Additionally, a WiFi module (ESP8266) transmits the sensed location wirelessly to the ThinkSpeak API cloud service for analysis and storage. An Android mobile application is also developed to utilize the tracked data, providing users with real-time information on the vehicle's precise location displayed on a map[14].
12. The article "A Cost-Effective GPS-Aided Autonomous Guided Vehicle for Global Path Planning" emphasizes the role of mobile robots as essential testbeds in Industry 4.0, highlighting path planning as a critical challenge in robotics. It categorizes path planning into two primary architectures: reactive (local) path planning, which functions without prior knowledge of the environment and generates paths in real-time based on sensor data, and deliberative (global) path planning, which relies on pre-existing environmental maps to create safe routes. The reactive approach is less computationally intensive and more suitable for dynamic environments, whereas deliberative planning is more effective in static contexts but requires greater processing power and may necessitate re-planning in response to changing conditions. The report details various implementations of these methods, showcasing different sensors and control architectures, each with unique advantages and limitations tailored to specific application scenarios[15].
13. The paper presents the autonomous navigation of the Komodo UGV made by the DOK-ING company. It can sustain the condition in extremely hot zones . The module allows multi user task assignment and supervision via the client server framework. The system consists of QGIS(Q-Geographic information system) integration with ROS navigation holding on navigation software . The user can supervise the position of the vehicle in the map . If the vehicle does not move the user can find out its current position through the home position . During the position the user can track the position of the view position request . Three scenarios are performed: Waypoint, Patrolling and Fail Safe. e. The experiments on the Komodo UGV demonstrated successful execution of the user-set waypoint, patrolling and fail-safe scenarios. Furthermore, experiments show that our navigation system provides fast and reactive motion among obstacles on flat terrain. Future work will focus on extending the navigation system for operation on uneven terrains[16].
14. The paper presents the use of GPS to navigate an unmanned vehicle . The main task of their project is UV travel waypoint A to waypoint B and Develop the obstacle avoidance algorithms. The UGV will work on the principle of mutual working of GPS with obstacle avoidance. The GPS will acquire the coordinates and check if the source and destination are not the same. Obstacle avoidance operations depend on the threshold voltage of the sensors inclined at an angle vertically so that they may also get some data out of the wall of the bot. This paper mainly shows the limitation in using only GPS . This can be handled by Sensor Fusion which integrates data from multiple sources like GPS ,IMU's and ultrasonic sensor is another solution to handle GPS signal loss or interference. By fusing input by various sensors the UGV can maintain reliable navigation in a challenging environment, even when GPS is unavailable. The ongoing development of

GPS ultrasonic system and adaptive algorithm contributes to enhancing UGV reliability , broadening their use in complex , real world scenarios[17].

6. Hardware Components:

6.1. HC-05 Bluetooth Module

1. **What it is**: A Bluetooth module for wireless communication between microcontrollers and devices.
2. **Key Features:**
   - Works on 3.3V to 6V, so 5V from Arduino is fine.
   - Faster responses.
   - 100m range for distant control in open areas.
3. **How it's made**: Assembled by integrating the Bluetooth chip, antenna, and supporting components onto a PCB(Printed Circuit Board).
4. **Why use it in your robot:** Enables wireless control and communication for remote robot operation.
5. **Advantages**:
   - Easy setup with microcontrollers
   - Reliable wireless communication
   - Low power consumption
   - Cost-effective Bluetooth solution
6. **Disadvantages:**
   - Limited data transfer speed
   - Range may be reduced with obstacles
   - Only supports slave mode
7. **Applications:**
   - Robotics and remote-controlled vehicles for precise navigation
   - Home automation systems for managing smart devices
   - Wireless data transfer for seamless communication between devices
   - IoT applications for connecting and controlling smart devices
   - Mobile applications for controlling hardware remotely, such as drones or robotic arms

Figure 1 : HC-05 Bluetooth Module[18]

6.2. L293D Motor Controller

1. **What it is:** A dual H-bridge motor driver IC for controlling DC and stepper motors.
2. **Key Features:**
   - Dual H-Bridge controls up to four DC motors.
   - Works with 4.5V to 36V.
   - Handles up to 600 mA per channel (1.2A peak).

3. How it's made: Integrated transistor switches and protection diodes in a single chip, packaged in DIP (Dual In-line Package).

4.Why use it in your robot: Simplifies wiring and control of motors for precise direction and speed management.

5. Advantages:

- Easy interfacing with microcontrollers
- Drives two motors at once
- Supports PWM for speed control
- Built-in protection features

6. Disadvantages:

- Limited current handling for high-power motors
- Requires external power supply for motors
- May generate heat under heavy loads

7. Applications:

- Robotics and robotic arms
- Remote-controlled vehicles
- Automated systems
- DIY motor projects
- Conveyor belts

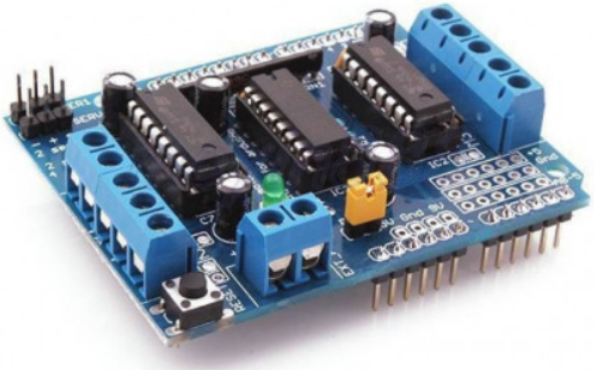


Figure 2 : L293D Motor Controller[19]

6.3. 11.1V Li-Po Battery (3-Cell)

1. What it is: A rechargeable power source made of three lithium polymer cells connected in series.

2. Key Features:

- Lightweight for improved mobile performance
- High discharge rates for power-demanding tasks
- Flat shape for easy fit in compact devices

3.How it's made: Constructed by layering lithium polymer electrolyte between electrodes in a flexible pouch.

4.Why use it in your robot: Provides high energy density and power in a compact form for applications needing power and weight balance.

5. Advantages:

- High energy density for longer run times
- Lightweight improves mobility and efficiency
- Capable of high discharge rates
- Rechargeable with a long lifecycle

6. Disadvantages:

- Requires careful handling and charging

- Can be more expensive than traditional batteries
- Sensitive to temperature extremes
- Requires a specific charger

7. Applications:

- Robotics and robotic vehicles
- Drones and aerial vehicles
- Remote-controlled cars
- Portable electronic devices
- High-performance hobbyist applications

Figure 3 : 11.1V Li-Po Battery (3-Cell) [20]

6.4. HMC5883L Magnetometer/Compass

1. What it is: A three-axis magnetometer used for measuring magnetic fields and determining orientation.

2. Key Features:

- Three-Axis Measurement: X, Y, and Z axes
- Low Power Consumption: Suitable for battery-operated devices
- Calibration Capability: Improves accuracy and compensates for interference

3. How it's made: Integrated magnetoresistive sensors and I²C interface into a single chip.

4. Why use it in your robot: Allows for orientation and navigation capabilities in robotic applications.

5. Advantages:

- Accurate and reliable heading information
- Easy integration with microcontrollers
- Low power consumption
- Compact size

6. Disadvantages:

- Performance affected by magnetic interference
- Requires calibration for accuracy
- Limited range in extreme magnetic environments
- Not as precise as specialized navigation systems

7. Applications:

- Robotics for navigation
- Drones for flight stability
- Smartphone compass applications
- IoT devices needing orientation awareness
- Wearable technology for activity tracking

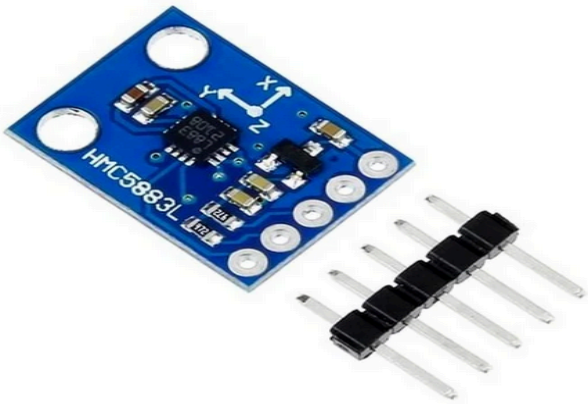


Figure 4 : HMC5883L Magnetometer/Compass[21]

6.5. Ublox NEO-6M GPS Module

1. **What it is:** A GPS receiver module for precise location tracking and navigation.
2. **Key Features:**
   - Operating voltage of 3.0V to 5.5V
   - Can track up to 22 satellites across 50 channels
   - High sensitivity for tracking weak satellite signals
   - Update rate of up to 5 Hz for real-time data
   - Built-in antenna with an external option for better reception
3. **How it's made:** Assembled GPS receiver chip, antenna, and components on a PCB.
4. **Why use it in your robot:** Provides reliable location data for navigation and mapping in robotic applications.
5. **Advantages:**
   - High accuracy for navigation
   - Easy to interface with microcontrollers
   - Low power consumption
   - Robust design for outdoor performance
6. **Disadvantages:**
   - Signal quality may degrade in urban canyons
   - Cold start takes longer to acquire satellites
   - Limited effectiveness indoors
   - May need additional components for optimal operation
7. **Applications:**
   - Robotics for navigation and mapping
   - Drones for flight path tracking
   - Outdoor navigation devices
   - IoT applications needing location services
   - Land surveying and data collection

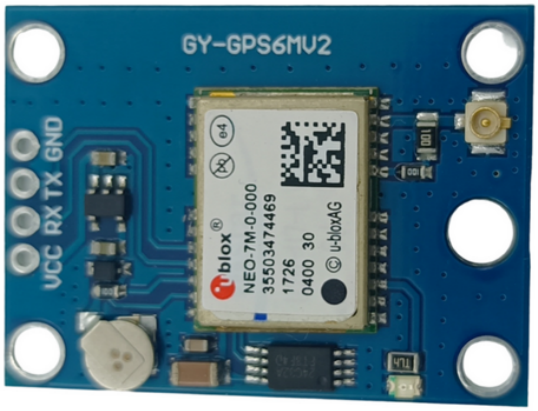


Figure 5 : Ublox NEO-6M GPS Module [22]

6.6. Jumper Wires

1. What it is: Insulated electrical wire used in electronic and electrical applications, color-coded for identification.

2. Key Features:

- Variety of Colors: Indicates specific functions (e.g., red for power, black for ground)
- Insulation: Typically PVC(polymerization of vinyl chloride) or similar material
- Flexible: Often made from stranded copper wire

3. How it's made: Copper drawn through dies to create gauge, insulated using extrusion techniques.

4. Why use it in your robot: Organizes wiring, reduces errors, and improves aesthetics in projects.

5. Advantages:

- Simplifies wiring organization
- Reduces risk of incorrect connections
- Flexible for versatile applications
- Available in various lengths and gauges

6. Disadvantages:

- Insulation durability varies
- Colors may fade over time
- Requires proper handling to avoid damage
- Colors may lead to assumptions without proper labeling

7. Applications:

- Robotics for connecting components
- Prototyping electronic circuits
- Home automation systems
- Automotive wiring

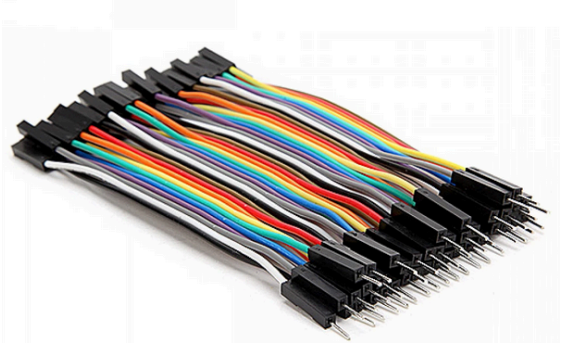


Figure 6 : Jumper wires [23]

6.7. Arduino mega 2560

1. **What It Is:** The Arduino Mega 2560 is a powerful microcontroller board based on the ATmega2560, designed for projects requiring more I/O pins, memory, and processing capability than the Arduino Uno. It's ideal for complex projects in robotics, automation, and installations.
2. **Key Features:**

- The operating voltage of this microcontroller is 5 volts
- 54 digital I/O pins, Analog pins are 16 input pins.
- Length, width, weight of this board is 101.52 mm, 53.3 mm, 36 g respectively.
- The input voltage will range from 6 volts to 20 volts[24].

3. **How It's Made:** The board is assembled with the ATmega2560 microcontroller, along with supporting components, connectors, and headers, on a larger printed circuit board (PCB) to accommodate the additional pins and memory.
4. **Why Use It in Your Robot:** The Arduino Mega 2560 serves as a robust control unit for projects that require numerous sensors, modules, or actuators, providing ample memory and connectivity options for advanced programming and complex operations.
5. **Advantages:**

- High Pin Count: 54 digital and 16 analog pins for extensive connectivity.
- Larger Memory Capacity: 256 KB flash, 8 KB SRAM, and 4 KB EEPROM for storing and handling complex programs.
- Compatibility with Multiple Shields and Modules: Ideal for multi-functional applications.
- Community Support: Like the Uno, the Mega 2560 has extensive resources and support

6. **Disadvantages:**

- Larger Size: May not fit as easily in compact projects.
- Higher Cost: Slightly more expensive than the Uno but still affordable.
- No Onboard Wi-Fi or Bluetooth: Requires external modules or shields for wireless communication.

7. **Applications:**

- Advanced robotics and automation projects
- Complex sensor-based systems or installations
- Projects requiring multiple modules and extensive I/O
- Prototyping for sophisticated electronic applications

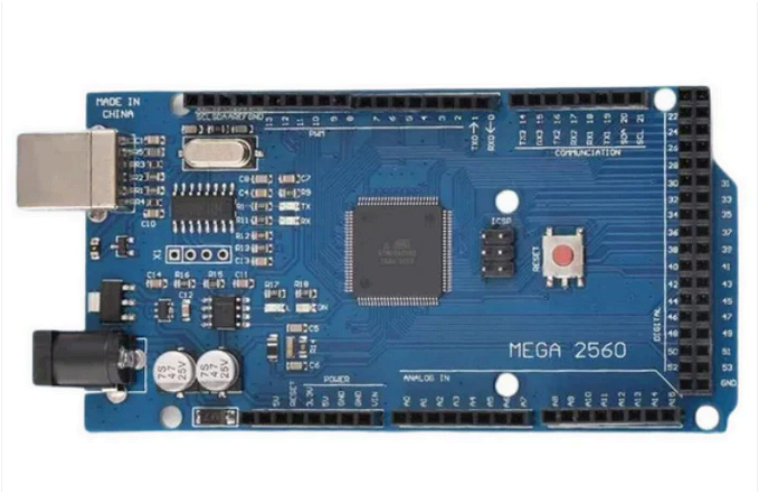


Figure 7 : Arduino Mega 2560 [25]

6.8. **Wheels**:

1. **What Are They:** Wheels are circular components that allow a robot to move by rolling over a surface, providing mobility and support for the robot's chassis.
2. **Key Features:**

- Material: Typically made of plastic, rubber, or foam for traction and durability.
- Diameter: Available in various sizes to accommodate different robot designs and terrains.
- Tread Design: Different tread patterns can provide better grip on various surfaces.

3. **How They're Made:** Wheels are manufactured through processes such as injection molding for plastic or rubber, ensuring they are lightweight yet strong enough to support the robot's weight.
4. **Why Use Them in Your Robot:** Wheels are essential for enabling movement and navigation in robotic applications, allowing robots to travel to desired locations or perform tasks.
5. **Advantages:**

- Simple design and easy to attach to a robot chassis
- Can be used on various terrains depending on material and tread design
- Lightweight options available to minimize overall robot weight

6. Disadvantages:

- Limited mobility on uneven or rough surfaces compared to tracks or legs
- May require specific wheel sizes and configurations based on the robot's design

7. Applications:

- Used in mobile robots, autonomous vehicles, and remote-controlled cars
- Common in educational robotics kits for hands-on learning
- Employed in robotic arms and automated guided vehicles (AGVs) for transporting materials

Figure 8 : Wheels [26]

7. Software:

1. **Arduino IDE:** The Arduino IDE (Integrated Development Environment) is a user-friendly platform designed for programming Arduino boards. It features an intuitive interface, a code editor with syntax highlighting and auto-completion, and a robust library management system, which simplifies code development by providing access to various built-in libraries for sensors and modules. Users can select their specific Arduino board for proper compilation and uploading. The Serial Monitor allows for real-time communication and debugging, while programs, referred to as "sketches," can be easily saved and modified. The IDE is cross-platform compatible,

available for Windows, macOS, and Linux, and benefits from extensive community support and documentation.

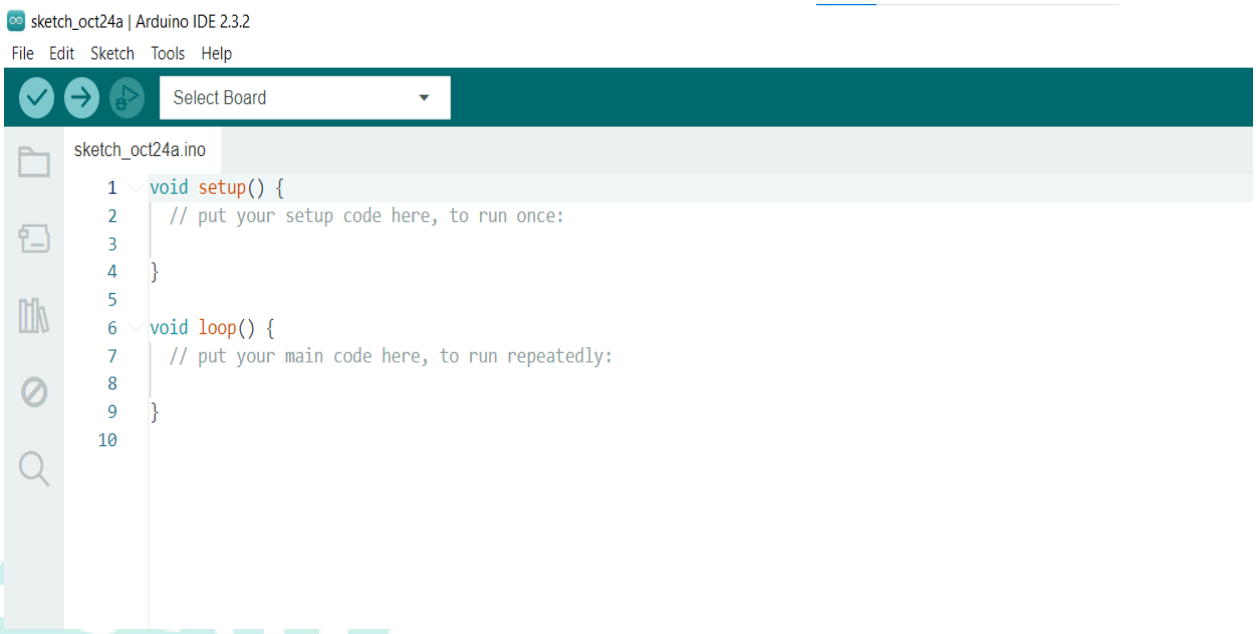


Figure 10 : Arduino IDE Interface [27]

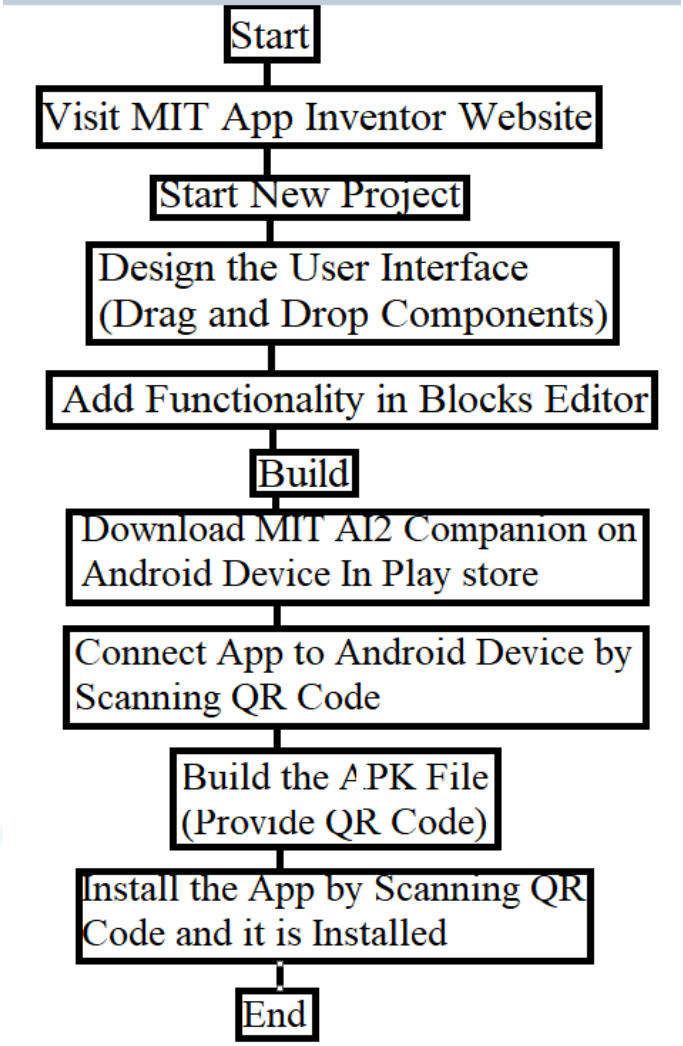
Procedure: To use the Arduino IDE, download and install it from the official Arduino website. Connect your Arduino board to your computer with a USB cable and launch the IDE. Select your board model under tools > board , and choose the appropriate port under tools > port . Open a new sketch by clicking file > now and write your code in the editor. Verify your code by clicking the checkmark (✓) icon; if there are no errors, upload it to your board by clicking the right arrow (→) icon. If your sketch uses serial communication, open the Serial Monitor by clicking the magnifying glass icon. Test your robot's behavior and make adjustments as needed, and remember to save your work under file > save to avoid losing progress.

2. **MIT App Inventor:** MIT App Inventor is a visual programming environment that enables users to create Android applications without prior coding experience. Developed by MIT, it features a drag-and-drop interface that allows users to select and customize components such as buttons and text fields to build their app's user interface. The functionality of the app is defined using a block-based programming language, which simplifies the understanding of programming concepts like loops and conditionals. A key feature of App Inventor is its ability to connect to hardware and software components, including Bluetooth devices and sensors like GPS. Users can test their apps in real-time using the MIT AI2 Companion app, providing immediate feedback during development. Once completed, apps can be packaged into APK files for easy distribution on Android devices. Overall, MIT App Inventor serves as an engaging tool for learning programming and app development, making it accessible to beginners and fostering creativity [28].

Procedure:

1. **Visit MIT App Inventor Website:** Go to MIT App Inventor and click on "Create Apps!". Sign in with your Google account or create one if needed.

2. **Start a New Project:** Once in the development environment, click "Projects", then select "Start new project" to begin creating your app.
3. **Design the User Interface:** In the design editor, drag and drop components (e.g., buttons, text fields) onto the screen to create the layout of your app.
4. **Add App Functionality**: Switch to the Blocks Editor to add functionality. Use built-in blocks to control actions like Bluetooth commands, sensors, or data display.
5. **Access the GPS Guided Robot App:** Search for the "GPS Guided Robot" app in the MIT App Inventor gallery. Open it to save a copy in your "My Projects" folder.
6. **Download MIT AI2 Companion:** On your Android device, download the MIT AI2 Companion app from the Google Play Store to connect and test your project.
7. **Connect the App to Your Device:** Open the MIT AI2 Companion app, then scan the QR code provided by the MIT App Inventor interface to connect your project for real-time testing.
8. **Test and Refine**: Run the app on your Android device to test its features and make necessary adjustments in the MIT App Inventor interface.
9. **Build the App**: Once finalized, click "Build" in MIT App Inventor and select "App (provide QR code for .apk)" to generate a barcode for the APK file.



10. **Install the App**: Use the MIT AI2 Companion to scan the QR code and install the APK on your device. Grant necessary installation permissions, and the app will be ready to use

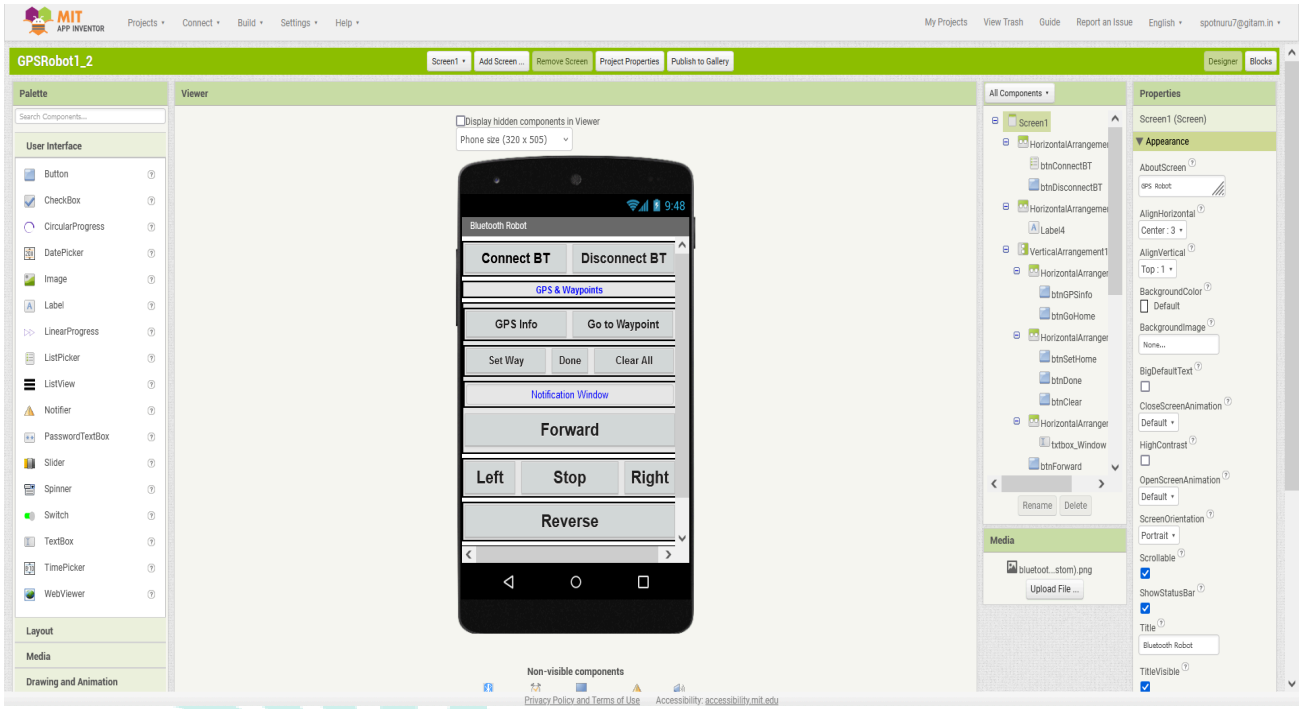


Figure 11 : Interface of MIT App inventor

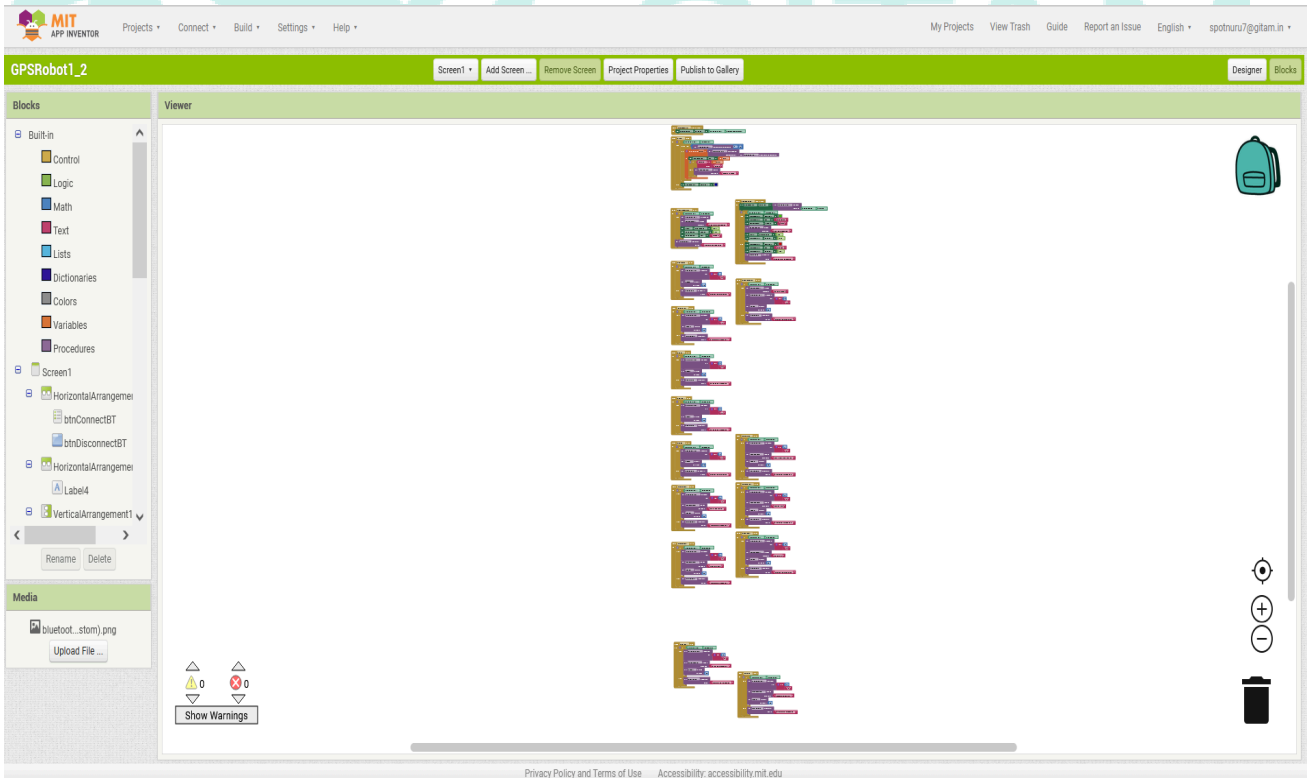


Figure 12 : Overall blocks in the MIT App

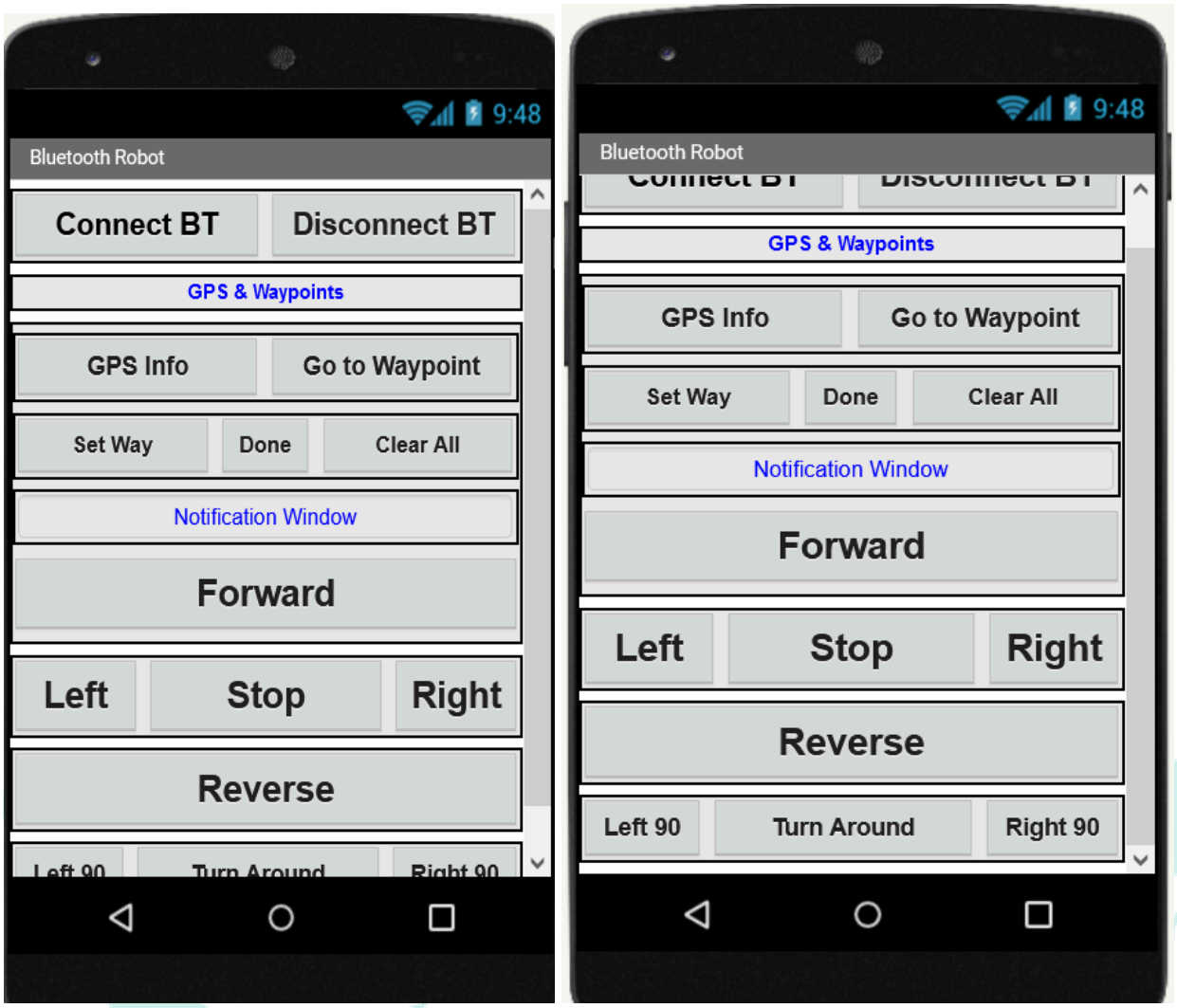
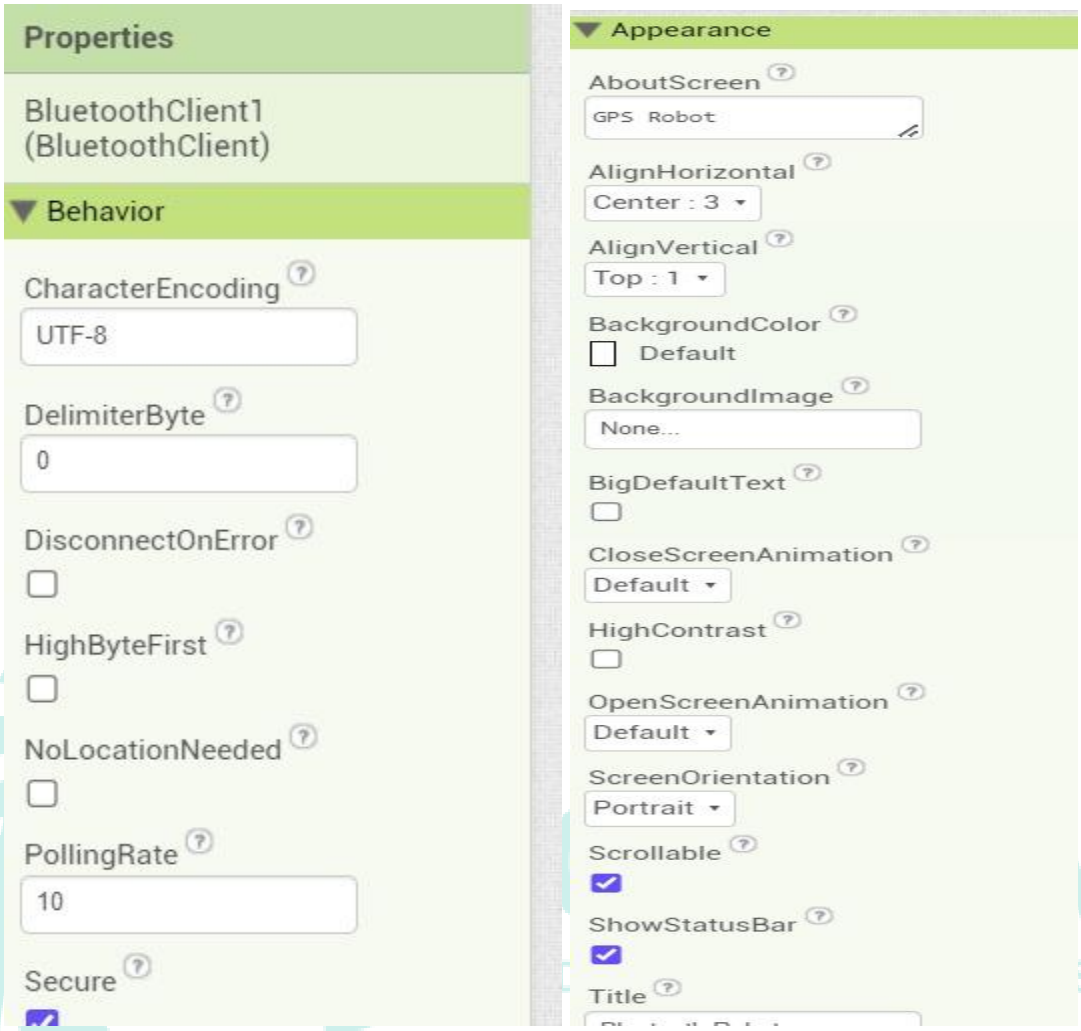
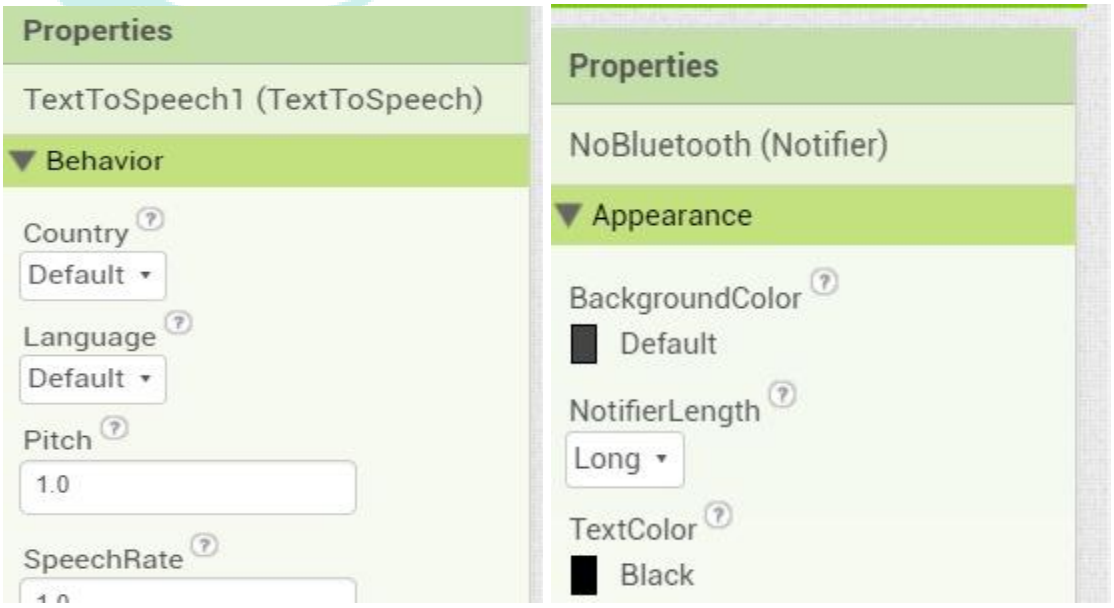


Figure 13 : Interface In MIT App Inventor


Properties
BluetoothClient1 (BluetoothClient)
Behavior
CharacterEncoding
UTF-8
DelimiterByte
0
DisconnectOnError
HighByteFirst
NoLocationNeeded
PollingRate
10
Secure
Appearance
AboutScreen
GPS Robot
AlignHorizontal
Center : 3
AlignVertical
Top : 1
BackgroundColor
Default
BackgroundImage
None...
BigDefaultText
CloseScreenAnimation
Default
HighContrast
OpenScreenAnimation
Default
ScreenOrientation
Portrait
Scrollable
ShowStatusBar
Title


Properties
TextToSpeech1 (TextToSpeech)
Behavior
Country
Default
Language
Default
Pitch
1.0
SpeechRate
Properties
NoBluetooth (Notifier)
Appearance
BackgroundColor
Default
NotifierLength
Long
TextColor
Black

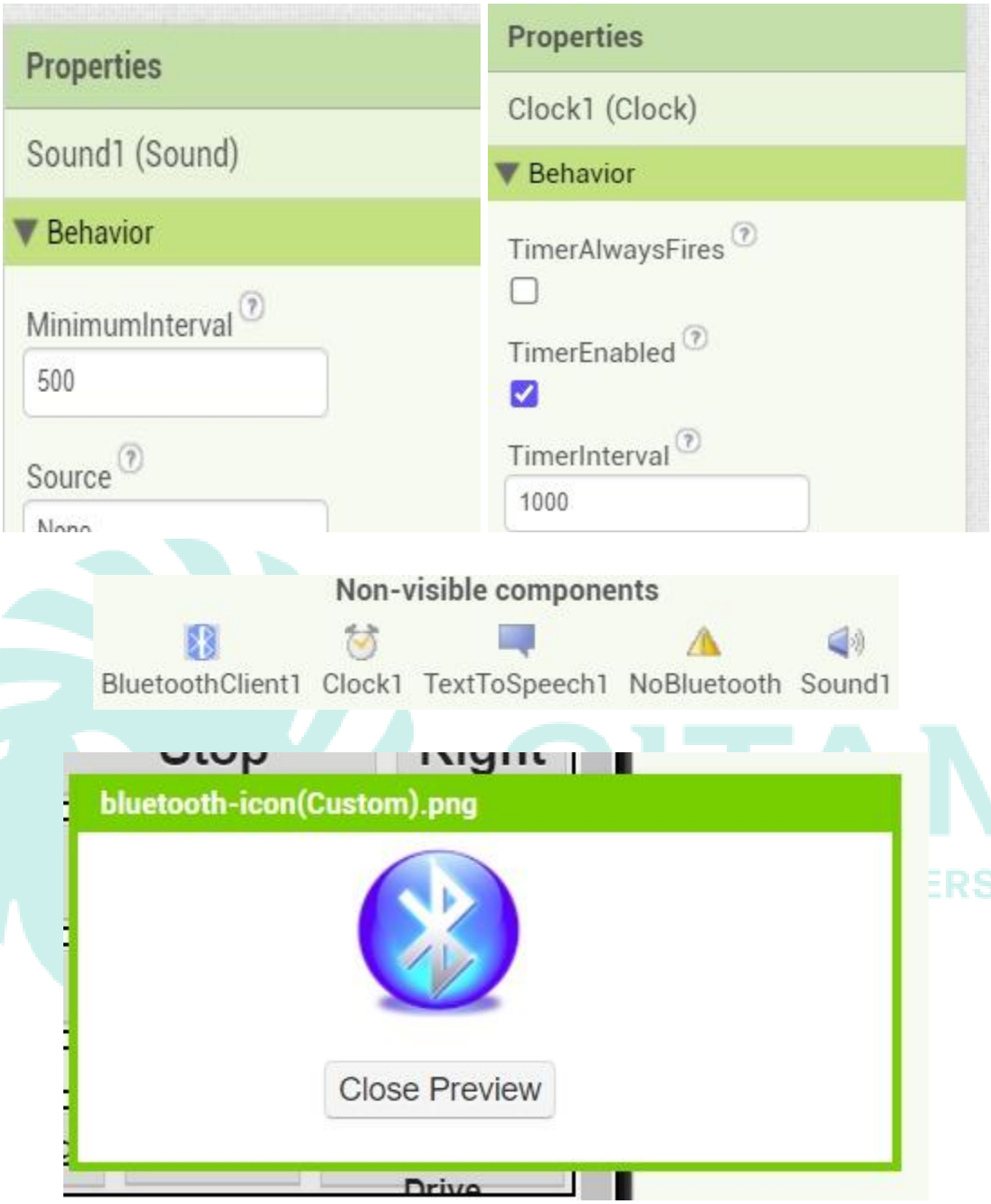


Figure 14 : Some Specifications From Blocks

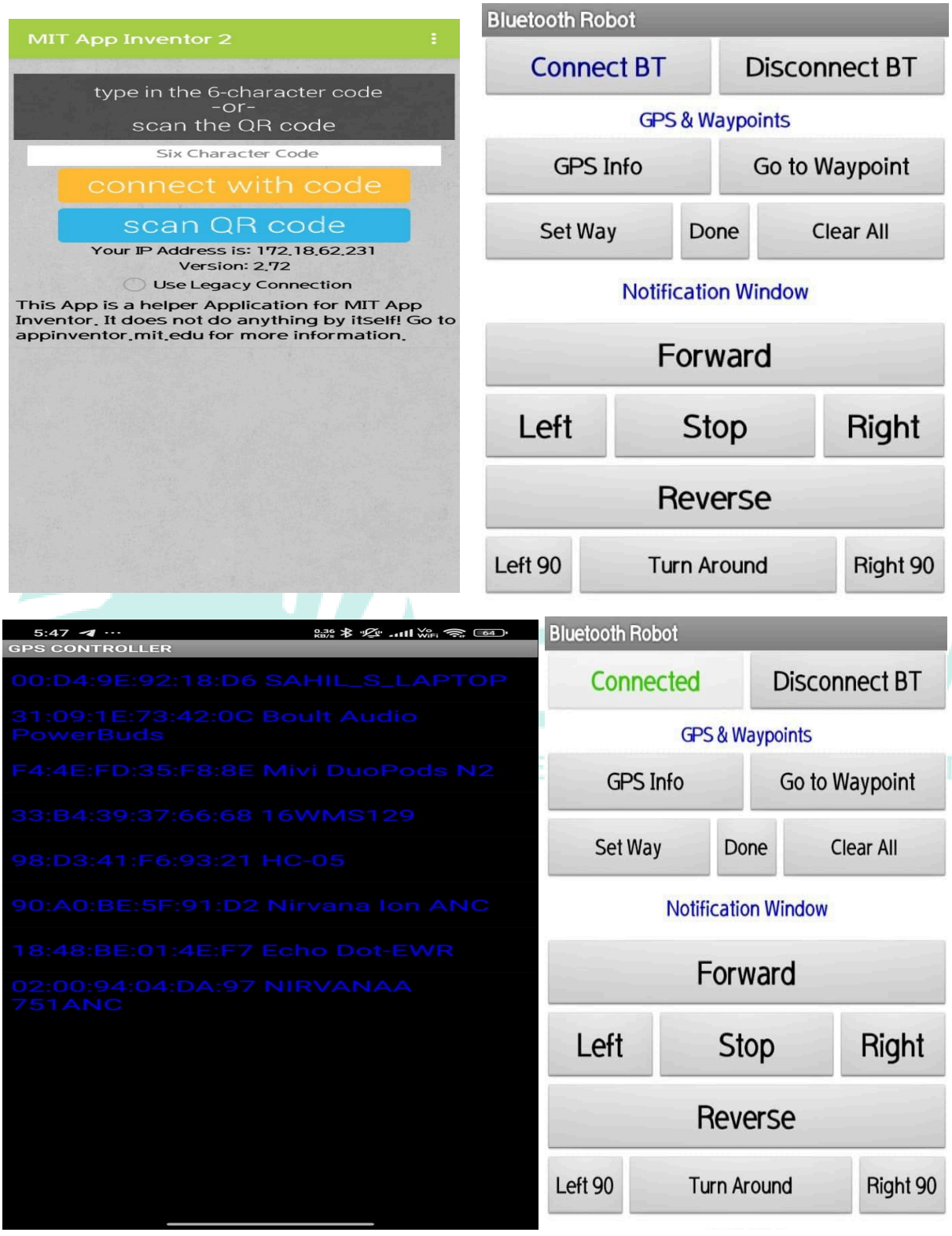


Figure 15 : Interface in the mobile phone Of An MIT APP Inventor (HC-05)

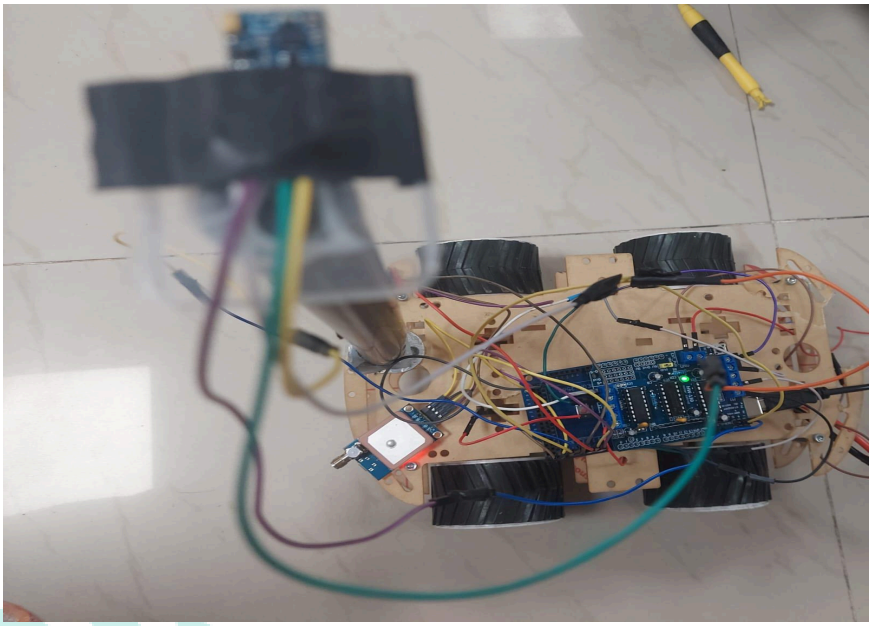


Figure 16 : Top view of GPS Guided Ground Vehicle

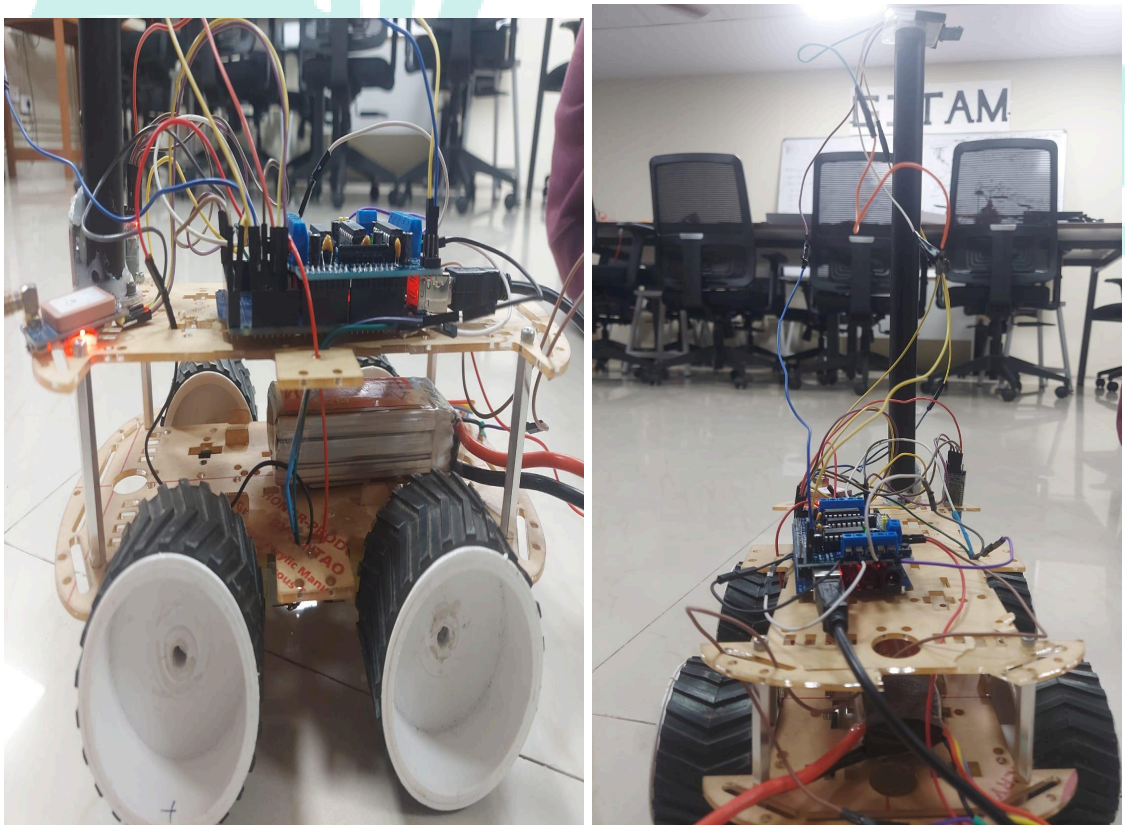
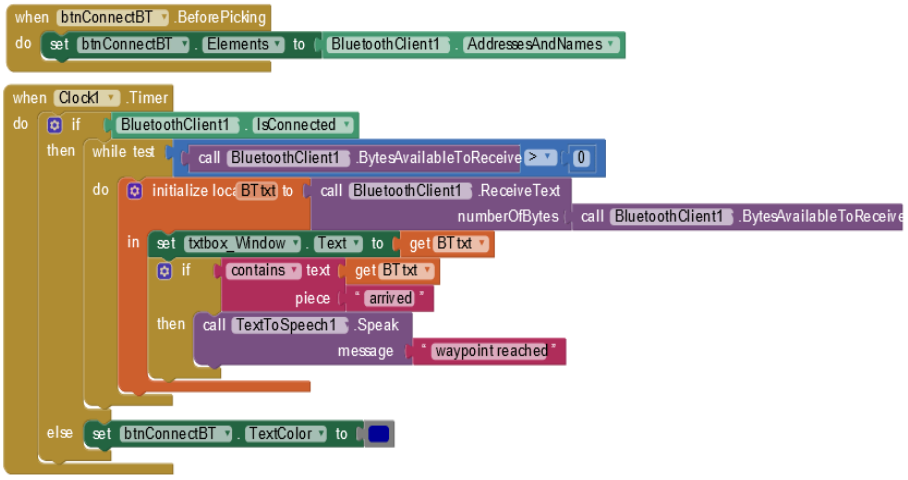
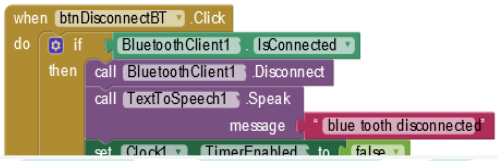
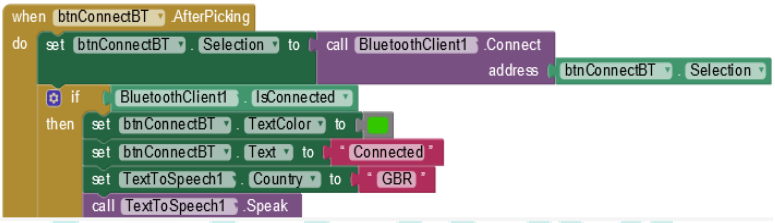
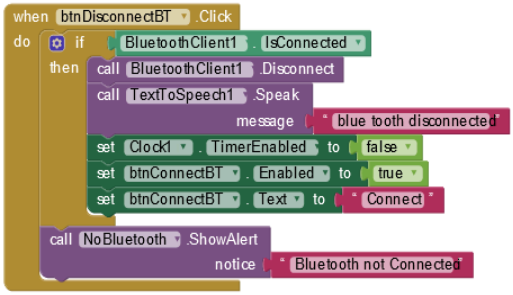


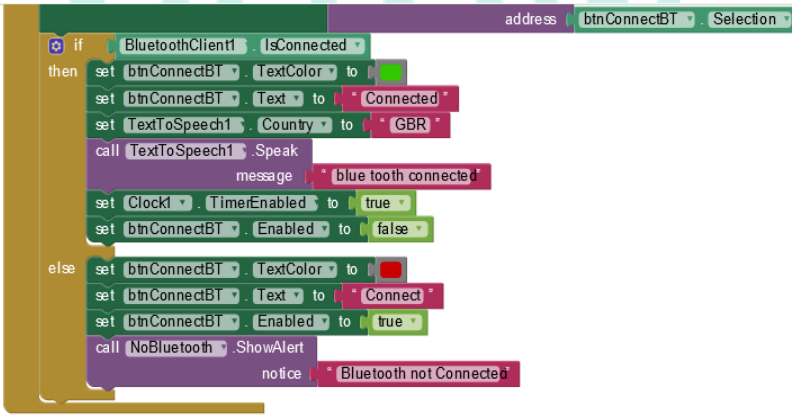
Figure 16 : Side and Front View Of GPS Guided Ground Vehicle

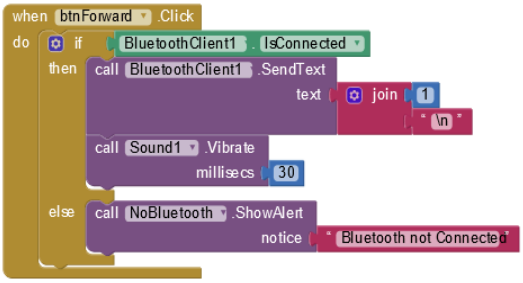

when btnConnectBT .BeforePicking
do set btnConnectBT . Elements to BluetoothClient1 . AddressesAndNames
when Clock1 .Timer
do if BluetoothClient1 . IsConnected
then while test call BluetoothClient1 .BytesAvailableToReceive > 0
do initialize local BTtxt to call BluetoothClient1 .ReceiveText
numberOfBytes call BluetoothClient1 .BytesAvailableToReceive
in set txtbox_Window . Text to get BTtxt
if contains text get BTtxt
piece " arrived "
then call TextToSpeech1 .Speak
message " waypoint reached "
else set btnConnectBT . TextColor to

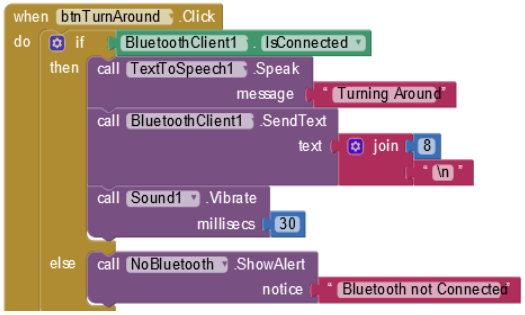

when btnDisconnectBT .Click
do if BluetoothClient1 . IsConnected
then call BluetoothClient1 .Disconnect
call TextToSpeech1 .Speak
message " blue tooth disconnected "

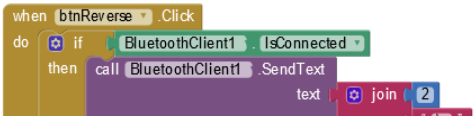

when btnConnectBT .AfterPicking
do set btnConnectBT . Selection to call BluetoothClient1 .Connect
address btnConnectBT . Selection
if BluetoothClient1 . IsConnected
then set btnConnectBT . TextColor to
set btnConnectBT . Text to " Connected "
set TextToSpeech1 . Country to " GBR "
call TextToSpeech1 .Speak

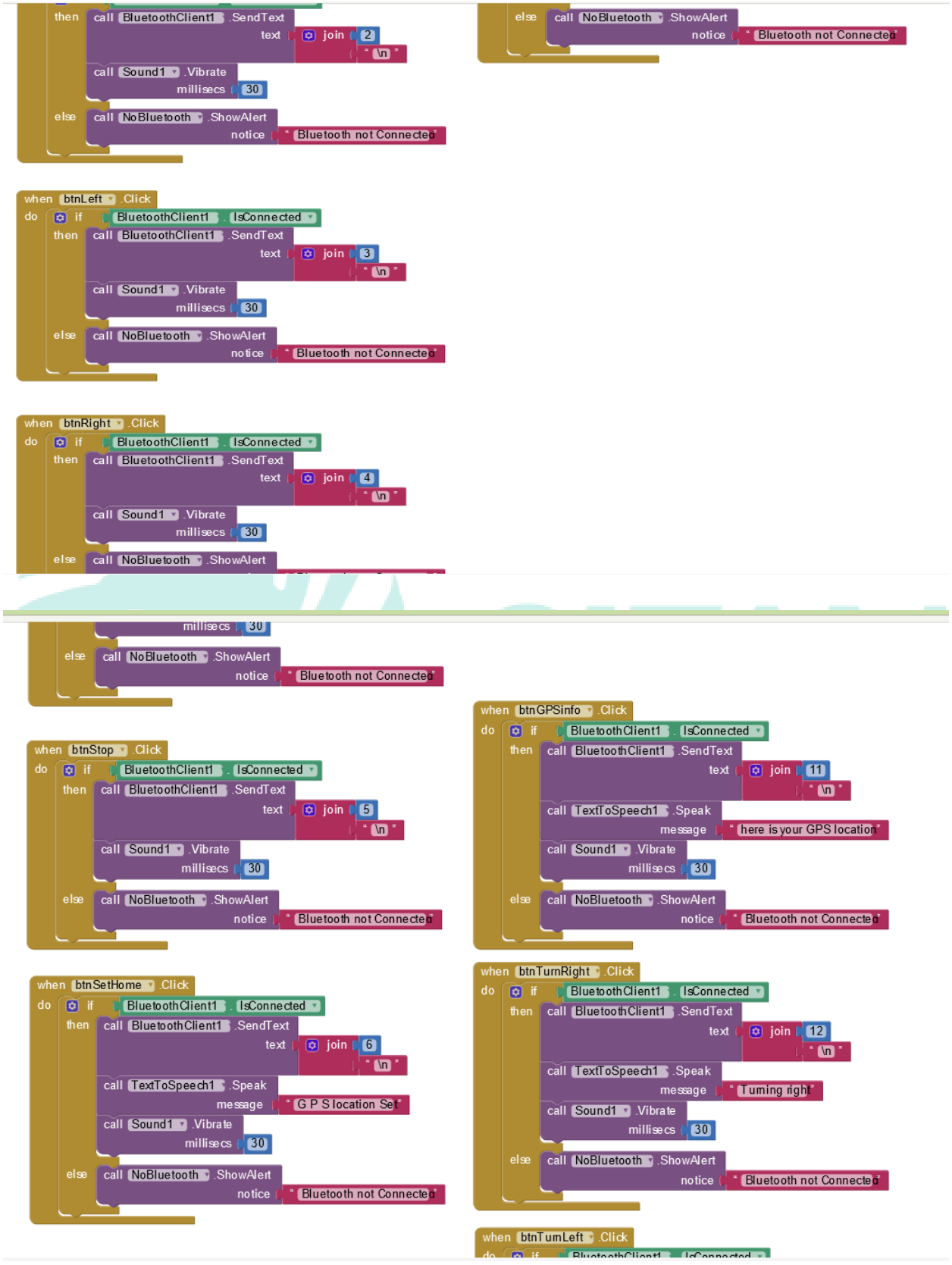

when btnDisconnectBT .Click
do if BluetoothClient1 . IsConnected
then call BluetoothClient1 .Disconnect
call TextToSpeech1 .Speak
message " blue tooth disconnected "
set Clock1 . TimerEnabled to false
set btnConnectBT . Enabled to true
set btnConnectBT . Text to " Connect "
call NoBluetooth .ShowAlert
notice " Bluetooth not Connected "


address btnConnectBT . Selection
if BluetoothClient1 . IsConnected
then set btnConnectBT . TextColor to
set btnConnectBT . Text to " Connected "
set TextToSpeech1 . Country to " GBR "
call TextToSpeech1 .Speak
message " blue tooth connected "
set Clock1 . TimerEnabled to true
set btnConnectBT . Enabled to false
else set btnConnectBT . TextColor to
set btnConnectBT . Text to " Connect "
set btnConnectBT . Enabled to true
call NoBluetooth .ShowAlert
notice " Bluetooth not Connected "


when btnForward .Click
do if BluetoothClient1 . IsConnected
then call BluetoothClient1 .SendText
text join 1
" \n "
call Sound1 .Vibrate
millisecs 30
else call NoBluetooth .ShowAlert
notice " Bluetooth not Connected "


when btnTurnAround .Click
do if BluetoothClient1 . IsConnected
then call TextToSpeech1 .Speak
message " Turning Around "
call BluetoothClient1 .SendText
text join 8
" \n "
call Sound1 .Vibrate
millisecs 30
else call NoBluetooth .ShowAlert
notice " Bluetooth not Connected "


when btnReverse .Click
do if BluetoothClient1 . IsConnected
then call BluetoothClient1 .SendText
text join 2


then call BluetoothClient1 .SendText text join 2 "\n"
call Sound1 .Vibrate millisecs 30
else call NoBluetooth .ShowAlert notice "Bluetooth not Connected"
else call NoBluetooth .ShowAlert notice "Bluetooth not Connected"
when btnLeft .Click
do if BluetoothClient1 . IsConnected
then call BluetoothClient1 .SendText text join 3 "\n"
call Sound1 .Vibrate millisecs 30
else call NoBluetooth .ShowAlert notice "Bluetooth not Connected"
when btnRight .Click
do if BluetoothClient1 . IsConnected
then call BluetoothClient1 .SendText text join 4 "\n"
call Sound1 .Vibrate millisecs 30
else call NoBluetooth .ShowAlert
millisecs 30
else call NoBluetooth .ShowAlert notice "Bluetooth not Connected"
when btnStop .Click
do if BluetoothClient1 . IsConnected
then call BluetoothClient1 .SendText text join 5 "\n"
call Sound1 .Vibrate millisecs 30
else call NoBluetooth .ShowAlert notice "Bluetooth not Connected"
when btnGPSinfo .Click
do if BluetoothClient1 . IsConnected
then call BluetoothClient1 .SendText text join 11 "\n"
call TextToSpeech1 .Speak message "here is your GPS location"
call Sound1 .Vibrate millisecs 30
else call NoBluetooth .ShowAlert notice "Bluetooth not Connected"
when btnSetHome .Click
do if BluetoothClient1 . IsConnected
then call BluetoothClient1 .SendText text join 6 "\n"
call TextToSpeech1 .Speak message "G P S location Set"
call Sound1 .Vibrate millisecs 30
else call NoBluetooth .ShowAlert notice "Bluetooth not Connected"
when btnTurnRight .Click
do if BluetoothClient1 . IsConnected
then call BluetoothClient1 .SendText text join 12 "\n"
call TextToSpeech1 .Speak message "Turning right"
call Sound1 .Vibrate millisecs 30
else call NoBluetooth .ShowAlert notice "Bluetooth not Connected"
when btnTurnLeft .Click

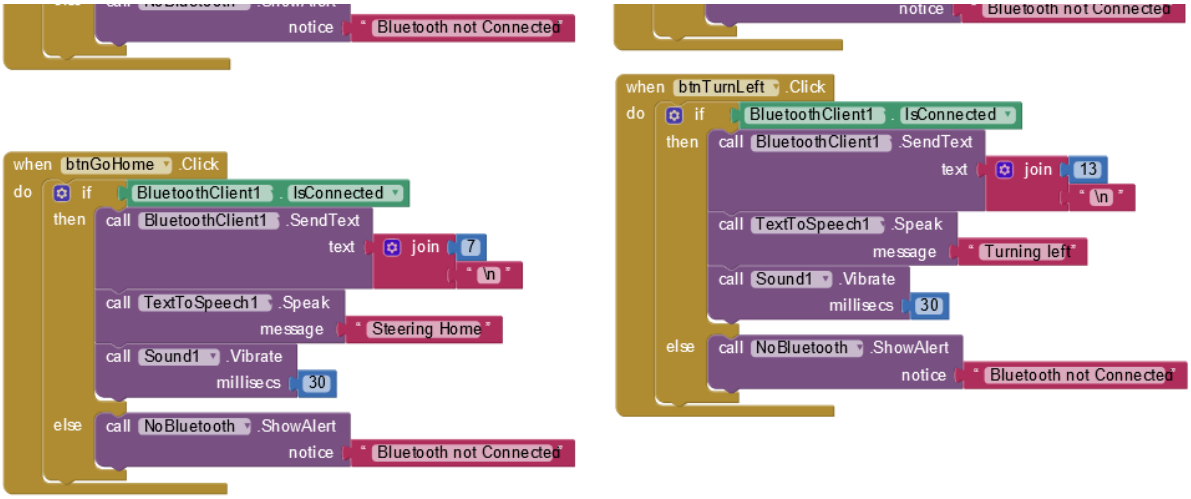


Figure 17 : Detailed View Of Blocks Of MIT APP Inventor

8. Procedure for Building a GPS-Guided Robot

1. **Assemble the Chassis:** Build the robot body using the chassis, attach wheels, and install motors.
2. **Set Up the Microcontroller:** Connect the microcontroller (e.g., Arduino Mega 2560) to your computer and install the necessary IDE (e.g., Arduino IDE).
3. **Connect the GPS Module:** Wire the GPS module to the microcontroller according to the specifications (usually using serial communication).
4. **Connect the Motor Driver:** Wire the motor driver to the microcontroller and connect it to the motors.
5. **Write the Code:** Develop a program to read data from the GPS module and control the motors based on that data. Use libraries suitable for your microcontroller (e.g., TinyGPS for Arduino).
6. **MIT App Inventor Integration:** Create a mobile app using MIT App Inventor to send commands to the robot. Use Bluetooth or Wi-Fi for communication between the app and the robot. Design the user interface to include controls for navigation, displaying GPS coordinates, and other relevant features.

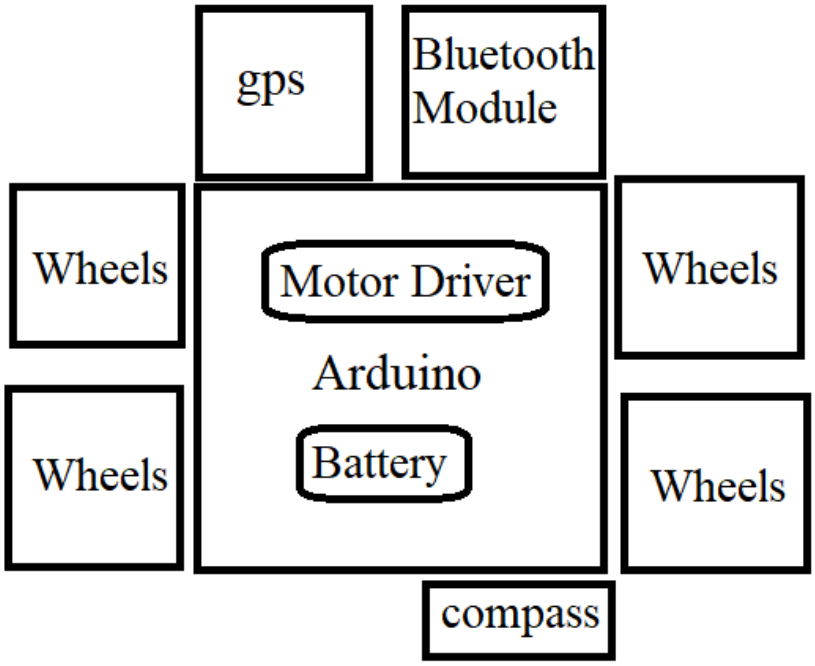


Figure 18 : Schematic Diagram Cf Connections

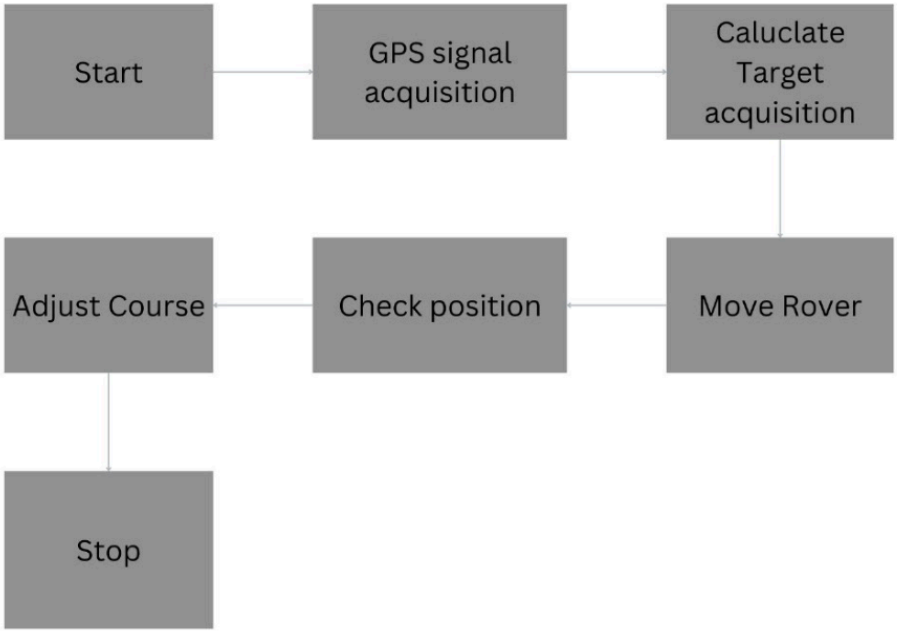


Figure 19 : Flow chart of Navigation

7. **Testing the GPS Module:** Test the GPS module independently to ensure it can get a location fix and output data correctly.

8. **Calibrate Movement:** Implement a way to translate GPS coordinates into movement commands for the motors. Calculate distance and bearing to move toward a specific GPS point.
9. **Power Management:** Ensure the power supply is adequate for all components and set up proper power management.
10. **Testing:** Test the robot in an open area, sending it to specific GPS coordinates and observing its behavior. Use the mobile app to control the robot and monitor its performance in real time.

Troubleshooting Steps:

1. **Check Connections:** Ensure all wires are secure and correctly connected.
2. **Inspect Power Supply:** Verify that all components receive adequate power and check voltage levels.
3. **GPS Module:** Ensure the GPS module has a clear view of the sky for better signal reception.
4. **Monitor Serial Output:** Use the Serial Monitor to check for valid GPS data.
5. **Review Code:** Check for errors in the code and ensure the correct libraries are included.
6. **Calibrate Movement:** Ensure calculations for distance and bearing are accurate.

Precautions:

1. **Avoid Water Exposure:** Keep all electronics dry to prevent damage.
2. **Secure Components:** Ensure all parts are firmly mounted on the chassis.
3. **Power Management:** Use appropriate power sources to avoid overload.
4. **Testing Environment:** Conduct tests in open areas for accurate GPS readings.
5. **Proper Grounding:** Ground all components to reduce electrical interference.
6. **Control Movements:** Avoid sudden movements to maintain GPS signal stability.

Final Adjustments: Fine-tune the code and make adjustments to the hardware as needed based on testing results.

9. Code:

1. **Calibrate HMC5883L**

```
/*
  Calibrate HMC5883L. Output for HMC5883L_calibrate_processing.pde
  Read more: http://www.jarzebski.pl/arduino/czujniki-i-sensory/3-osiowy-magnetometr-hmc5883l.html
  GIT: https://github.com/jarzebski/Arduino-HMC5883L
  Web: http://www.jarzebski.pl
  (c) 2014 by Korneliusz Jarzebski
*/

#include <Wire.h>
#include <HMC5883L.h>

HMC5883L compass;

int minX = 0;
int maxX = 0;
int minY = 0;
int maxY = 0;
int minZ = 0;
int maxZ = 0;
```

```
int offX = 0;
int offY = 0;
int offZ = 0;

void setup()
{
 Serial.begin(9600);

 // Initialize Initialize HMC5883L
 while (!compass.begin())
 {
        delay(500);
 }

 // Set measurement range
 compass.setRange(HMC5883L_RANGE_1_3GA);

 // Set measurement mode
 compass.setMeasurementMode(HMC5883L_CONTINOUS);

 // Set data rate
 compass.setDataRate(HMC5883L_DATARATE_30HZ);

 // Set number of samples averaged
 compass.setSamples(HMC5883L_SAMPLES_8);
}

void loop()
{
 Vector mag = compass.readRaw();

 // Determine Min / Max values
 if (mag.XAxis < minX) minX = mag.XAxis;
 if (mag.XAxis > maxX) maxX = mag.XAxis;
 if (mag.YAxis < minY) minY = mag.YAxis;
 if (mag.YAxis > maxY) maxY = mag.YAxis;
 if (mag.ZAxis < minZ) minZ = mag.ZAxis;
 if (mag.ZAxis > maxZ) maxZ = mag.ZAxis;

 // Calculate offsets
 offX = (maxX + minX)/2;
 offY = (maxY + minY)/2;
 offZ = (maxZ + minZ)/2;



 Serial.print(offX);
 Serial.print(":");
 Serial.print(offY);
 Serial.print(":");
 Serial.print(offZ);
```

```
  Serial.print("\n");
}
```

2. HMC5883L Compass

```
/*
HMC5883L.h - Header file for the HMC5883L Triple Axis Digital Compass Arduino Library.

Version: 1.1.0
(c) 2014 Korneliusz Jarzebski
www.jarzebski.pl

This program is free software: you can redistribute it and/or modify
it under the terms of the version 3 GNU General Public License as
published by the Free Software Foundation.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#define HMC5883L_h
#define HMC5883L_h

#if ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif

#define HMC5883L_ADDRESS           (0x1E)
#define HMC5883L_REG_CONFIG_A      (0x00)
#define HMC5883L_REG_CONFIG_B      (0x01)
#define HMC5883L_REG_MODE          (0x02)
#define HMC5883L_REG_OUT_X_M       (0x03)
#define HMC5883L_REG_OUT_X_L       (0x04)
#define HMC5883L_REG_OUT_Z_M       (0x05)
#define HMC5883L_REG_OUT_Z_L       (0x06)
#define HMC5883L_REG_OUT_Y_M       (0x07)
#define HMC5883L_REG_OUT_Y_L       (0x08)
#define HMC5883L_REG_STATUS        (0x09)
#define HMC5883L_REG_IDENT_A       (0x0A)
#define HMC5883L_REG_IDENT_B       (0x0B)
#define HMC5883L_REG_IDENT_C       (0x0C)

typedef enum
{
        HMC5883L_SAMPLES_8      = 0b11,
```

```
        HMC5883L_SAMPLES_4      = 0b10,
        HMC5883L_SAMPLES_2      = 0b01,
        HMC5883L_SAMPLES_1      = 0b00
} hmc5883l_samples_t;

typedef enum
{
        HMC5883L_DATARATE_75HZ          = 0b110,
        HMC5883L_DATARATE_30HZ          = 0b101,
        HMC5883L_DATARATE_15HZ          = 0b100,
        HMC5883L_DATARATE_7_5HZ         = 0b011,
        HMC5883L_DATARATE_3HZ           = 0b010,
        HMC5883L_DATARATE_1_5HZ         = 0b001,
        HMC5883L_DATARATE_0_75_HZ       = 0b000
} hmc5883l_dataRate_t;

typedef enum
{
        HMC5883L_RANGE_8_1GA    = 0b111,
        HMC5883L_RANGE_5_6GA    = 0b110,
        HMC5883L_RANGE_4_7GA    = 0b101,
        HMC5883L_RANGE_4GA      = 0b100,
        HMC5883L_RANGE_2_5GA    = 0b011,
        HMC5883L_RANGE_1_9GA    = 0b010,
        HMC5883L_RANGE_1_3GA    = 0b001,
        HMC5883L_RANGE_0_88GA = 0b000
} hmc5883l_range_t;

typedef enum
{
        HMC5883L_IDLE       = 0b10,
        HMC5883L_SINGLE             = 0b01,
        HMC5883L_CONTINOUS      = 0b00
} hmc5883l_mode_t;

#ifndef VECTOR_STRUCT_H
#define VECTOR_STRUCT_H
struct Vector
{
        float XAxis;
        float YAxis;
        float ZAxis;
};
#endif

class HMC5883L
{
        public:

        bool begin(void);
```

```
        Vector readRaw(void);
        Vector readNormalize(void);

        void  setOffset(int xo, int yo, int zo);

        void  setRange(hmc5883l_range_t range);
        hmc5883l_range_t getRange(void);

        void  setMeasurementMode(hmc5883l_mode_t mode);
        hmc5883l_mode_t getMeasurementMode(void);

        void  setDataRate(hmc5883l_dataRate_t dataRate);
        hmc5883l_dataRate_t getDataRate(void);

        void  setSamples(hmc5883l_samples_t samples);
        hmc5883l_samples_t getSamples(void);

        Vector selfTest();

        private:

        float mgPerDigit;
        Vector v;
        int xOffset, yOffset, zOffset;

        void writeRegister8(uint8_t reg, uint8_t value);
        uint8_t readRegister8(uint8_t reg);
        uint8_t fastRegister8(uint8_t reg);
        int16_t readRegister16(uint8_t reg);
};

#endif
```


3. Sweep

```
void sweep()                    // Can be used to simulate Metal Detecting or to rotate Ping Sensor
{
 myservo.attach(9);
 StopCar();
 Forward_Meter();
 StopCar();

 for(pos = 60; pos <= 120; pos += 1) // goes from 0 degrees to 180 degrees
 {                          // in steps of 1 degree
        myservo.write(pos);             // tell servo to go to position in variable 'pos'
        delay(15);              // waits 15ms for the servo to reach the position
 }
 for(pos = 120; pos>=60; pos-=1)        // goes from 180 degrees to 0 degrees
 {
        myservo.write(pos);             // tell servo to go to position in variable 'pos'
        delay(15);              // waits 15ms for the servo to reach the position
```

```
  }

        myservo.write(90);              // tell servo to go to position in variable 'pos'
        delay(15);                      // waits 15ms for the servo to reach the position

   myservo.detach();                    // Disengage Servo Motor
}
```

4. Steering

```
void Forward()
{
  Ping();
  Serial.println("Forward");
  motor1.setSpeed(mtr_Spd);
  motor2.setSpeed(mtr_Spd);
  motor3.setSpeed(mtr_Spd);
  motor4.setSpeed(mtr_Spd);

  motor1.run(FORWARD);                                  // go forward all wheels
  motor2.run(FORWARD);
  motor3.run(FORWARD);
  motor4.run(FORWARD);
}
//
**********************************************************************************
**************************************************************************
void Forward_Meter()
{
  motor1.setSpeed(mtr_Spd);
  motor2.setSpeed(mtr_Spd);
  motor3.setSpeed(mtr_Spd);
  motor4.setSpeed(mtr_Spd);

  motor1.run(FORWARD);                                  // go forward all wheels for specified time
  motor2.run(FORWARD);
  motor3.run(FORWARD);
  motor4.run(FORWARD);
  delay(500);
}

//
**********************************************************************************
**************************************************************************

void Reverse()
{
  motor1.setSpeed(mtr_Spd);
  motor2.setSpeed(mtr_Spd);
```

```
  motor3.setSpeed(mtr_Spd);
  motor4.setSpeed(mtr_Spd);

  motor1.run(BACKWARD);                                  // Reverse all wheels
  motor2.run(BACKWARD);
  motor3.run(BACKWARD);
  motor4.run(BACKWARD);
}

//
*********************************************************************************
*********************************************************************************

void LeftTurn()
{
  motor1.setSpeed(mtr_Spd);
  motor2.setSpeed(mtr_Spd);
  motor3.setSpeed(mtr_Spd);
  motor4.setSpeed(mtr_Spd);

  motor1.run(BACKWARD);                                  // Turn left
  motor2.run(FORWARD);
  motor3.run(FORWARD);
  motor4.run(BACKWARD);
  delay(100);
}

//
*********************************************************************************
*********************************************************************************

void RightTurn()
{
  motor1.setSpeed(mtr_Spd);
  motor2.setSpeed(mtr_Spd);
  motor3.setSpeed(mtr_Spd);
  motor4.setSpeed(mtr_Spd);

  motor1.run(FORWARD);
  motor2.run(BACKWARD);
  motor3.run(BACKWARD);
  motor4.run(FORWARD);
  delay(100);                                    //delay for 100ms more responsive turn per push on
bluetooth
}

//
*********************************************************************************
*********************************************************************************
```

```
void SlowLeftTurn()
{

  motor1.setSpeed(turn_Speed);
  motor2.setSpeed(turn_Speed);
  motor3.setSpeed(turn_Speed);
  motor4.setSpeed(turn_Speed);

  motor1.run(BACKWARD);
  motor2.run(FORWARD);
  motor3.run(FORWARD);
  motor4.run(BACKWARD);
}

//
**********************************************************************************
**********************************************************************************

// Turning too fast will over-shoot the compass and GPS course


void SlowRightTurn()
{

  motor1.setSpeed(turn_Speed);
  motor2.setSpeed(turn_Speed);
  motor3.setSpeed(turn_Speed);
  motor4.setSpeed(turn_Speed);

  motor1.run(FORWARD);
  motor2.run(BACKWARD);
  motor3.run(BACKWARD);
  motor4.run(FORWARD);
}

//
**********************************************************************************
**********************************************************************************

void StopCar()
{


  motor1.run(RELEASE);
  motor2.run(RELEASE);
  motor3.run(RELEASE);
  motor4.run(RELEASE);

}
```

```
// Compass Drive Section
// This Portion of code steers the Vehicle based on the compass heading
//
*********************************************************************************
*********************************************************************************

void CompassTurnRight()                                    // This Function Turns the car 90 degrees to
the right based on the current desired heading
{
  StopCar();
  getCompass();                                            // get current compass heading

  desired_heading = (desired_heading + 90);                          // set desired_heading to plus 90
degrees
  if (desired_heading >= 360) {desired_heading = (desired_heading - 360);}       // if the desired heading is
greater than 360 then subtract 360 from it

  while ( abs(desired_heading - compass_heading) >= compass_dev)              // If the desired heading is
more than Compass Deviation in degrees from the actual compass heading then
     {                                                     // correct direction by turning left or right

    getCompass();                                          // Update compass heading during While Loop
    bluetooth();                                          // if new bluetooth value received break from loop
    if (blueToothVal == 5){break;}                                 // If a Stop Bluetooth command ('5') is
received then break from the Loop

    if (desired_heading >= 360) {desired_heading = (desired_heading - 360);}     // if the desired heading is
greater than 360 then subtract 360 from it

    int x = (desired_heading - 359);                               // x = the GPS desired heading - 360
    int y = (compass_heading - (x));                               // y = the Compass heading - x
    int z = (y - 360);                                    // z = y - 360

       if ((z <= 180) && (z >= 0))                                 // if z is less than 180 and not a negative
value then turn left
          {                                                // otherwise turn right
           SlowLeftTurn();
          }
          else
          {
           SlowRightTurn();
          }
       }
    {
     StopCar();                                            // Stop the Car when desired heading and compass
heading match
    }
 }
```

```
//
**********************************************************************************
**********************************************************************************

void CompassTurnLeft()                                  // This procedure turns the car 90 degrees to
the left based on the current desired heading
{
 StopCar();
 getCompass();                                          // get current compass heading
 //desired_heading = (compass_heading - 90);                    // set desired_heading to compass
value minus 90 degrees

 desired_heading = (desired_heading - 90);                    // set desired_heading to minus 90
degrees
 if (desired_heading <= 0) {desired_heading = (desired_heading + 360);}        // if the desired heading is
greater than 360 then add 360 to it
 while ( abs(desired_heading - compass_heading) >= compass_dev)              // If the desired heading is
more than Compass Deviation in degrees from the actual compass heading then
    {                                                   // correct direction by turning left or right
   getCompass();                                            // Get compass heading again during While Loop
   bluetooth();                                         // if new bluetooth value received break from loop
   if (blueToothVal == 5){break;}                               // If a 'Stop' Bluetooth command is
received then break from the Loop

   if (desired_heading >= 360) {desired_heading = (desired_heading - 360);}     // if the desired heading is
greater than 360 then subtract 360 from it

   int x = (desired_heading - 359);                             // x = the desired heading - 360
   int y = (compass_heading - (x));                             // y = the Compass heading - x
   int z = (y - 360);                                   // z = y - 360
      if (z <= 180)                                     // if z is less than 180 and not a negative value then
turn left
    // if ((z <= 180) && (z >= 0))
        {                                               // otherwise turn right
         SlowLeftTurn();
        }
        else
        {
         SlowRightTurn();
        }
      }
  {
    StopCar();                                          // Stop the Car when desired heading and compass
heading match
   }
 }

//
**********************************************************************************
**********************************************************************************
```

```
void Compass_Forward()
{
  while (blueToothVal == 9)                          // Go forward until Bluetooth 'Stop' command is
sent

  //while (true)
    {
    getCompass();                                    // Update Compass Heading
    bluetooth();                                     // Check to see if any Bluetooth commands have been
sent
    if (blueToothVal == 5) {break;}                  // If a Stop Bluetooth command ('5') is received
then break from the Loop

    if ( abs(desired_heading - compass_heading) <= compass_dev )      // If the Desired Heading and the
Compass Heading are within the compass deviation, X degrees of each other then Go Forward
                                                     // otherwise find the shortest turn radius and turn left or right
      {
       Forward();
       Ping();
      } else
        {
          int x = (desired_heading - 359);           // x = the GPS desired heading - 360
          int y = (compass_heading - (x));           // y = the Compass heading - x
          int z = (y - 360);                         // z = y - 360

          if ((z <= 180) && (z >= 0))                // if z is less than 180 and not a negative value then
turn left
          {                                          // otherwise turn right
            SlowLeftTurn();
            Ping();
          }
          else
          {
            SlowRightTurn();
            Ping();
          }
      }
 }                                                   // End While Loop
}                                                    // End Compass_Forward

//
*****************************************************************************
*****************************************************************************

void turnAround()                                    // This procedure turns the Car around 180 degrees,
every time the "Turn Around" button is pressed
 {                                                   // the car alternates whether the next turn will be to the left or
right - this is determined by the 'pass' variable
                                                     // Imagine you are cutting the grass, when you get to the end of
the row - the first pass you are turning one way and on the next pass you turn the opposite
   if (pass == 0) { CompassTurnRight(); }            // If this is the first pass then turn right
```

```
   else { CompassTurnLeft(); }                    // If this is the second pass then turn left

   //Forward_Meter();                             // Go forward one meter (approximately)
   StopCar();                                     // Stop the car


   if (pass == 0)                                 // If this is the first pass then Turn Right
     {
      CompassTurnRight();                         // Turn right
      pass = 1 ;                                  // Change the pass value to '1' so that the next turn will be to
the left
     }

   else
     {

   if (desired_heading == Heading_A)              // This section of code Alternates the desired
heading 180 degrees
    {                                             // for the Compass drive forward
     desired_heading = Heading_B;
    }
   else if (desired_heading == Heading_B)
    {
     desired_heading = Heading_A;
    }

      CompassTurnLeft();                          // If this is the second pass then Turn Left
      pass = 0;                                   // Change the pass value to '0' so that the next turn will be to
the right

     }

  Compass_Forward();                             // Maintain the 'desired heading' and drive forward
}
```

5. Startup

```
void Startup()
{
    myservo.detach();
    Serial.println("Pause for Startup... ");

    for (int i=5; i >= 1; i--)                   // Count down for X seconds
     {
      Serial1.print("Pause for Startup... ");
      Serial1.print(i);
      delay(1000);                               // Delay for X seconds
     }
```

```
  Serial1.println("Searching for Satellites ");
  Serial.println("Searching for Satellites ");

  while (Number_of_SATS <= 4)                    // Wait until x number of satellites are acquired before
starting main loop
  {
    getGPS();                                    // Update gps data
    Number_of_SATS = (int)(gps.satellites.value());   // Query Tiny GPS for the number of Satellites
Acquired
    bluetooth();                                 // Check to see if there are any bluetooth commands being received
  }
  setWaypoint();                                 // set intial waypoint to current location
  wpCount = 0;                                   // zero waypoint counter
  ac = 0;                                        // zero array counter

  Serial1.print(Number_of_SATS);
  Serial1.print(" Satellites Acquired");
}
```

6. Go to the way point

```
void goWaypoint()
{
 Serial1.println("Go to Waypoint");
//Serial.print("Home_Latarray ");
//Serial.print(Home_LATarray[ac],6);
//Serial.print(" ");
//Serial.println(Home_LONarray[ac],6);

//Serial1.print("Distance to Home");
//Serial1.print(Distance_To_Home);

//Serial1.print("ac= ");
//Serial1.print(ac);

 while (true)
  {                                              // Start of Go_Home procedure
  bluetooth();                                   // Run the Bluetooth procedure to see if there is any data
being sent via BT
  if (blueToothVal == 5){break;}                 // If a 'Stop' Bluetooth command is received then
break from the Loop
  getCompass();                                  // Update Compass heading
  getGPS();                                      // Tiny GPS function that retrieves GPS data - update GPS
location// delay time changed from 100 to 10

  if (millis() > 5000 && gps.charsProcessed() < 10)          // If no Data from GPS within 5 seconds
then send error
    Serial1.println(F("No GPS data: check wiring"));
```

```
  Distance_To_Home =
TinyGPSPlus::distanceBetween(gps.location.lat(),gps.location.lng(),Home_LATarray[ac],
Home_LONarray[ac]);  //Query Tiny GPS for Distance to Destination
  GPS_Course =
TinyGPSPlus::courseTo(gps.location.lat(),gps.location.lng(),Home_LATarray[ac],Home_LONarray[ac]);
//Query Tiny GPS for Course to Destination

   /*
    if (Home_LATarray[ac] == 0) {
      Serial1.print("End of Waypoints");
      StopCar();
      break;
      }
   */
    if (Distance_To_Home == 0)                          // If the Vehicle has reached it's Destination, then
Stop
        {
        StopCar();                                    // Stop the robot after each waypoint is reached
        Serial1.println("You have arrived!");                 // Print to Bluetooth device - "You have arrived"
        ac++;                                       // increment counter for next waypoint
        break;                                      // Break from Go_Home procedure and send control back to
the Void Loop
                                                    // go to next waypoint

        }


   if ( abs(GPS_Course - compass_heading) <= 15)              // If GPS Course and the Compass Heading
are within x degrees of each other then go Forward
                                                    // otherwise find the shortest turn radius and turn left or right
       {
        Forward();                                  // Go Forward
       } else
         {
           int x = (GPS_Course - 360);                      // x = the GPS desired heading - 360
           int y = (compass_heading - (x));                 // y = the Compass heading - x
           int z = (y - 360);                         // z = y - 360

           if ((z <= 180) && (z >= 0))                      // if z is less than 180 and not a negative value then
turn left otherwise turn right
                { SlowLeftTurn();  }
            else { SlowRightTurn(); }
       }


  }                                                 // End of While Loop


}                                                   // End of Go_Home procedure
```

7. GPS compass

```
void calibrateCompass()                          // Experimental Use Only to Calibrate Magnetometer/
Compass
{
 int minX = 0;
 int maxX = 0;
 int minY = 0;
 int maxY = 0;
 int offX = 0;
 int offY = 0;

 for (int i=1000; i >= 1; i--)
 {
 Vector mag = compass.readRaw();                  // Read compass data

 // Determine Min / Max values
 if (mag.XAxis < minX) minX = mag.XAxis;
 if (mag.XAxis > maxX) maxX = mag.XAxis;
 if (mag.YAxis < minY) minY = mag.YAxis;
 if (mag.YAxis > maxY) maxY = mag.YAxis;

 offX = (maxX + minX)/2;                          // Calculate offsets
 offY = (maxY + minY)/2;

 delay(10);
 //Serial.print("Compass X & Y offset: ");
 //Serial.print(offX);
 //Serial.print(" ");
 //Serial.print(offY);
 //Serial.print("\n");

 }                                                // end of for loop

 StopCar();

 Serial1.print("Compass X & Y offset: ");
 Serial1.print(offX);
 Serial1.print(" ");
 Serial1.print(offY);
 Serial.print("\n");
 compass.setOffset(-35,-157,89);                  // Set calibration offset
}

 //
**********************************************************************************
**********************************************************

void getGPS()                                    // Get Latest GPS coordinates
{
```

```
  while (Serial2.available() > 0)
  gps.encode(Serial2.read());
}

//
**************************************************************************************
************************************************************

void setWaypoint()                              // Set up to 5 GPS waypoints
{

//if ((wpCount >= 0) && (wpCount < 50))
if (wpCount >= 0)
 {
  Serial1.print("GPS Waypoint ");
  Serial1.print(wpCount + 1);
  Serial1.print(" Set ");
  getGPS();                                   // get the latest GPS coordinates
  getCompass();                                 // update latest compass heading

  Home_LATarray[ac] = gps.location.lat();             // store waypoint in an array
  Home_LONarray[ac] = gps.location.lng();               // store waypoint in an array

  Serial.print("Waypoint #1: ");
  Serial.print(Home_LATarray[0],8);
  Serial.print(" ");
  Serial.println(Home_LONarray[0],8);
  Serial.print("Waypoint #2: ");
  Serial.print(Home_LATarray[1],8);
  Serial.print(" ");
  Serial.println(Home_LONarray[1],8);
  Serial.print("Waypoint #3: ");
  Serial.print(Home_LATarray[2],8);
  Serial.print(" ");
  Serial.println(Home_LONarray[2],8);
  Serial.print("Waypoint #4: ");
  Serial.print(Home_LATarray[3],8);
  Serial.print(" ");
  Serial.println(Home_LONarray[3],8);
  Serial.print("Waypoint #5: ");
  Serial.print(Home_LATarray[4],8);
  Serial.print(" ");
  Serial.println(Home_LONarray[4],8);

  wpCount++;                                    // increment waypoint counter
  ac++;                                       // increment array counter

 }
 else {Serial1.print("Waypoints Full");}
}
```

```
//
**********************************************************************************
**********************************************************

void clearWaypoints()
{
  memset(Home_LATarray, 0, sizeof(Home_LATarray));            // clear the array
  memset(Home_LONarray, 0, sizeof(Home_LONarray));            // clear the array
  wpCount = 0;                                        // reset increment counter to 0
  ac = 0;

  Serial1.print("GPS Waypoints Cleared");                    // display waypoints cleared

}

 //
**********************************************************************************
**********************************************************

void getCompass()                                     // get latest compass value
 {

  Vector norm = compass.readNormalize();

  // Calculate heading
  float heading = atan2(norm.YAxis, norm.XAxis);

  if(heading < 0)
    heading += 2 * M_PI;
  compass_heading = (int)(heading * 180/M_PI);                // assign compass calculation to variable
(compass_heading) and convert to integer to remove decimal places

 }

 //
**********************************************************************************
**********************************************************

void setHeading()
                                          // This procedure will set the current heading and the Heading(s)
of the robot going away and returning using opposing degrees from 0 to 360;
                                          // for instance, if the car is leaving on a 0 degree path (North), it
will return on a 180 degree path (South)
{
  for (int i=0; i <= 5; i++)                          // Take several readings from the compass to insure
accuracy
     {
       getCompass();                                  // get the current compass heading
     }
```

```
   desired_heading = compass_heading;                    // set the desired heading to equal the current
compass heading
   Heading_A = compass_heading;                         // Set Heading A to current compass
   Heading_B = compass_heading + 180;                     // Set Heading B to current compass heading +
180

     if (Heading_B >= 360)                              // if the heading is greater than 360 then subtract 360
from it, heading must be between 0 and 360
       {
        Heading_B = Heading_B - 360;
       }

   Serial1.print("Compass Heading Set: ");
   Serial1.print(compass_heading);
   Serial1.print(" Degrees");

   Serial.print("desired heading");
   Serial.println(desired_heading);
   Serial.print("compass heading");
   Serial.println(compass_heading);

}

//
**********************************************************************************
**********************************************************

void gpsInfo()                                  // displays Satellite data to user
 {
      Number_of_SATS = (int)(gps.satellites.value());        //Query Tiny GPS for the number of Satellites
Acquired
      Distance_To_Home =
TinyGPSPlus::distanceBetween(gps.location.lat(),gps.location.lng(),Home_LATarray[ac],
Home_LONarray[ac]);  //Query Tiny GPS for Distance to Destination
      Serial1.print("Lat:");
      Serial1.print(gps.location.lat(),8);
      Serial1.print(" Lon:");
      Serial1.print(gps.location.lng(),8);
      Serial1.print(" ");
      Serial1.print(Number_of_SATS);
      Serial1.print(" SATs ");
      Serial1.print(Distance_To_Home);
      Serial1.print("m");
Serial.print("Distance to Home ");
Serial.println(Distance_To_Home);

 }
```

8. collision avoidance

```
void Ping()
{
```

```
 currentMillis = millis();

 if ((currentMillis - previousMillis >= interval) && (pingOn == true))
 {
 previousMillis = currentMillis;
 digitalWrite(trigPin, LOW);
 delayMicroseconds(5);
 digitalWrite(trigPin, HIGH);
 delayMicroseconds(10);
 digitalWrite(trigPin, LOW);

 pinMode(echoPin, INPUT);
 duration = pulseIn(echoPin, HIGH);

 int inches = (duration / 2) / 74;                    // convert the time into a distance
 Ping_distance == inches;

   if ((inches < 12) && (blueToothVal != 5))
     {
      Serial1.print("Crash! ");
      Serial1.println(inches);
      Reverse();                                  // Quick reverse to Stop quickly
      delay(100);
      StopCar();
      blueToothVal = 5;                           // Set bluetooth value to "Stop"
     }

 }       // end if statement

}        // end Ping()

void pingToggle()                                 // Turns Collision avoidance on/ off
 {
 if (pingOn == true) {
   pingOn = false;
   Serial1.print("Collision Avoidance OFF");
 }
   else if (pingOn == false) {
   pingOn = true;
   Serial1.print("Collision Avoidance ON");
 }

 }
```

9. Bluetooth

```
//*************************************************************************************
************************************************************
// This procedure reads the serial port - Serial1 - for bluetooth commands being sent from the Android
device
```

```
void bluetooth()
{
 while (Serial1.available())                    // Read bluetooth commands over Serial1 // Warning: If an error with Serial1 occurs, make sure Arduino Mega 2560 is Selected
 {
  {
     str = Serial1.readStringUntil('\n');          // str is the temporary variable for storing the last sring sent over bluetooth from Android device
     //Serial.print(str);                          // for testing purposes
  }

   blueToothVal = (str.toInt());                   //  convert the string 'str' into an integer and assign it to blueToothVal
   Serial.print("BlueTooth Value ");
   Serial.println(blueToothVal);

//
*****************************************************************************************************************

 switch (blueToothVal)
 {
    case 1:
      Serial1.println("Forward");
      Forward();
      break;

    case 2:
      Serial1.println("Reverse");
      Reverse();
      break;

    case 3:
      Serial1.println("Left");
      LeftTurn();
      StopCar();
      break;

    case 4:
      Serial1.println("Right");
      RightTurn();
      StopCar();
      break;

    case 5:
      Serial1.println("Stop Car ");
      StopCar();
      break;

    case 6:
      setWaypoint();
```

```
  break;

case 7:
  goWaypoint();
  break;

case 8:
  Serial1.println("Turn Around");
  turnAround();
  break;

case 9:
   Serial1.println("Compass Forward");
  setHeading();
  Compass_Forward();
  break;

case 10:
  setHeading();
  break;

case 11:
   gpsInfo();
  break;

case 12:
  Serial1.println("Compass Turn Right");
  CompassTurnRight();
  break;

case 13:
  Serial1.println("Compass Turn Left");
  CompassTurnLeft();
  break;

case 14:
  Serial1.println("Calibrate Compass");
  calibrateCompass();
  break;

case 15:
  pingToggle();
  break;

case 16:
  clearWaypoints();
  break;

case 17:                    // finish with waypoints
  ac = 0;
  Serial1.print("Waypoints Complete");
```

```
      break;


 } // end of switch case

//
*****************************************************************************************
*************************************************************
// Slider Value for Speed

if (blueToothVal)
  {
   //Serial.println(blueToothVal);
 if (blueToothVal >= 1000)
{
   Serial1.print("Speed set To:  ");
   Serial1.println(blueToothVal - 1000);
   turn_Speed = (blueToothVal - 1000);
   Serial.println();
   Serial.print("Turn Speed ");
   Serial.println(turn_Speed);
 }
  }

/*

//
*****************************************************************************************
*************************************************************
// Detect for Mines - Sweep Not Used

 else if (blueToothVal== 15)
  {
   Serial1.println("Detecting");
   sweep();
  }

//
*****************************************************************************************
*************************************************************
*/
 }                                          // end of while loop Serial1 read

                                           // if no data from Bluetooth
  if (Serial1.available() < 0)                    // if an error occurs, confirm that the arduino mega board
is selected in the Tools Menu
    {
     Serial1.println("No Bluetooth Data ");
    }

} // end of bluetooth procedure
```

9. gps guided robot

```
// The GPS Guided Bluetooth Robot
// Designed by Mark Conner
// Files can be found at Github
// https://github.com/mkconer/GPSRobot
// You can find the Instructable here
// http://www.instructables.com/id/How-to-Build-a-GPS-Guided-Robot/
// The Robot requires an Arduino Mega, L293D motor controller, HMC5883L magnetometer, Ublox 6m GPS, and an HC-05 Bluetooth module
// Remember to run the HMC5883L compass calibration test and enter the results into the code below under void Setup() compass.setOffset(0,0);

// If you are having problem with the compass Check the I2C address, mine is 0x1E
// You may need to connect the compass to the Arduino and run a I2C Scanner Sketch to check the I2C address
// The HMC5883L chip used in the compass module has a fixed I2C address of 0x1E.
// This means that the Jarzebski HMC5883L library assumes that the compass module is connected to the I2C bus with this address,
// and the library's code is written accordingly.
// In the library's source code, you can find the line that sets the I2C address to 0x1E in the HMC5883L.h header file:
// #define HMC5883L_I2CADDR 0x1E
// This sets the I2C address to 0x1E, which is the default address of the HMC5883L chip used in the compass module.
// If your compass module (i.e. qmc5883) has a different I2C address
// you can modify this value in the header file to match the address of your module.
// In that case, you need to upload the modified library to the Arduino board in order to use it.


#include <AFMotor.h>                          // the Adafruit Motor Shield Library ver. 1 https://learn.adafruit.com/adafruit-motor-shield/library-install
#include "Wire.h"                             // Used by I2C and HMC5883L compass
#include "I2Cdev.h"                           // I2C Communications Library (used for compass)
#include "HMC5883L.h"                         // Library for the compass - Download from Github @ https://github.com/jarzebski/Arduino-HMC5883L
#include <Servo.h>                            // Servo library to control Servo arm for metal detector
#include <SoftwareSerial.h>                   // Software Serial for Serial Communications - not used
#include <TinyGPS++.h>                        // Tiny GPS Plus Library - Download from http://arduiniana.org/libraries/tinygpsplus/
                                              // TinyGPS++ object uses Hardware Serial 2 and assumes that you have a
                                              // 9600-baud serial GPS device hooked up on pins 16(tx) and 17(rx).

//*************************************************************************************************
// GPS Variables & Setup
```

```
int GPS_Course;                                 // variable to hold the gps's determined course to
destination
int Number_of_SATS;                             // variable to hold the number of satellites acquired
TinyGPSPlus gps;                                // gps = instance of TinyGPS
                                                // pin 17 (blue)   is connected to the TX on the GPS
                                                // pin 16 (yellow) is connected to the RX on the GPS

//*************************************************************************************************
// Setup Drive Motors using the Adafruit Motor Controller version 1.0 Library

AF_DCMotor motor1(1, MOTOR12_64KHZ);            // create motor #1, 64KHz pwm
AF_DCMotor motor2(2, MOTOR12_64KHZ);            // create motor #2, 64KHz pwm
AF_DCMotor motor3(3, MOTOR12_64KHZ);            // create motor #3, 64KHz pwm
AF_DCMotor motor4(4, MOTOR12_64KHZ);            // create motor #4, 64KHz pwm

int turn_Speed = 175;                           // motor speed when using the compass to turn left and
right
int mtr_Spd = 200;                              // motor speed when moving forward and reverse

//*************************************************************************************************
// Compass Variables & Setup

HMC5883L compass;                               // HMC5883L compass(HMC5883L)
int16_t mx, my, mz;                             // variables to store x,y,z axis from compass
(HMC5883L)
int desired_heading;                            // initialize variable - stores value for the new desired
heading
int compass_heading;                            // initialize variable - stores value calculated from
compass readings
int compass_dev = 5;                            // the amount of deviation that is allowed in the
compass heading - Adjust as Needed
                                                // setting this variable too low will cause the robot to
continuously pivot left and right
                                                // setting this variable too high will cause the robot to veer off
course

int Heading_A;                                  // variable to store compass heading
int Heading_B;                                  // variable to store compass heading in Opposite direction
int pass = 0;                                   // variable to store which pass the robot is on

//*************************************************************************************************
// Servo Control

Servo myservo;                                  // create servo object to control a servo
int pos = 0;                                    // variable to store the servo position

//*************************************************************************************************
```

```
// Ping Sensor for Collision Avoidance

boolean pingOn = false;                          // Turn Collision detection On or Off

int trigPin = 43;                               // Trig - Orange
int echoPin = 42;                                // Echo - Yellow
long duration, inches;
int Ping_distance;

unsigned long currentMillis = 0;
unsigned long previousMillis = 0;                     // Store last time Ping was updated
const long interval = 200;                          // Ping the Distance every X miliseconds

//*********************************************************************************
*****************
// Bluetooth Variables & Setup

String str;                                    // raw string received from android to arduino
int blueToothVal;                                // stores the last value sent over via bluetooth
int bt_Pin = 34;                                // Pin 34 of the Aruino Mega used to test the Bluetooth
connection status - Not Used

//*********************************************************************************
*****************
// GPS Locations

unsigned long Distance_To_Home;                          // variable for storing the distance to destination

int ac =0;                                    // GPS array counter
int wpCount = 0;                                 // GPS waypoint counter
double Home_LATarray[50];                              // variable for storing the destination Latitude -
Only Programmed for 5 waypoint
double Home_LONarray[50];                               // variable for storing the destination Longitude -
up to 50 waypoints


int increment = 0;

void setup()
{
  Serial.begin(115200);                           // Serial 0 is for communication with the computer
  Serial1.begin(9600);                           // Serial 1 is for Bluetooth communication - DO NOT
MODIFY - JY-MCU HC-05 v1.40
  Serial2.begin(9600);                           // Serial 2 is for GPS communication at 9600 baud - DO
NOT MODIFY - Ublox Neo 6m
  myservo.attach(9);                            // attaches the servo to pin 9 (Servo 0 on the Adafruit
Motor Control Board)

  pinMode(36, OUTPUT);                                // define pin 36 as an output for an LED indicator -
Not Used
```

```
  pinMode(bt_Pin, INPUT);                          // This pin(34) is used to check the status of the
Bluetooth connection - Not Used

  // Ping Sensor
  pinMode(trigPin, OUTPUT);                         // Ping Sensor
  pinMode(echoPin, INPUT);                          // Ping Sensor

  // Compass
  Wire.begin();                                  // Join I2C bus used for the HMC5883L compass
  compass.begin();                                // initialize the compass (HMC5883L)
  compass.setRange(HMC5883L_RANGE_1_3GA);                  // Set measurement range
  compass.setMeasurementMode(HMC5883L_CONTINOUS);            // Set measurement mode
  compass.setDataRate(HMC5883L_DATARATE_30HZ);              // Set data rate
  compass.setSamples(HMC5883L_SAMPLES_8);                // Set number of samples averaged
  compass.setOffset(81,-217,105);                       // Set calibration offset

  Startup();                                    // Run the Startup procedure on power-up one time
}

//*********************************************************************************
********************
// Main Loop

void loop()
{

  bluetooth();                                  // Run the Bluetooth procedure to see if there is any data
being sent via BT
  getGPS();                                   // Update the GPS location
  getCompass();                                 // Update the Compass Heading
  Ping();                                   // Use at your own discretion, this is not fully tested

}
```

9. Conclusion: Building this GPS-guided robot was a highly rewarding and educational experience. Throughout the process, I learned how to effectively integrate and program a variety of hardware components, including the Arduino Mega 2560, Ublox NEO-6M GPS module, L293D motor controller, HMC5883L compass, and the HC-05 Bluetooth module. Each of these components played a crucial role in the robot's ability to autonomously navigate towards pre-defined GPS waypoints.

One of the most valuable aspects of this project was learning how GPS signals could be translated into real-time robotic movement. By programming the Arduino to interpret GPS coordinates and optimize the robot's path, I gained a deeper understanding of navigation systems and the complexities involved in ensuring accuracy, especially when dealing with signal delays or errors. The use of the magnetometer allowed the robot to maintain a proper heading, which was essential for following precise routes.

Powering the robot with an 11.1V Li-Po battery gave me insights into managing power consumption for autonomous systems. Working with the L293D motor driver also provided me practical experience in controlling motor speed and direction, allowing for precise movement control through pulse-width modulation (PWM).

Another significant takeaway was the importance of proper calibration and troubleshooting. I encountered challenges with GPS accuracy, compass interference, and wireless communication, which pushed me to experiment with hardware adjustments, shielding, and software refinements. Each problem-solving step enhanced my practical skills in robotics and electronics.

Overall, this project offered me a comprehensive introduction to autonomous systems, combining hardware interfacing, real-time navigation, and microcontroller programming. It has opened the door to more advanced robotic projects and further exploration into fields like drone navigation, autonomous vehicles, and IoT devices.

10. Weekly report:

S no.	Week	Work done
1.	**1st Week**	Learned about the hardware requirements , researched papers
2.	**2nd Week**	Integration of components
3.	**3rd Week**	Created an application in mobile phone using mit app inventor
4.	**4th Week**	Documentation and working of gps guided robot

11. Future challenges:

1. Obstacle Detection and Avoidance
2. Indoor Navigation
3. Weather and Environmental Conditions
4. GPS Accuracy and Reliability

12. Reference:

1. https://www.instructables.com/How-to-Build-a-GPS-Guided-Robot/
2. https://github.com/mkconer/GPSRobot
3. https://www.youtube.com/watch?v=yVpqi-oyFOg&list=PLq6SKTDbk4LCN8TNmIWZGxU0crX_0V9sq
4. https://www.iosrjournals.org/iosr-jece/papers/Vol.%2016%20Issue%203/Series-2/A1603020111.pdf
5. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10677758
6. https://www.acadpubl.eu/hub/2018-119-15/1/169.pdf
7. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7429883
8. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9313890
9. https://www.researchgate.net/publication/340950648_Design_of_an_Autonomous_Controlled_and_GPS-guided_Experimental_Small-Scale_Prototype
10. https://ijisrt.com/assets/upload/files/IJISRT22APR1551_(1).pdf
11. https://www.iosrjournals.org/iosr-jece/papers/Vol.%2016%20Issue%203/Series-2/A1603020111.pdf
12. https://www.researchgate.net/publication/350560496_A_cost-effective_GPS-aided_autonomous_guided_vehicle_for_global_path_planning

13. https://ieeexplore.ieee.org/abstract/document/7429883
14. https://ieeexplore.ieee.org/document/9683883
15. https://www.researchgate.net/publication/350560496_A_cost-effective_GPS-aided_autonomous_guided_vehicle_for_global_path_planning
16. https://www.sciencedirect.com/science/article/pii/S2405896322010047
17. https://www.researchgate.net/publication/349866750_Design_of_Obstacle_Avoidance_and_Waypoint_Navigation_using_Global_position_sensor_Ultrasonic_sensor
18. https://robocraze.com/products/hc-06-bluetooth-module?currency=INR&variant=40192663814297&stkn=c03aaf8845ec&campaignid=20685484537&adgroupid=&keyword=&device=c&gad_source=1&gclid=Cj0KCQjwmt24BhDPARIsAJFYKk3XYKGd-HUUC6DZrxj4o7hWE2QrS85zI522P9QuHEvOqQP0LppX-4AaAmM4EALw_wcB
19. https://www.electronicscomp.com/l293d-motor-driver-shield-arduino-india?srsltid=AfmBOoq60n_TNBt5KSqNFuYJpZDU78E1GvkYit56BYaRasjyTGZXcXrgonE
20. https://robu.in/product/orange-2200mah-3s-30c60c-lithium-polymer-battery-pack-lipo/
21. https://etstore.in/index.php/product/hmc5883l-power-triple-axis-compass-magnetometer-sensor-module-for-arduino-compass-module-hmc5883l/
22. https://zbotic.in/product/ublox-neo-7m-gps-module-with-eeprom-for-c-aeroquad-with-antenna-%EF%BC%88with-battery/?gad_source=4&gclid=Cj0KCQjwmt24BhDPARIsAJFYKk2B3jVLbl0w82EUUbGa_tFi2-AgAtmzlU24ZVuNUoVqBO14gN8MN6UaApmtEALw_wcB
23. https://www.flyrobo.in/10cm_male_to_female_jumper_cable_wire_for_arduino?tracking=ads&srsltid=AfmBOor_kqeEzqsGOSyT-I7dltvLMd0cw6jCnTObr_7A0EJ6sKzSPLjf6kQ
24. https://www.elprocus.com/arduino-mega-2560-board/
25. https://robocraze.com/products/mega-2560-r3-board-compatible-with-arduino?variant=40192428343449&currency=INR&utm_medium=product_sync&utm_source=google&utm_content=sag_organic&utm_campaign=sag_organic&utm_source=google&utm_medium=cpc&utm_campaign=BL+%7C+Pmax+%7C+Feed+Only+%7C+Top+40+Revenue+%7C+09%2F06&utm_source=googleads&utm_medium=ppc&utm_campaign=21373062889&utm_content=_&utm_term=&campaignid=21373062889&adgroupid=&campaign=21373062889&gad_source=1&gclid=CjwKCAjwyfe4BhAWEiwAkIL8sNxYepyIap_Q_njb8Z2o_zdai0-UykRCjNBoot0dT3vGLpNLyIBTAxoCIg8QAvD_BwE
26. https://www.amazon.in/Rubber-Wheel-Smart-Wheels-YELLOW/dp/B097RXFYGP
27. https://www.arduino.cc/en/software