

B.Tech. CSE (IoT & IS)

III Year (5th Sem)

Project Based Learning – III

IS3170

Project Report

On

Integrated Interception Terminal Demonstration (IIT-D)

Group Members:

1. Devadyumna Vijay Kumar

2. Lalit M Rao

3. Navya Gupta

|  |  |
| --- | --- |
| Submitted To  Dr. Hemlata Goyal | Submitted By:  Name: Devadyumna Vijay  Kumar  Registration No.: 229311020 |

# **Index**

|  |  |
| --- | --- |
| **Name** | **Page Number** |
| OBJECTIVE | 3 |
| PROBLEM STATEMENT | 3 |
| MICROCONTROLLER STRUCTURE | 3 |
| SENSORS USED IN IIT-D SYSTEM | 6 |
| CIRCUIT DIAGRAM | 8 |
| CODE | 8 |
|  |  |
|  |  |
|  |  |
|  |  |

**OBJECTIVE**

To design and implement an autonomous system capable of detecting potential threats, analyzing their proximity, and initiating countermeasures. The project aims to simulate advanced defence mechanisms by integrating sensors, radar visualization, and real-time alert systems for rapid and efficient threat response.

**Problem Statement**

With the increasing complexity of modern threats, traditional defence systems face challenges in detecting and responding to potential dangers in real time. Delayed response times due to reliance on manual intervention can lead to significant vulnerabilities in high-stakes scenarios. There is a growing need for an automated system that can autonomously detect, analyse, and respond to threats, ensuring faster and more accurate countermeasures. Existing solutions often lack seamless integration between detection technologies, real-time visualization, and effective communication protocols. This project aims to address these gaps by developing a scalable system that combines sensors, radar interfaces, and alert mechanisms to simulate a comprehensive and responsive defence framework.

**Microcontroller Structure**

The microcontroller in the IIT-D system functions as the central hub for processing sensor data, executing programmed logic, and managing outputs to ensure seamless operation of all components. The structure is detailed as follows:  
**1. Input Modules**

The input modules capture data from the environment, which is processed to detect threats and trigger appropriate actions:

* **Ultrasonic Sensor (HC-SR04):**
  + Measures the distance to nearby objects by emitting ultrasonic waves and calculating the time taken for the echo to return.
  + Data is collected in real time and analysed to identify objects within predefined distance thresholds.
* **IR Sensor:**
  + Detects high-intensity light sources, simulating events like explosions or other high-energy occurrences.
  + Functions as a trigger for critical alerts, signalling the need for immediate countermeasures.

**2. Processing Core**

The microcontroller, a **Raspberry Pi**, acts as the central processing unit (CPU), coordinating all operations:

* Runs Python scripts for real-time data analysis and decision-making.
* **Multithreading:**
  + Ensures simultaneous execution of various functional modules, such as the radar UI updates, distance measurement, and alert system.
  + Optimizes performance by processing sensor data, updating the radar, and managing communication without delays.
* Incorporates thresholds for defining alert levels based on proximity and intensity.

**3. Output Modules**

These modules provide visual, auditory, and functional feedback based on detected threats:

* **Radar UI (Pygame Visualization):**
  + Simulates a 180-degree radar sweep to visually display the location of detected objects.
  + Updates dynamically in real time, showing red blips for detected objects relative to the ultrasonic sensor’s range.
* **LEDs and Buzzer:**
  + Visual and audio indicators to signify threat levels.
  + LEDs flash, and the buzzer activates based on proximity or intensity of detected objects.
* **Servo Motor:**
  + Drives the radar's sweeping mechanism to mimic a real-world radar system.

**4. Communication Module**

The system integrates a robust communication layer for remote alerts and command handling:

* **Twilio SMS API:**
  + Sends real-time SMS alerts to authorized personnel detailing detected threats.
  + Requests command authorization for critical decisions, such as initiating countermeasures.
  + Implements a fallback mechanism: If no response is received within a set time, automated countermeasures are activated.

**5. Power Management**

A stable power supply ensures uninterrupted operation of all components:

* Provides regulated power to sensors, actuators (servo motor, LEDs, and buzzer), and the microcontroller.
* Includes safety mechanisms to prevent power fluctuations from disrupting system functionality.

**Integrated Functionality**

* **Data Flow:** Sensors → Raspberry Pi → Processing (Python modules) → Outputs (Radar UI, Alerts, LEDs, Buzzer).
* **Synchronization:** All components are synchronized through the microcontroller, ensuring real-time detection, visualization, and response.
* **Expandability:** The structure allows for future enhancements, such as adding more sensors (e.g., LiDAR or thermal cameras) or advanced communication protocols (e.g., satellite or radio).

**Sensors Used in IIT-D System**

The IIT-D system utilizes two primary sensors, each serving a specific role in detecting potential threats and triggering appropriate responses. Below is a detailed overview of the sensors used:

**1. Ultrasonic Sensor (HC-SR04)**

* **Function:**  
  Measures the distance of objects by emitting ultrasonic waves and detecting their echoes.
* **Working Principle:**
  + The sensor emits ultrasonic waves at a frequency of 40 kHz.
  + The time taken for the waves to return after hitting an object is recorded.
  + Using the speed of sound in air, the distance is calculated as:  
    Distance=Time×Speed of Sound2\text{Distance} = \frac{\text{Time} \times \text{Speed of Sound}}{2}Distance=2Time×Speed of Sound​
* **Application in IIT-D:**
  + Continuously monitors a predefined area for objects within a specified range.
  + Triggers visual and audio alerts when objects enter critical proximity thresholds.
  + Feeds data to the radar UI for real-time visualization of detected objects.
* **Key Features:**
  + Range: 2 cm to 400 cm
  + Accuracy: ±3 mm
  + Simple integration with microcontrollers like Raspberry Pi.

**2. Infrared (IR) Sensor**

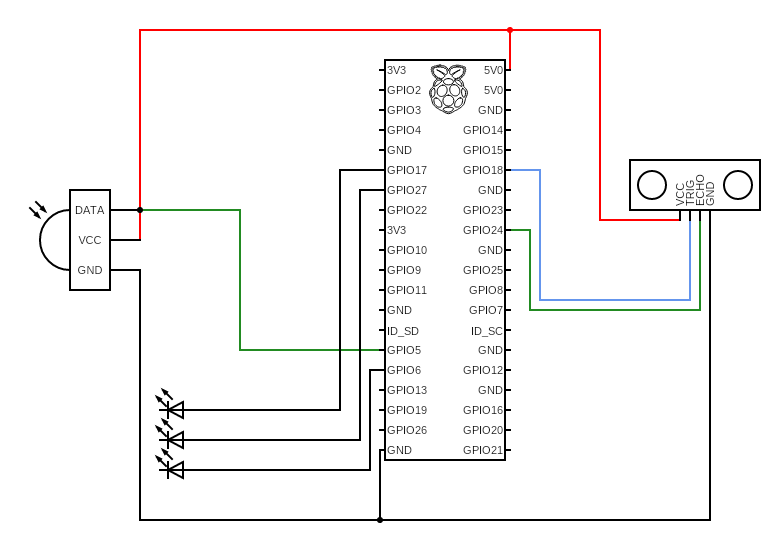
* **Function:**  
  Detects high-intensity light sources to simulate critical high-energy events such as explosions or nuclear-like scenarios.
* **Working Principle:**
  + The IR sensor detects infrared radiation emitted by light sources.
  + When exposed to intense light, the sensor outputs a signal indicating the presence of a high-energy event.
* **Application in IIT-D:**
  + Acts as a trigger for high-alert scenarios.
  + Activates the Nuclear Permission System, sending an abort request to command personnel.
  + Initiates automated countermeasures if no response is received within a designated timeframe.
* **Key Features:**
  + Sensitive to a wide range of infrared wavelengths.
  + Fast response time for critical scenarios.

**Future Sensor Integration Possibilities**

To enhance the system's capabilities, additional sensors such as thermal cameras, LiDAR, or microwave sensors could be integrated to detect and classify a broader range of threats with higher accuracy.

These sensors form the foundation of the IIT-D system, enabling real-time detection, analysis, and response to potential threats.

**Circuit Diagram**



**Codes**

**deadhand.py**

import RPi.GPIO as GPIO

import time

import threading

import pygame

import math

import random

from texts import create\_code, send\_message

# GPIO setup

GPIO.setmode(GPIO.BCM)

NUKE = 6

BUZZER = 22

IR = 5

GPIO.setup(NUKE, GPIO.OUT)

GPIO.setup(BUZZER, GPIO.OUT)

GPIO.setup(IR, GPIO.IN)

GPIO.output(NUKE, GPIO.LOW)

GPIO.output(BUZZER, GPIO.LOW)

# Radar UI setup

pygame.init()

width, height = 600, 600

screen = pygame.display.set\_mode((width, height))

pygame.display.set\_caption("180° Radar Simulation")

black = (0, 0, 0)

green = (0, 255, 0)

red = (255, 0, 0)

center\_x, center\_y = width // 2, height // 2 + 150

radius = 300

angle = 0

direction = 1

def radar\_ui():

global angle, direction

running = True

while running:

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

screen.fill(black)

# Draw radar semi-circle and concentric circles

for r in range(50, radius + 1, 50):

pygame.draw.arc(screen, green, (center\_x - r, center\_y - r, 2 \* r, 2 \* r),

math.radians(0), math.radians(180), 1)

pygame.draw.line(screen, green, (center\_x - radius, center\_y), (center\_x + radius, center\_y), 2)

# Draw radar sweep line

x = center\_x + radius \* math.cos(math.radians(angle))

y = center\_y - radius \* math.sin(math.radians(angle))

pygame.draw.line(screen, green, (center\_x, center\_y), (x, y), 2)

# Draw simulated blips

detected\_objects = [(random.randint(50, radius), random.randint(0, 180)) for \_ in range(5)]

for dist, ang in detected\_objects:

rad\_x = center\_x + dist \* math.cos(math.radians(ang))

rad\_y = center\_y - dist \* math.sin(math.radians(ang))

pygame.draw.circle(screen, red, (int(rad\_x), int(rad\_y)), 5)

angle += direction

if angle >= 180 or angle <= 0:

direction \*= -1

pygame.display.flip()

pygame.time.delay(20)

pygame.quit()

def request\_permission():

print("Requesting permission from Nuclear Command.")

code = create\_code()

send\_message("Code to abort launch sequence: " + code)

user\_input = {'code': None}

def get\_user\_input():

ans = input("Enter abort code: ")

user\_input['code'] = ans

input\_thread = threading.Thread(target=get\_user\_input)

input\_thread.start()

input\_thread.join(timeout=5)

if user\_input['code'] == code:

print("Launch sequence aborted.")

GPIO.output(NUKE, GPIO.LOW)

GPIO.output(BUZZER, GPIO.LOW)

else:

print("No valid code entered. Launching missiles!")

GPIO.output(NUKE, GPIO.HIGH)

GPIO.output(BUZZER, GPIO.HIGH)

def launch\_control():

try:

while True:

if GPIO.input(IR) == 0:

request\_permission()

time.sleep(2)

except KeyboardInterrupt:

print("Exiting program")

finally:

GPIO.cleanup()

# Run radar UI and launch control concurrently

radar\_thread = threading.Thread(target=radar\_ui)

control\_thread = threading.Thread(target=launch\_control)

radar\_thread.start()

control\_thread.start()

radar\_thread.join()

control\_thread.join()

**texts.py**

from twilio.rest import Client

import random

# Twilio account info

account\_sid = 'AC62a15209b71414ccbd43383e3d24689d'

auth\_token = 'b73153811c883b2ebf96d679809d5f71'

client = Client(account\_sid, auth\_token)

twilio\_number = '+12762262715'

# Numbers of recipients

n\_num = '+919352939475'

l\_num = '+919080356802'

d\_num = '+918905959835'

command\_numbers = [d\_num]

# Function to send a custom message to recipients

def send\_message(body):

for num in command\_numbers:

message = client.messages.create(

body=body,

from\_=twilio\_number,

to=num

)

print(message.sid)

def create\_code():

code = 0

for \_ in range(0, 6):

code = code \* 10 + random.randint(0, 9)

return int(code)

def main():

code = create\_code() # Add parentheses to call the function

send\_message("Enter the following code to cancel auto-launch sequence: " + str(code))

# Run main function

main()