

*A Project Report*

*on*

# **SMART ARDUINO BLIND STICK**

*carried out as part of the course CSE CS3270 Submitted by*

***Name of student : Ayush Goyal***

***Roll no : 219301331.***

***VI-CSE***

*in partial fulfilment for the award of the degree*

*of*

**BACHELOR OF TECHNOLOGY**

**In**

**Computer Science and Engineering**



**MANIPAL UNIVERSITY  
JAIPUR**

*(University under Section 2(f) of the UGC Act)*

**Department of Computer Science and Engineering,  
School of Computer Science and Engineering,  
Manipal University Jaipur,  
*JAN-MAY 2024***

## Acknowledgement

This project would not have completed without the help, support, comments, advice, cooperation, and coordination of various people. However, it is impossible to thank everyone individually; I am hereby making a humble effort to thank some of them.

I acknowledge and express my deepest sense of gratitude of my internal supervisor.

Dr. Juhi Singh for her constant support, guidance, and continuous engagement. I highly appreciate her technical comments, suggestions, and criticism during the progress of this project “ **SMART ARDUINO BLIND STICK** ”.

I owe my profound gratitude to **Prof. Neha Chaudhary**, Head, Department of CSE, for her valuable guidance and facilitating me during my work. I am also very grateful to all the faculty members and staff for their precious support and cooperation during the development of this project.

Finally, I extend my heartfelt appreciation to my classmates for their help and encouragement.

### Student Signature

**Student Name : Ayush Goyal**  
**Registration No. : 219301331**



MANIPAL UNIVERSITY  
JAIPUR

(University under Section 2(f) of the UGC Act)

**Department of Computer Science and Engineering**  
**School of Computer Science and Engineering**

Date: 15<sup>Th</sup> May 2024

**CERTIFICATE**

This is to certify that the project entitled “**Smart Arduino Blind Stick**” is a bonafide work carried out as ***Minor Project (Course Code: CS3270)*** in partial fulfilment for the award of the degree of Bachelor of Technology in Computer Science and Engineering, under my guidance by **Ayush Goyal** bearing registration number **219301331** , during the academic semester *VI of year 2023-24*.

**Signature of the project guide:**

**Name of the project guide: Dr Juhi Singh**

**Place: Manipal University Jaipur, Jaipur**

# Contents

Table of Contents		Page No.
1	Introduction	1
1.1	Objective of the project	
1.2	Brief description of the project	
1.3	Technology used.	
1.3.1	H / W Requirement	
1.3.2	Software Requirement	
2	Design Description	2
2.1	Prototype	
2.2	Flow chart	
3	Input/Output Form Design	4-5
4	Testing cases	6
5	Implementation & Maintenance	7
5.1	Deployment Guide	
6	Future scope	8
7	Conclusion	9
8	Bibliography	9
9	Course Outcome	10

# 1 Introduction

## 1.1. Objective:

The objective of the Smart Arduino Blind Stick (SABS) project is to develop an assistive device for visually impaired individuals that enhances their mobility and safety. The device integrates multiple functionalities, including obstacle detection, water hazard sensing, GPS navigation, and emergency alert capabilities, into a single, cost-effective solution.

## 1.2 Description of the Project:

The Smart Arduino Blind Stick (SABS) is an innovative tool designed to aid visually impaired individuals in navigating their environment safely and independently. Developed with an investment of 3500 Indian Rupees, the SABS incorporates various sensors and modules to provide real-time feedback on obstacles, objects, and environmental hazards. The device is equipped with ultrasonic sensors for obstacle detection, a moisture sensor for water hazard detection, a GPS module for navigation, an emergency button for immediate assistance, and an ESP32 camera module for object detection using the YOLOv3 OpenCV model. When an object is detected by the camera, the device provides an alert through the buzzer.

## 1.3 Technology used:

### 1.3.1 H/W requirements:

- Arduino Uno
- Ultrasonic Sensors (for obstacle detection)
- Moisture Sensor (for water hazard detection)
- Power Supply
- Emergency Button
- GPS Module (for navigation)
- GSM Module (for sending emergency alerts)
- ESP32 Camera
- Module (for object detection)
- Speaker (for voice output)
- Buzzer (for audible alerts)

### 1.3.2 S/W requirements:

- OpenCV (for object detection using YOLOv3)
- Arduino-IDE
- Software Serial Library
- TinyGPS Library

## 2 Design Description:

### 2.1 Prototype :

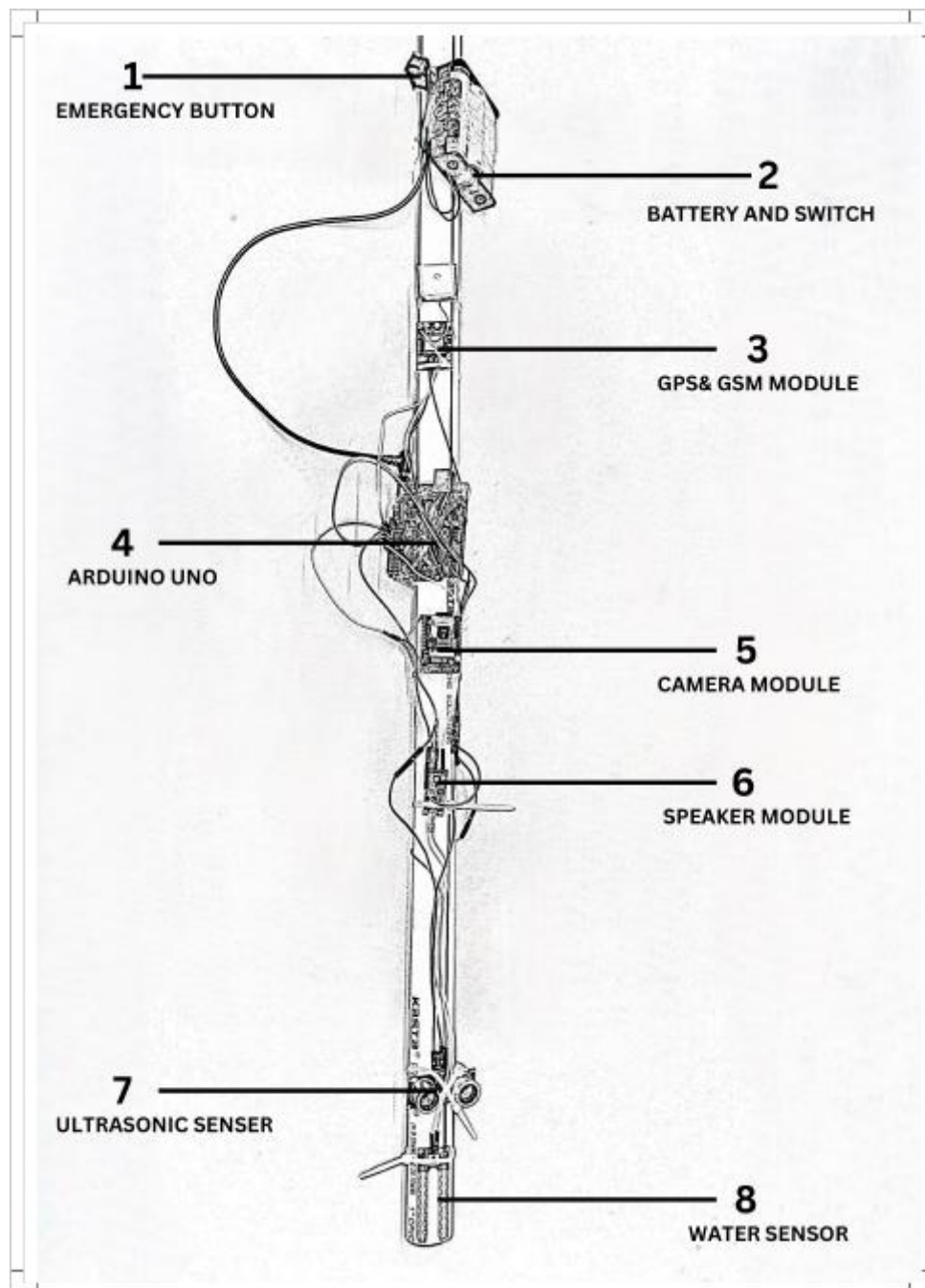


Fig 2.1 Prototype design

## 2.2 Flow Chart:

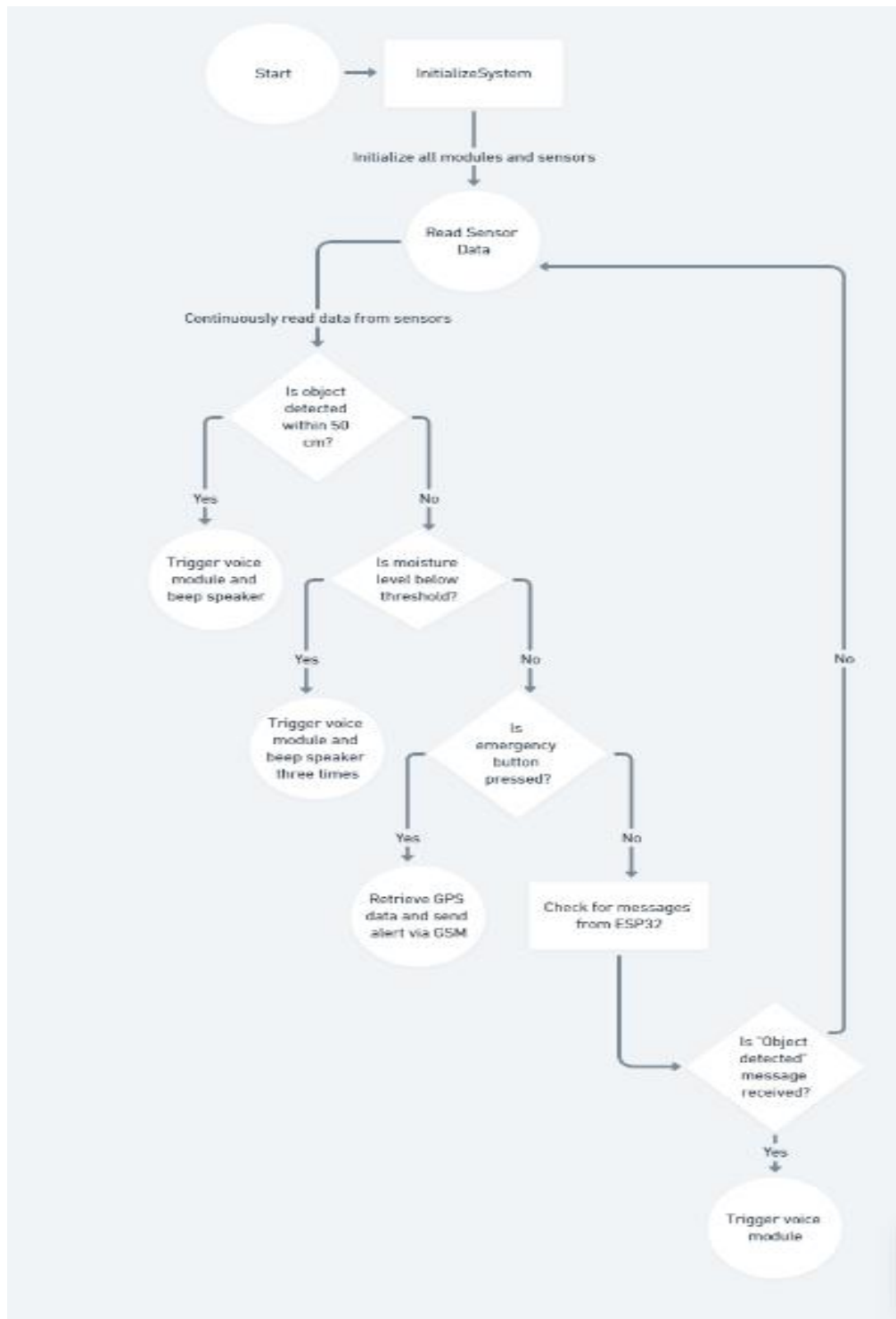


Fig 2.2 Flow chart

## 3 Input/Output Form Design:

The input and output form design for the Smart Arduino Blind Stick (SABS) project is crucial for ensuring that the device interacts effectively with the user and the environment. This section outlines the design considerations and implementations for both input and output components of the system.

### 3.1 Input Form Design

The input components of the SABS are responsible for gathering data from the environment and the user, which are then processed to provide meaningful outputs. The primary inputs include:

**1. Ultrasonic Sensors (Obstacle Detection)**

- Trigger Pin (UTrigPin) and Echo Pin (UEchoPin)
- Design Consideration: Positioned at the front of the stick to detect obstacles within a certain range. The trigger pin sends out ultrasonic pulses, and the echo pin receives the reflected signal to calculate the distance to the obstacle.

**2. Moisture Sensor (Water Hazard Detection)**

Moisture Pin (MoisPin)

Design Consideration: Placed near the tip of the stick to detect water or other liquid hazards on the ground. The sensor measures the soil moisture level and sends an analog signal to the Arduino.

**3. GPS Module (Navigation)**

GPSRX and GPSTX Pins

Design Consideration: Mounted securely to the stick, ensuring clear reception of satellite signals. It provides real-time location data for navigation and emergency purposes.

**4. ESP32 Camera Module (Object Identification)**

Data Pins and Power Supply

Design Consideration: Positioned to capture the user's forward-facing view. The camera continuously processes images using the YOLOv3 model to identify objects and alerts the user via the buzzer.

**5. Emergency Button**

Emergency Button Pin (EMERGENCY\_BUTTON)

Design Consideration: Located within easy reach on the handle of the stick. When pressed, it triggers the GPS module to send the user's location to a predefined contact number via the GSM module.

**6. GSM Module (Communication)**

GSM\_TX and GSM\_RX Pins

Design Consideration: Integrated into the stick to send SMS alerts in emergency situations.

### 3.2 Output Form Design

The output components provide feedback and information to the user based on the processed input data. The primary outputs include:

**1. Speaker (Voice Feedback)**

Speaker Pin (SPKPin)

Design Consideration: Provides auditory feedback to the user. When an obstacle or water hazard is detected, or when the emergency button is pressed, the speaker plays pre-recorded messages or tones.

**2. Buzzer (Alert Signals)**

Buzzer Pin (BuzzPin)

Design Consideration: Emits beeps to alert the user of obstacles or identified objects. The intensity and frequency of the beeps can vary based on the proximity of the detected object.



### 3. Serial Monitor (Debugging and Status Updates)

Serial Communication Pins

Design Consideration: Used during development and testing to display real-time data from the sensors and modules. This output is crucial for debugging and ensuring the system functions correctly.

### 3.3 Input/Output Interaction

The interaction between input and output components is managed through the Arduino code, which processes the inputs and triggers the appropriate outputs. Here is an overview of the interaction:

- **Obstacle Detection:**  
Input: Ultrasonic sensor detects an obstacle.  
Output: Speaker plays a warning message, and the buzzer emits a beep.
- **Object Identification:**  
Input: ESP32 camera detects an object using YOLOv3 model.  
Output: Buzzer emits a beep to alert the user.
- **Water Hazard Detection:**  
Input: Moisture sensor detects water.  
Output: Speaker plays a warning message.
- **GPS Navigation:**  
Input: GPS module provides location data.  
Output: Location data is displayed on the serial monitor and used for emergency alerts.
- **Emergency Alert:**  
Input: Emergency button is pressed.  
Output: GSM module sends an SMS with the user's location to a predefined contact.

### 3.4 User Interface Design

Although the primary user interface for the SABS is auditory and tactile feedback, the design ensures that all interactions are intuitive and user-friendly:

- Auditory Feedback: Clear, distinct voice messages and beep tones guide the user.
- Tactile Feedback: Physical buttons (like the emergency button) are easily accessible.
- Visual Feedback: During development, the serial monitor provides visual feedback for testing and debugging purposes.

By carefully designing the input and output forms, the SABS ensures that visually impaired users receive timely and accurate information, significantly enhancing their mobility and safety.

## 4. Testing Cases:

### 1. Obstacle Detection

- i. Objective: Verify the ultrasonic sensors accurately detect obstacles within a specified range.
- ii. Procedure:
  - Place obstacles of varying sizes and shapes at different distances from the SABS.
  - Activate the SABS and approach each obstacle.
  - Note if the speaker plays a warning message and the buzzer emits a beep upon detecting the obstacles.
- iii. Expected Result: The SABS should detect obstacles within the specified range and provide timely alerts.

## **2. Water Hazard Detection**

- i. Objective: Ensure the moisture sensor correctly identifies the presence of water.
- ii. Procedure:
  - Place the moisture sensor in a dry environment.
  - Gradually introduce water or moisture to the sensor.
  - Observe if the speaker plays a warning message upon detecting moisture.
- iii. Expected Result: The SABS should detect water or moisture and provide a warning message.

## **3. GPS Navigation**

- i. Objective: Test the GPS module for accurate location coordinates.
- ii. Procedure:
  - Activate the SABS in an open outdoor area with clear sky visibility.
  - Monitor the serial monitor output to verify the GPS module provides accurate location coordinates.
- iii. Expected Result: The GPS module should provide reliable and accurate location coordinates.

## **4. Emergency Alert**

- i. Objective: Confirm the GSM module sends an SMS alert when the emergency button is pressed.
- ii. Procedure:
  - Press the emergency button on the SABS.
  - Check the designated recipient's phone for the SMS alert containing the user's location coordinates.
- iii. Expected Result: The GSM module should send an SMS alert with the user's location coordinates upon pressing the emergency button.

## **5. Voice Output**

- i. Objective: Check that the speaker provides clear and accurate voice alerts.
- ii. Procedure:
  - Trigger different scenarios that require voice alerts (e.g., obstacle detection, water hazard detection, emergency alert).
  - Listen to the voice messages played by the speaker and ensure they are clear and understandable.
- iii. Expected Result: The speaker should deliver clear and accurate voice alerts for various situations.

## **6. Object Detection**

- i. Objective: Ensure the ESP32 camera correctly identifies objects and triggers the buzzer.
- ii. Procedure:
  - Place objects of different shapes and sizes within the camera's field of view.
  - Activate the SABS and observe the buzzer's response when objects are detected.
- iii. Expected Result: The ESP32 camera should accurately identify objects using the YOLOv3 model and trigger the buzzer to emit beeps upon detection.

## 5 Implementation & Maintenance:

### 5.1 Deployment Guide

#### Step 1: Hardware Assembly

##### 1. Components Required:

- Arduino Uno
- Ultrasonic Sensor (HC-SR04)
- Moisture Sensor
- GPS Module
- GSM Module
- ESP32 Camera Module
- Speaker
- Buzzer
- Emergency Button
- Connecting wires and a breadboard

##### 2. Circuit Connections:

- Connect the ultrasonic sensor's Trig pin to Arduino pin 5 and Echo pin to pin 6.
- Connect the moisture sensor to analog pin A0 of the Arduino.
- Connect the GPS module to pins 3 (RX) and 4 (TX) of the Arduino using Software Serial.
- Connect the GSM module to pins 10 (TX) and 12 (RX) of the Arduino.
- Connect the speaker to pin 11 of the Arduino.
- Connect the buzzer to pin 13 of the Arduino.
- Connect the emergency button to pin 7 of the Arduino.
- Connect the ESP32 camera module as per its specifications and ensure it is properly powered.

##### 3. Power Supply:

- Ensure the Arduino and all sensors/modules are powered appropriately. Use an external power source if needed to provide sufficient power to all components.

#### Step 2: Software Setup

##### 1. Arduino IDE Setup:

- Install the Arduino IDE from the official website.
- Install necessary libraries: `TinyGPS`, `SoftwareSerial`, and any additional libraries required for the ESP32 camera module and GSM module.

##### 2. Uploading Code:

- Open the Arduino IDE.
- Copy and paste the provided code into a new sketch.
- Select the correct board (Arduino Uno) and port from the Tools menu.
- Upload the code to the Arduino.

#### Step 3: Testing and Calibration

##### 1. Initial Testing:

- Power on the device.
- Check the serial monitor for outputs related to GPS coordinates, obstacle detection, water hazard detection, and emergency alerts.
- Press the emergency button to test the SMS alert functionality.
- Place objects within the ESP32 camera's view to test object detection and listen for the buzzer.

##### 2. Calibration:

- Adjust the ultrasonic sensor's threshold distance if needed for accurate obstacle detection.
- Calibrate the moisture sensor by testing it in different moisture conditions and adjusting the threshold value in the code.

- Ensure the GPS module acquires satellite signals and provides accurate location data.
- Verify the GSM module's ability to send SMS alerts and troubleshoot any connectivity issues.

#### Step 4: Final Deployment

1. Enclosure:
  - Place all components in a secure, portable enclosure to protect the hardware.
  - Ensure the sensors are positioned correctly for optimal functionality.
2. Field Testing:
  - Conduct real-world testing by navigating various environments with the SABS.
  - Gather user feedback and make necessary adjustments to improve performance and usability.

#### Step 5: Maintenance

1. Regular Checks:
  - Periodically check connections and sensor functionality.
  - Replace any faulty components as needed.
  - Ensure the firmware is updated to incorporate any improvements or bug fixes.
2. Troubleshooting:
  - If the device fails to operate correctly, verify each sensor and module individually.
  - Use the serial monitor for debugging and identifying issues.
  - Refer to the respective component datasheets for troubleshooting tips and guidance.
3. Updates and Enhancements:
  - Regularly update the software to improve performance and add new features.
  - Explore the potential of integrating additional sensors or modules to enhance the device's capabilities.

## 6 Future Scope:

The Smart Arduino Blind Stick (SABS) has significant potential for future enhancements. Integrating advanced AI models for improved object recognition and cloud-based real-time processing can further refine its capabilities. Adding indoor navigation features using Bluetooth beacons or Wi-Fi triangulation would enhance navigation in complex environments.

Future versions could include health monitoring sensors for vital signs and fall detection, increasing user safety. Voice recognition and a companion mobile app could enhance the user interface, allowing remote configuration and monitoring. Upgrading to 5G for faster emergency alerts and integrating with IoT ecosystems would improve connectivity.

Energy efficiency could be boosted with solar charging and optimized power consumption. Multi-language support and customizable alerts would make the device more accessible. Additional sensors for air quality and temperature could provide more comprehensive safety features. Open sourcing the project and collaborating with research institutions could drive further innovations, ultimately enhancing the independence and quality of life for visually impaired individuals.

## 7 Conclusion:

The Smart Arduino Blind Stick (SABS) successfully integrates multiple assistive technologies into a cost-effective and user-friendly device, significantly enhancing the mobility and safety of visually impaired individuals. By combining obstacle detection, object identification, water hazard sensing, GPS navigation, and emergency alert capabilities, SABS offers a comprehensive solution that promotes independence and confidence in navigation. Future enhancements and continued innovation will further improve its functionality, making it an indispensable tool for visually impaired users.

## 8 Bibliography:

### Journal References

- [1] Kumar, R., Sharma, P., and Gupta, S. (2017). "Design and Development of Smart Blind Stick using Ultrasonic Sensor." *International Journal of Advanced Research in Computer Science*, 8(5), 250-256.
- [2] Kadam, D., Jain, A., and Patel, R. (2018). "Smart Cane: Assistive Navigation Device for Visually Impaired People." *International Journal of Engineering and Technology*, 7(4), 152-158.

### Books

- [3] Zadeh, L.A. (1981). "Possibility theory and soft data analysis." *Mathematical Frontiers of the Social and Policy Sciences*, L. Cobb and R.M. Tharall, eds., Westview, Boulder, Colo., 69-129.
- [4] Cotton, F.A. (2003). *Chemical Applications of Group Theory*, McMillan, London.

### Conference Proceedings and Symposiums

- [5] Garrett, D.L. (2003). "Coupled analysis of floating production systems." *Proc., Int. Symp. on Deep Mooring Systems*, ASCE, Reston, Va., 152-167.

### Web Pages

- [6] OpenCV. (2023). "OpenCV Library Documentation." [<https://opencv.org/documentation>], accessed on Feb 3, 2024.
- [7] Arduino. (2023). "Arduino Uno Documentation." [<https://www.arduino.cc/en/Main/ArduinoBoardUno>], accessed on Feb 3, 2024.
- [8] Espressif Systems. (2023). "ESP32 Camera Module Documentation." [<https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/modules-and-boards.html>] accessed on Feb 25, 2024.
- [9] TinyGPS Library Documentation. (2023). "TinyGPS++." [<http://arduiniiana.org/libraries/tinygpsplus/>] accessed on March 1, 2024.


## 9 Course Outcome:

The development of the Smart Arduino Blind Stick (SABS) as part of this course has led to significant technical and practical achievements. We have gained hands-on experience in integrating various hardware components such as ultrasonic sensors, GPS modules, moisture sensors, and ESP32 camera modules with Arduino. The project has enhanced their skills in programming, circuit design, and real-time problem-solving. The comprehensive nature of the SABS, combining multiple functionalities into a single assistive device, exemplifies innovation and practical application of theoretical knowledge. Moreover, this project has real-world impact, aiming to improve the lives of visually impaired individuals by enhancing their mobility and safety.

We have already filed a patent for the Smart Arduino Blind Stick, underscoring the originality and potential commercial viability of the project. This experience has provided valuable insights into the process of protecting intellectual property and the importance of innovation in engineering. The tentative publication date for the patent is expected to be within the next six months, marking a significant milestone in the course and the students' academic and professional journey.

## Code appendix:

### 1. Arduino IDE Code:



The screenshot shows the Arduino IDE interface. At the top, there's a title bar with the Arduino logo and the text 'sketch\_smartarduinoblindstick | Arduino 1.8.19'. Below the title bar is a menu bar with 'File', 'Edit', 'Sketch', 'Tools', and 'Help'. Under the 'Sketch' menu, there are icons for a checkmark, a right arrow, a grid, an up arrow, and a down arrow. The main workspace displays the sketch name 'sketch\_smartarduinoblindstick' and the following code:

```
#include <SoftwareSerial.h>
#include <TinyGPS.h>

#define SPKPin 11
#define BuzzPin 13
#define MoisPin A0
#define GPSTX 3
#define GPSTX 4
#define UTrigPin 5
#define UEchoPin 6
#define EMERGENCY_BUTTON 7
#define GSM_TX 10
#define GSM_RX 12

TinyGPS gps;
SoftwareSerial serialGPS(GPSTX, GPSTX);
SoftwareSerial serialESP(8, 9);
SoftwareSerial gsmModule(GSM_TX, GSM_RX);

int threshold = 500; // Adjust threshold as needed for detecting voice playback

void setup() {
  Serial.begin(115200);
  serialGPS.begin(9600);
  serialESP.begin(115200);
  gsmModule.begin(9600); // Change baud rate if needed
  pinMode(SPKPin, OUTPUT);
  pinMode(UTrigPin, OUTPUT);
  pinMode(UEchoPin, INPUT);
  pinMode(EMERGENCY_BUTTON, INPUT_PULLUP);
}

void loop() {
  bool newData = false;
  unsigned long chars;
  unsigned short sentences, failed;

  for (unsigned long start = millis(); millis() - start < 1000;) {
    while (serialGPS.available()) {
      char c = serialGPS.read();
      if (gps.encode(c))
        newData = true;
    }
  }
}
```

```

    }
}

if (newData) {
    float flat, flon;
    unsigned long age;
    gps.f_get_position(&flat, &flon, &age);
    Serial.print("LAT=");
    Serial.print(flat == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flat, 6);
    Serial.print(" LON=");
    Serial.print(flton == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flon, 6);
    Serial.print(" SAT=");
    Serial.print(gps.satellites() == TinyGPS::GPS_INVALID_SATELLITES ? 0 : gps.satellites());
    Serial.println("");
}
}

void beep() {
    digitalWrite(BuzzPin, HIGH);
    delay(100);
    digitalWrite(BuzzPin, LOW);
    delay(100);
}

void playVoice() {
    digitalWrite(SPKPin, HIGH); // Trigger playback
    delay(100); // Adjust delay as needed
    digitalWrite(SPKPin, LOW);
}

void dist() {
    int duration, distance;
    digitalWrite(UTrigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(UTrigPin, LOW);
    duration = pulseIn(UEchoPin, HIGH);
    distance = (duration / 2) / 29.1;
    if (distance < 50) {
        Serial.println("Obstacle detected");
        playVoice();
    }
}

void mois() {
    int moisture;
    moisture = analogRead(MoisPin);
    if (moisture < 800) {
        Serial.println("Water detected");
        playVoice();
    }
}

void checkEmergencyButton() {
    if (digitalRead(EMERGENCY_BUTTON) == LOW) {
        loca(); // Get current location
        Serial.println("Emergency Button Pressed!");
        float flat, flon;
        unsigned long age;
        gps.f_get_position(&flat, &flon, &age); // Retrieve latitude and longitude
        gsmModule.println("AT+CMGF=1"); // Set SMS mode to text
        delay(100);
        gsmModule.println("AT+CMGS=\"+918306708847\""); // Replace with recipient's phone number
        delay(100);
        gsmModule.print("Emergency! Current Location: LAT=");
        gsmModule.print(flat, 6);
        gsmModule.print(" LON=");
        gsmModule.print(flton, 6);
        //gsmModule.print(". ESP32 IP Address: ");
        //gsmModule.print(esp32IPAddress);
        gsmModule.write(26); // End of message character
        Serial.println("Message Sent!");
        //delay(5000); // Wait for SMS to be sent
    }
}

void loop() {
    loca();
    dist();
    mois();
    checkEmergencyButton();
    while (serialESP.available() > 0) {
        char inByte = serialESP.read();
        Serial.write(inByte);
    }
}

```





## sketch\_esp32

```
#include <WebServer.h>
#include <WiFi.h>
#include <esp32cam.h>

#define BuzzPin 2

const char* WIFI_SSID = "Ayush";
const char* WIFI_PASS = "ayush235";

WebServer server(80);

static auto loRes = esp32cam::Resolution::find(320, 240);
static auto midRes = esp32cam::Resolution::find(350, 530);
static auto hiRes = esp32cam::Resolution::find(800, 600);
void serveJpg()
{
    auto frame = esp32cam::capture();
    if (frame == nullptr) {
        //Serial.println("CAPTURE FAIL");
        server.send(503, "", "");
        return;
    }
    /*Serial.printf("CAPTURE OK %dx%d %db\n", frame->getWidth(), frame->getHeight(),
        static_cast<int>(frame->size()));*/

    server.setContentLength(frame->size());
    server.send(200, "image/jpeg");
    WiFiClient client = server.client();
    frame->writeTo(client);
}

void handleJpgLo()
{
    if (!esp32cam::Camera.changeResolution(loRes)) {
        //Serial.println("SET-LO-RES FAIL");
    }
    serveJpg();
}

void handleJpgHi()
{

```



sketch\_esp32

```
{
  if (!esp32cam::Camera.changeResolution(loRes)) {
    //Serial.println("SET-LO-RES FAIL");
  }
  serveJpg();
}

void handleJpgHi()
{
  if (!esp32cam::Camera.changeResolution(hiRes)) {
    //Serial.println("SET-HI-RES FAIL");
  }
  serveJpg();
}

void handleJpgMid()
{
  if (!esp32cam::Camera.changeResolution(midRes)) {
    //Serial.println("SET-MID-RES FAIL");
  }
  serveJpg();
}

void buzzon(){
  digitalWrite(BuzzPin, HIGH);
  server.setContentLength(0);
  server.send(200, "");
}

void buzzoff(){
  digitalWrite(BuzzPin, LOW);
  server.setContentLength(0);
  server.send(200, "");
}

void setup(){
  Serial.begin(115200);
  Serial.println();
  {
    using namespace esp32cam;
    Config cfg;
    cfg.setPins(pins::AiThinker);
```



sketch\_esp32

```

}

void setup() {
  Serial.begin(115200);
  Serial.println();
  {
    using namespace esp32cam;
    Config cfg;
    cfg.setPins(pins::AiThinker);
    cfg.setResolution(hiRes);
    cfg.setBufferCount(2);
    cfg.setJpeg(80);

    bool ok = Camera.begin(cfg);
    Serial.println(ok ? "CAMERA OK" : "CAMERA FAIL");
  }
  pinMode(BuzzPin, OUTPUT);
  WiFi.persistent(false);
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }
  Serial.print("http://");
  Serial.println(WiFi.localIP());
  Serial.println(" /cam-lo.jpg");
  Serial.println(" /cam-hi.jpg");
  Serial.println(" /cam-mid.jpg");

  server.on("/cam-lo.jpg", handleJpgLo);
  server.on("/cam-hi.jpg", handleJpgHi);
  server.on("/cam-mid.jpg", handleJpgMid);
  server.on("/buzz-on", buzzon);
  server.on("/buzz-off", buzzoff);

  server.begin();
}

void loop()
{
  server.handleClient();
}

```

## 2. SMS Alert Confirmation:

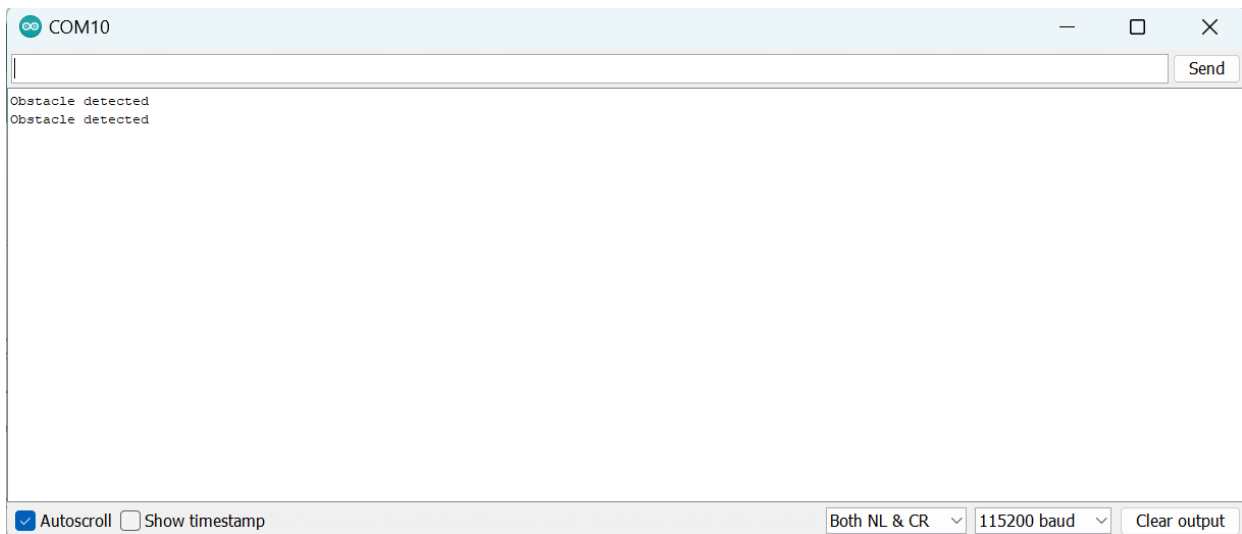


The screenshot shows a terminal window titled "COM10" with a "Send" button in the top right corner. The terminal displays the following text:

```
Emergency Button Pressed!  
Message Sent!  
Emergency Button Pressed!  
Message Sent!  
Emergency Button Pressed!  
Message Sent!  
Emergency Button Pressed!  
Message Sent!  
AT+CMGF=1  
OK  
AT+CMGS="+918306708847"  
> Emergency! Current Location: LAT=1000.000000 LON=1000.000000  
+CMGS: 188  
  
OK
```

At the bottom of the window, there are controls: a checked "Autoscroll" checkbox, an unchecked "Show timestamp" checkbox, a "Both NL & CR" dropdown menu, a "115200 baud" dropdown menu, and a "Clear output" button.

## 3. Obstacle detected Confirmation:



The screenshot shows a terminal window titled "COM10" with a "Send" button in the top right corner. The terminal displays the following text:

```
Obstacle detected  
Obstacle detected
```

At the bottom of the window, there are controls: a checked "Autoscroll" checkbox, an unchecked "Show timestamp" checkbox, a "Both NL & CR" dropdown menu, a "115200 baud" dropdown menu, and a "Clear output" button.

#### 4. Esp32 Module connectin Confirmation:

### Network & internet > Mobile hotspot

Power saving  
When no devices are connected, automatically turn off mobile hotspot

On ☐

Properties

Network properties

Edit

Name: Ayush

Password: ayush235

Band: 2.4 GHz

Devices connected: 1 of 8

Device name	IP address	Physical address (MAC)
esp32-391B04	192.168.137.106	08:d1:f9:39:1b:04

#### 4.Yolo v3 model (opencv)

Home

Untitled

localhost:8888/notebooks/Downloads/Project/Untitled.ipynb

Coverage Database... (1) Ayush Goyal | Lin... Course: Getting Star... Manipal University J... project My IMS 3.0 Forum | POD All Bookmarks

Jupyter Untitled Last Checkpoint: 18 hours ago

File Edit View Run Kernel Settings Help Trusted

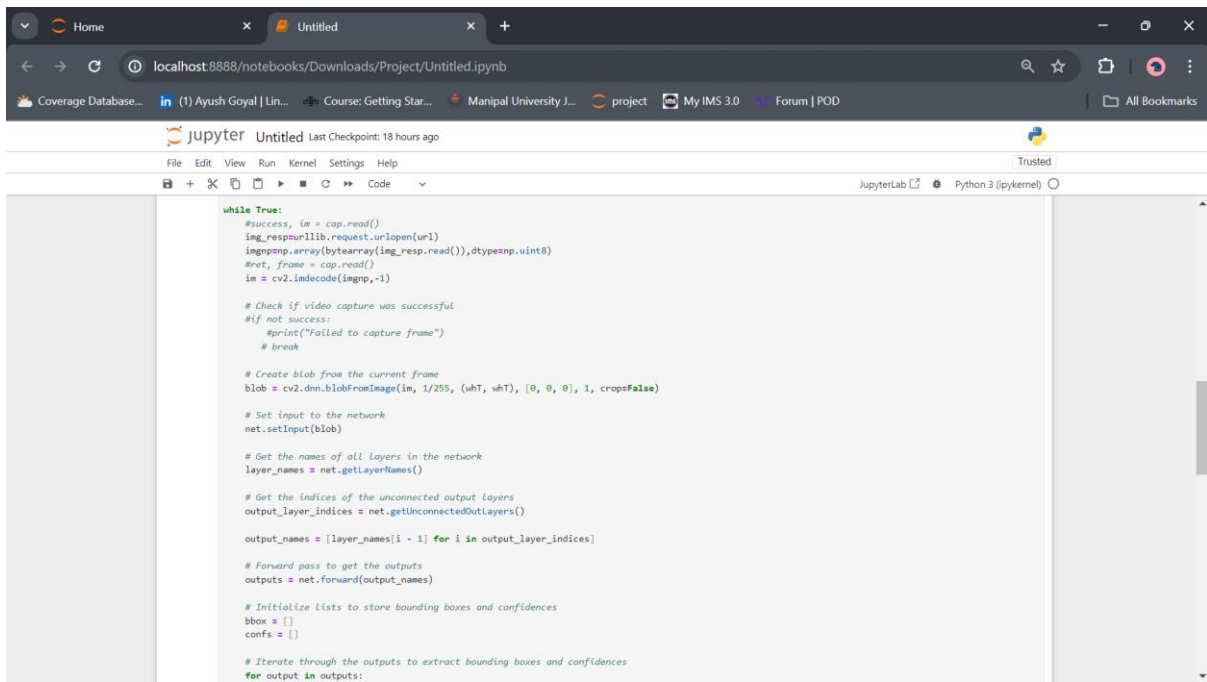
JupyterLab Python 3 (ipykernel)

```
[1]: import cv2
import urllib.request
import numpy as np
import requests
IP="192.168.137.91"
# Change URL to 0 to access the default camera device
url = f"http://{IP}/cam-hi.jpg"

cap = cv2.VideoCapture(url)
wInt=320
confThreshold = 0.5
nmsThreshold = 0.3
classesFile="coco.names.txt"
classNames=[]
with open(classesFile,'rt') as f:
    classNames=f.read().rstrip('\n').split('\n')

modelConfig = 'yolov3.cfg'
modelWeights= 'yolov3.weights'
net = cv2.dnn.readNetFromDarknet(modelConfig,modelWeights)
net.setPreferableBackend(cv2.dnn.DNN_BACKEND_OPENCV)
net.setPreferableTarget(cv2.dnn.DNN_TARGET_CPU)

def findObject(outputs,im):
    hT,wT,cT = im.shape
    bbox = []
    classIds = []
    confs = []
    found_cat = False
    found_bird = False
    for output in outputs:
        for det in output:
```



The screenshot shows a JupyterLab notebook titled 'Untitled' running on a local host. The code is a Python script that processes video frames to detect objects. It starts by reading a video frame from a URL, converting it to a NumPy array, and then using a pre-trained model to detect objects. The code includes comments for each step, such as 'Check if video capture was successful' and 'Create blob from the current frame'. The output of the code is a list of detected objects, which is shown in the next screenshot.

```
while True:
    #success, im = cap.read()
    img_response = request.urlopen(url)
    img_np = np.array(bytearray(img_response.read()), dtype=np.uint8)
    #ret, frame = cap.read()
    im = cv2.imdecode(img_np, -1)

    # Check if video capture was successful
    #if not success:
    #    #print("Failed to capture frame")
    #    break

    # Create blob from the current frame
    blob = cv2.dnn.blobFromImage(im, 1/255, (w, h), [0, 0, 0], 1, crop=False)

    # Set input to the network
    net.setInput(blob)

    # Get the names of all layers in the network
    layer_names = net.getLayerNames()

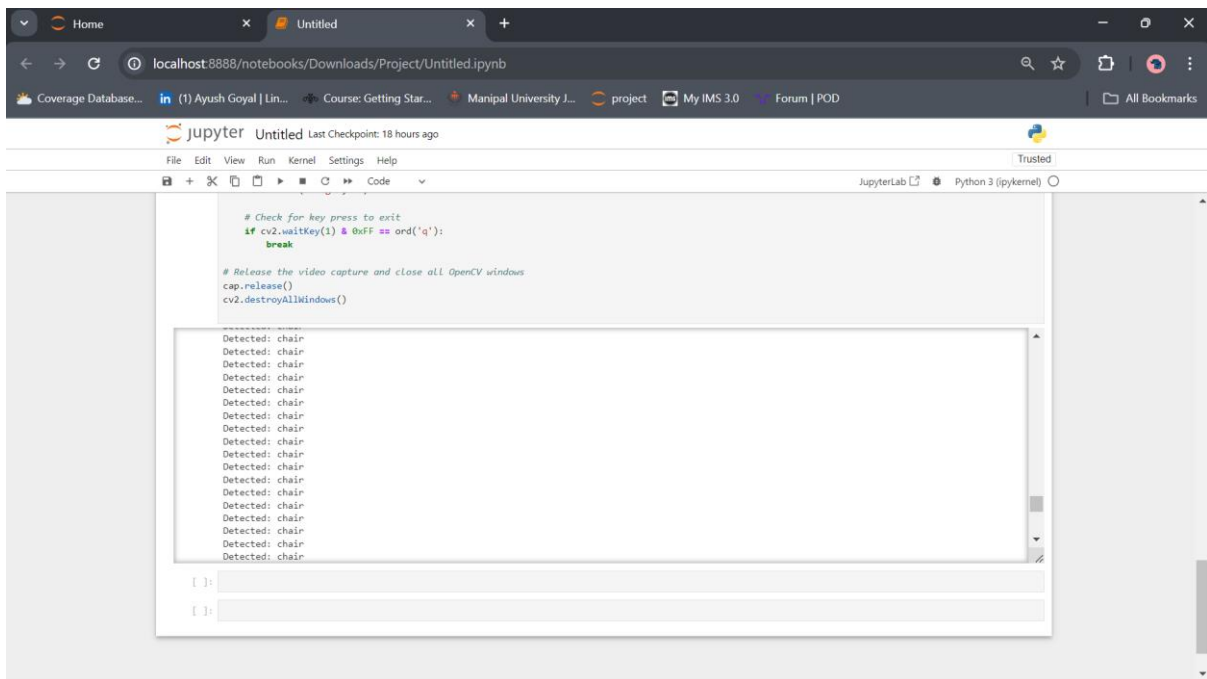
    # Get the indices of the unconnected output layers
    output_layer_indices = net.getUnconnectedOutLayers()

    output_names = [layer_names[i - 1] for i in output_layer_indices]

    # Forward pass to get the outputs
    outputs = net.forward(output_names)

    # Initialize lists to store bounding boxes and confidences
    bbox = []
    confs = []

    # Iterate through the outputs to extract bounding boxes and confidences
    for output in outputs:
```



The screenshot shows the same JupyterLab notebook, but now the code has been executed. The output of the code is a list of detected objects, which is displayed in the notebook's output area. The output is a list of 15 'Detected: chair' entries, indicating that the model has successfully detected chairs in the video frames.

```
# Check for key press to exit
if cv2.waitKey(1) & 0xFF == ord('q'):
    break

# Release the video capture and close all OpenCV windows
cap.release()
cv2.destroyAllWindows()

----- chair
Detected: chair
Detected: chair
Detected: chair
Detected: chair
Detected: chair
Detected: chair
Detected: chair
Detected: chair
Detected: chair
Detected: chair
Detected: chair
Detected: chair
Detected: chair
Detected: chair
Detected: chair
```