Laser Sentry Guard

Nume si Prenume: Stoica Mihai-Bogdan
Grupa: 335CA

Introduction

This project features an Arduino Uno (ATmega328P)-based autonomous sentry system inspired by the defensive turrets from **Helldivers 2**. The system is built around a finite state machine that handles idle scanning, movement detection and target tracking.
A potentiometer allows real time adjustment of the system's aggression level. Additionally, a laser is used to visually indicate the direction the sentry is currently facing.

For a better understanding of how such a sentry behaves, check out this link:
https://helldivers.fandom.com/wiki/Sentries

General Description

The sentry operates by continuously rotating to scan its surroundings while in an idle state. When motion is detected, it transitions into an active tracking state, aligning itself with the target and maintaining focus as long as movement persists. The behavior and response time of the system are influenced by the aggression level, which can be manually adjusted using a potentiometer. This control affects how quickly the sentry reacts and how long it remains alert after detecting movement. A laser mounted on the sentry provides a visual cue, pointing precisely in the direction the system is currently facing, simulating a targeting mechanism.

An initial block diagram of the system is shown below:

Aggression Control and Behavior

The aggression control allows the user to adjust the sentry's responsiveness to motion using a potentiometer. Higher aggression levels make the sentry more sensitive and reactive, quickly transitioning to tracking mode even with small movements, while lower settings result in slower, more passive reactions. This control offers flexibility, allowing the sentry to adapt to different scenarios, from cautious monitoring to high-alert defense.

Servo Selection: MG996R 360°

For actuation, I chose the **MG996R 360° servo** because I am unsure of the weight it will need to support. This servo is known for its high torque, providing flexibility for adjusting based on the weight of the system and components added later. It will ensure smooth and precise movement when tracking the detected motion.

Hardware Design

The following table lists the main components used in this project along with their corresponding links(Digi-Key or Mouser):

Component	Description	Digi-Key Link
Arduino Uno	Microcontroller board	Link
HC-SR04 Sensor	Ultrasonic distance	Link
AM312 Sensor	PIR motion detection	Link
MG996r Servo	360° rotation servo	Link
Potentiometer	Aggression control	Link
Laser	Visual target	TODO
Prototyping Board	A perforated prototyping board	Link
Battery Support	Support for 4xAA batteries	Link

I have designed the schematic for the system using Fusion 360. The schematic can be found below:

Sensor Placement and Configuration

This project utilizes two HC-SR04 ultrasonic distance sensors positioned at a 90-degree angle relative to each other to minimize signal interference and eliminate blind spots in coverage. The sensors are placed with a 6 cm gap between their centers, providing optimal spacing for accurate distance measurement without overlap. Positioned centrally between the two ultrasonic sensors is the AM312 PIR motion sensor. This strategic layout ensures that the PIR sensor effectively detects motion in the area covered by both HC-SR04 units, enabling reliable activation and tracking of nearby movement.

The entire sensor assembly is mounted on a servo motor, which allows it to rotate and scan the environment. The servo motor and the microcontroller (MCU) are powered from separate power sources to ensure stable operation and prevent voltage drops during movement.

The pins used for the assembly are:

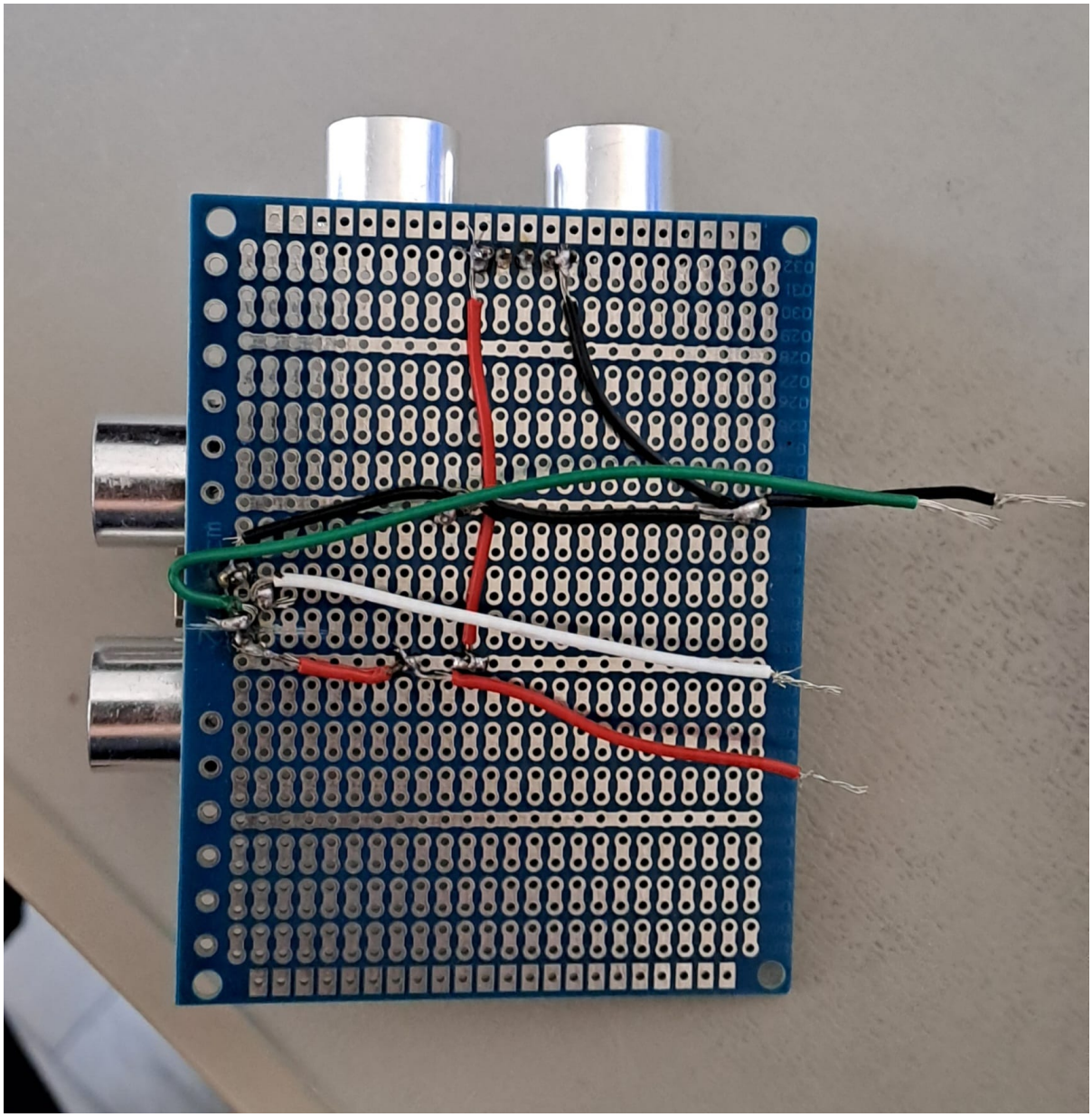
- D2, D3 - sensor 1, HC-SR04

- D6, D7 - sensor 2, HC-sr04
- D8 - sensor 3, PIR(am312)
- D9 - High Torque Servo Motor(360), MG996r

Why this configuration?

I decided to go with this setup for the following reasons: Pins D5 and D4 are related to UART functionality and tend to interfere with the proper operation of the sensors, likely due to timing or signal conflicts during serial communication. D9 is well-suited for controlling the servo motor because it's a PWM-capable pin, which ensures smooth and accurate rotation. D8 is a good choice for the PIR sensor because it supports PCINT0 (Pin Change Interrupt), allowing motion detection without relying on the limited external interrupt pins (D2 and D3), which are already in use. This configuration balances the hardware limitations of the Arduino Uno while maintaining responsiveness and reliability across all components.

To simplify wiring and ensure a stable power supply, I will solder the GND and 5V lines of all sensors together into common rails. This will reduce cable clutter and improve reliability.



Software Design

The firmware is written in C, using direct AVR register manipulation for all peripherals such as timers, ADC, EEPROM, and interrupts. The project is developed using the Arduino IDE, relying on few(digitalWrite) Arduino functions. This environment is used solely for its ease of compilation and flashing, while the core logic is entirely low-level and bare-metal. The target microcontroller is the ATmega328P, programmed via a USB-to-serial interface using the standard Arduino bootloader.

Key functionality includes:

- Motion detection using interrupts via the PIR sensor (Pin Change Interrupt on PCINT0)
- Distance measurement with two ultrasonic sensors triggered via GPIO and read through timer logic (Timer0)
- A potentiometer connected to an analog pin for aggression level adjustment, read using the ADC
- Servo movement handled using PWM generated by a 16-bit timer for precise control of rotation and

alignment
- Serial debug output through USART, showing distance and aggression level values
- A watchdog timer configured in interrupt mode to periodically reset the system's motion flag, simulating a return to idle state if no movement persists

Code structure:

- All sensor initialization and control functions are encapsulated for readability
- Debugging via USART is included with optional output toggled by the `DEBUG` macro
- The `movement` struct stores motion and sensor states, aggression value, and system flags
- The main loop continuously checks for motion, reads distances and aggression level, and responds accordingly

The design ensures modularity, debuggability, and responsiveness, tailored to the project's behavioral logic.

Design Decisions

The HC-SR04 ultrasonic sensors are read using polling instead of interrupts. Since their echo durations are very short, polling ensures fast and accurate readings without the risk of stack corruption or interrupt conflicts, which could occur due to overlapping or nested ISR executions.

On the other hand, interrupts were used for the PIR sensor (via PCINT0) and the watchdog timer (WWDT). The PIR interrupt enables real-time motion detection without constantly checking the pin in the main loop, while the WWDT interrupt helps manage system state by resetting the motion flag after a predefined timeout, ensuring the sentry returns to idle mode if no motion persists.

Operating Modes: Radar and Sentry

The system supports two distinct operating modes: **Radar Mode** and **Sentry Mode**. The currently active mode is stored in the internal **EEPROM** of the ATmega328P and is read at startup to configure behavior.

- **Radar Mode ('R')**

The servo continuously sweeps between its angle limits (e.g., 0° to 180°), mimicking a scanning motion similar to radar. This mode ignores sensor input and focuses on regular motion patterns.

- **Sentry Mode ('S')**

The system remains idle until motion is detected using the PIR sensor. Once triggered, the servo turns toward the direction with the closest detected object (using data from the ultrasonic sensors).

- **Pause Mode ('P')**

The system remains idle and does not move the servo. It continuously reads the distance sensors to identify the closest object but does not perform any servo movement or motion-triggered actions.

The modes are stored in EEPROM, enabling persistent configuration even after a power cycle. EEPROM access is performed via direct register manipulation (`EEAR`, `EEDR`, `EECR`) to minimize overhead and increase speed.

This dual-mode architecture allows the system to behave either:

- **Proactively**, scanning its surroundings (Radar)
- **Reactively**, responding only when motion is detected (Sentry)

Source Code and Demo

The complete project is available on GitHub:

- **Main Repository**: https://github.com/bogdanstoicasn/laser-sentry-guard

The firmware is written in **C** and developed using the **Arduino IDE**, but it relies on **direct register manipulation** for full control over peripherals.

- The main project logic is located in the `ardu.ino` file.

A simplified **sensor demo** project is also included, showing how to read from:

- The ultrasonic sensors (HC-SR04)
- The PIR motion sensor
- The potentiometer via ADC

It is located in `main.c` and uses a custom driver lib made by me(**avratlib**), located at:

- https://github.com/bogdanstoicasn/mcus-driver-lib

The **demo video** is available in the main repository as well:

- **Demo Video**: https://github.com/bogdanstoicasn/laser-sentry-guard

Resources

- **`avratlib` custom driver library**

https://github.com/bogdanstoicasn/mcus-driver-lib

- **Project inspiration – Sentry Turret (Helldivers)**

https://helldivers.fandom.com/wiki/Sentries

- **ATmega328P Reference Manual**

ATmega328P Datasheet

Export to PDF

From:
http://ocw.cs.pub.ro/courses/ - **CS Open CourseWare**

Permanent link:
http://ocw.cs.pub.ro/courses/pm/prj2025/abirlica/mihai_bogdan.stoica

Last update: **2025/05/29 20:06**