

Overall Idea: 1 Car, 3 Modes

You will have **one smart rover** with:

- **Main controller** → ESP32 Dev Board
- **AI camera brain** → ESP32-S3 AI Camera
- **Sensors** → 4 Ultrasonic + 1 mmWave Human Presence Sensor
- **Motion** → Johnson motors + motor driver + chassis
- **Audio** → Mic + DFPlayer + Speaker
- **Communication with Mobile** → WiFi or Bluetooth from ESP32

And one **“mode variable”** in your code:

- MODE = AUTONOMOUS
- MODE = GUEST
- MODE = UNIFORM_CHECK

You can switch this **via mobile app / web page / Bluetooth commands** or via physical switches.

1) Mode 1 – Autonomous Vehicle (Car)

- 4 Ultrasonic Sensors → Walls, objects, obstacles
- 1 Human Presence Sensor → Humans, insects, etc.
- Car moves automatically

How to place and use sensors

- **Ultrasonic Sensors (x4)**
 - Front-left
 - Front-right
 - Rear-left
 - Rear-rightThey give distances to nearby obstacles → used for **path planning & turning**.
- **mmWave Human Presence Sensor (x1)**

- Mounted in front
- Detects **people's presence/movement** in front of the rover

How the logic works

In AUTONOMOUS mode:

1. Read all 4 **ultrasonic sensors**
 - If distance < threshold → obstacle
 - Decide: turn left/right/stop/go back
2. Read **mmWave sensor**
 - If human presence is detected → treat obstacle as **person**
 - Else → treat it as wall/object
3. Decision:
 - If **only ultrasonic detects obstacle**, but **no human signal**
→ It is likely **wall/object** → **turn / reroute**
 - If **human sensor detects presence**
→ It is **person** → **STOP**, trigger extra logic (uniform check or greeting, depending on active mode).

Note:

mmWave **won't reliably detect insects** (too small).

So practically, you will handle:

- Walls / objects → detected only by ultrasonic
- Humans → detected by mmWave (+ possibly camera)

“**Insect detection**” is more of a theoretical idea; in reality, the rover will just react to “**small obstacle close to sensor**” → you can **treat that as “small object/insect”** and just turn.

2) Mode 2 – Guest Receiving Mode

- Guest Receiving Mode: ON
- Autonomous Mode: OFF
- Mobile Operate – remote control (RC car style)

- Car can travel in all directions based on mobile input
- What you speak in phone → car speaks out loud

This is very practical and impressive for demos.

What happens in GUEST MODE

Here:

- **MODE = GUEST**
- **Autonomous navigation is disabled.**
- Rover movement is **fully controlled via mobile.**

Remote Control Movement

You can do this in two main ways:

Option A – WiFi Web Remote (Simple)

1. ESP32 creates **WiFi Access Point** or connects to router.
2. ESP32 runs a **simple web server** with buttons:
 - Forward / Back / Left / Right / Stop
3. You open the IP in your mobile browser.
4. When you tap buttons, it sends HTTP requests:
 - /move?dir=forward
 - /move?dir=left
5. ESP32 reads those commands and drives the motors.

Option B – Bluetooth Remote

1. Use ESP32 BLE or classic Bluetooth.
2. Make a simple Android app or use a generic Bluetooth controller app.
3. When you press buttons, it sends characters:
 - F → Forward
 - B → Back
 - L → Left
 - R → Right

- S → Stop
4. ESP32 reads serial data over Bluetooth and controls the car.

Voice: “What we talk in phone, car speaks”

This part has two approaches:

Easiest Practical Way (Recommended for Project Demo)

- Put a **Bluetooth speaker** on the car.
- Connect that Bluetooth speaker to your **mobile phone**.
- Now, **whatever you speak in phone**, or any audio (call, mic, app), plays on the car speaker.

This is:

- Very easy
- No complex streaming code
- Works with any normal call or mic app

You can stand near the gate, speak in your phone, and your voice is heard from the car.

More Advanced Way (If you want IoT style)

- Build your own **audio streaming**:
 - Mobile app captures microphone audio.
 - Sends audio packets over WiFi (e.g., using WebSocket/UDP) to ESP32.
 - ESP32 decodes PCM data and outputs via I2S amplifier to speaker.

This is more complex (buffering, latency, encoding), but **possible** with ESP32-Audio libraries.

For a B.Tech project, I strongly suggest:

Bluetooth Speaker on Car + Mobile Phone as Mic

→ Clean, simple, and still “smart” for demo.

3) Mode 3 – Student Uniform Analysis

- Guest Mode: OFF
- Student uniform mode: ON
- Autonomous Mode: ON

In this mode, rover is **autonomous**, but when it sees a **person**, it performs **uniform checking**.

Basic Flow

1. MODE = UNIFORM_CHECK
2. Car moves in **autonomous mode** like earlier:
 - Uses ultrasonic + mmWave for basic navigation.
3. When **mmWave detects a human**:
 - Car stops.
 - ESP32 sends a signal to ESP32-S3 AI camera:
→ “Capture frame and analyze student”
4. **ESP32-S3 AI Camera**:
 - Runs your ML model (trained with images of:
 - Full uniform
 - Partial uniform
 - No uniform)
 - Outputs a label: FULL, PARTIAL, or NONE.
5. Based on result:
 - If FULL
→ ESP32 triggers DFPlayer to play:
 “Super — fully uniformed!”
 - If PARTIAL or NONE
→ Play:
 “You are not in full college uniform. Get out from the college.”
6. After finishing:
 - The rover continues navigation to find next person.

How Components Talk to Each Other

To make all modes work together:

- **Main ESP32:**
 - Handles:
 - Motors
 - Ultrasonic sensors
 - mmWave sensor
 - Mode selection
 - Bluetooth/WiFi for mobile control
 - DFPlayer commands
- **ESP32-S3 AI Camera:**
 - Does ONLY:
 - Video capture
 - Running AI model
 - Sending uniform result to ESP32

They can communicate via:

- **UART / Serial** (simple & recommended)
 - ESP32 → “DETECT_UNIFORM”
 - ESP32-S3 → “UNIFORM:FULL” or “UNIFORM:PARTIAL”

What Is NOT Exactly Possible / Needs Adjusting

1. Insects classification as a separate category

- Ultrasonic and mmWave cannot reliably detect “insect” type.
- Realistically: you can treat **very small nearby obstacles** as “small object/insect” → just avoid it.
- But you cannot say “**this is an insect**” confidently with these sensors.

2. Perfect real-time live audio streaming over WiFi

- Technically possible but advanced.
- For college project, **use Bluetooth speaker solution** → much simpler, still very impressive.

Everything else you described is **practical and implementable** with your current components.