Fall 2024 Electrical and Computer Engineering 301
Circuits/Electromechanical Components
ECE 301
December 6, 2024, Knoxville, TN, USA

ECE301

ULTRASONIC RADAR PROJECT: PROXIMITY-BASED ALARM SYSTEM

Patrik A. Sehulster[1]

[1]Mechanical Engineering - University of Tennessee, Knoxville, USA

ABSTRACT

This project seeks to apply the sonar capabilities of the Arduino Uno to create a proximity-based alarm system that can be used in a number of applications, though primarily intended for car safety features. Created utilizing the Arduino Starter Kit, the Arduino IDE, and the Processing IDE with a computer, this alarm system will activate if an object passes within 30 cm of its sensor. The Arduino board is programmed using the C++ language, while the processing code is written in JavaScript. There are two different codes written for the Arduino: one for a rotating sensor, which could primarily be used for security purposes, and one for a stationary device to be used for vehicular anti-collision systems. Through testing, it was discovered that both of these codes work as intended. With refinement, they could be used in conjunction with existing technology, or implemented on top of older systems that do not yet integrate modern technology.

Keywords: Arduino, Processing, Alarm, Sensor, Application, Rotating, Stationary

1. INTRODUCTION

There are countless proximity alarm systems that are currently in place to deter theft and other crime. However, the technology is making its way into new fields: newer car models have proximity sensors to avoid collisions, for example. In the United States, there are over 6 million car collisions every year, leading to over 38,000 deaths and 2.3 million injuries [1]. Car companies, and the government, are working to combat this on their newer vehicles: new regulations mandate that all newly produced cars must implement these sensors to their vehicles, and Market.US estimates the global automative sensor market reached $26 billion in 2024, and projects that this number will more than double in the next eight years [2].

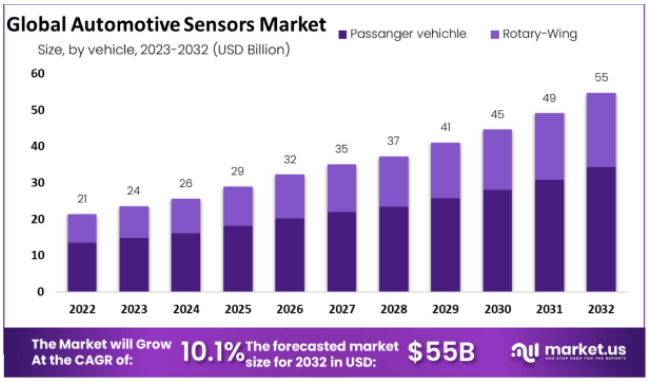


FIGURE 1: ESTIMATED FUTURE PROJECTIONS OF THE GLOBAL AUTOMOTIVE SENSORS MARKET [2].

With these projections in mind, a cheap and easy-to-develop proximity sensor would greatly benefit this industry. However, the main focus of this project is post-production implementation. While new cars will be less likely to get into an accident, the existing vehicles on the road without these safety features are left behind and rely on only the driver's abilities to avoid crashes. By creating a simple code that can be integrated into the existing system in the car, public safety would exponentially increase.

With a wide scope of applications, the usefulness of sonic radar is limitless. This project attempts to utilize this usefulness to develop a system that, with minor code adjustments, can be integrated into any of these applications. This is why this project develops two separate designs: although its primary purpose is for car sensors, implementing the technology for another use displays the ease of application and modification to suit the user's needs. There are only ten lines of code separating the difference between the rotating & stationary sensors, and the entire project itself is easily repeatable with the required materials.

2. MATERIALS AND METHODS

2.1 Materials

© 2019 by ASME

Materials & Software required:

- Arduino Uno Starter Kit
  - Arduino Uno Board
  - Breadboard
  - Servo Motor
  - Ultrasonic Sensor
  - Wires
- Arduino IDE software
- Processing IDE software
- Computer compatible with Arduino & Processing IDEs
- Cardboard

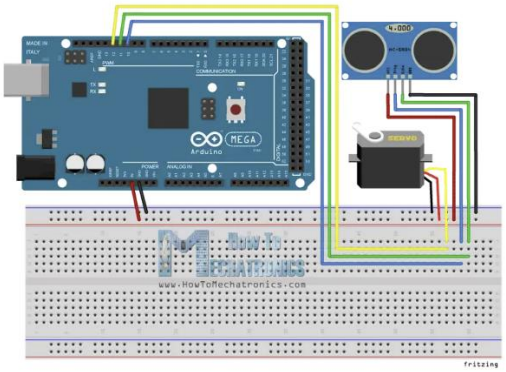


FIGURE 4: Circuit Schematic for Proximity-Based Alarm [3]

2.2 Methods

The design process for this project was relatively simple. Because it was going to be utilizing the circuit and code configuration from an open-sourced Arduino Radar Project [3], a proximity based alarm was the only design considered. The idea of creating both a stationary sensor and a rotating sensor was also a relatively simple decision, as it allowed the analysis and comparison of the two products to determine which would be more effective in achieving its respective purpose. However, there were countless obstacles that came with the modifications necessary to create this alarm system. When initially creating the if loop for the alarm sound, there was no delay preventing the sound from playing on repeat. Using generative AI, a new loop was created to avoid this issue. Another problem that arose during the development of this project was in adapting the processing code to work with both of the different Arduino codes. Initially, the system would immediately sound the alarm, even when there was no object in range. This was occurring because the stationary code initially did not include a section of code from the original, which recorded each individual distance at each individual angle and stored these data points within the Arduino. This was initially deleted because the angle was not changing, but the result of that deletion was that the distance data had nowhere to be stored, and the Processing IDE was pulling the data already stored within the Arduino. By doing this, the Processing IDE also had no way of knowing the data point it was supposed to be reading – and the alarm would set if *any* of the data points were within the given range. By reimplementing these lines of code, the processing IDE began reading only the most recent distance calculation in order to determine whether or not to sound the alarm.

There are three parts to the creation of this project: the physical circuit, the two separate Arduino codes, and the processing code.

2.2a Circuit Configuration

The physical alarm system was created using the wire configuration of an open-source Arduino Radar Project [3]. A cardboard enclosure is built to house the servo motor and the attached sensor, and the circuit is assembled as follows:

2.2b Arduino Codes

There are two Arduino IDE codes for this project – both nearly identical to that of the open-source radar available online, with minor adjustments to account for each given application.

The first code is for a sensor designed to rotate 350 degrees and alert when any object is detected within its range. This code would be used for anti-theft applications – such as those utilized in banks, homes, or stores. It is designed to constantly rotate back and forth between its two extremes and includes a function to continuously calculate and store distance. This is achieved using repeating loop functions to rotate between the minimum & maximum rotation angle. The calculateDistance function works by intermittently sending out signals from the sensor's trigPin. It then reads the echoPin's duration, or time taken for the signal to echo back to the sensor, and to calculate the distance between itself and the object it is facing.

The complete source code for the rotating sensor is as displayed:

```
// Includes the Servo library
#include <Servo.h>.

// Defines Tirg and Echo pins of the Ultrasonic Sensor
const int trigPin = 10;
const int echoPin = 11;
// Variables for the duration and the distance
long duration;
int distance;

Servo myServo; // Creates a servo object for controlling the servo motor

void setup() {
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input
  Serial.begin(9600);
```


© 2019 by ASME

```
mServo.attach(12); // Defines on which pin is the servo motor attached
}
void loop() {
// rotates the servo motor from 5 to 355 degrees
for(int i=5;i<=355;i++){
myServo.write(i);
delay(30);
distance = calculateDistance();// Calls a function for calculating the distance measured by the Ultrasonic sensor for each degree

Serial.print(i); // Sends the current degree into the Serial Port
Serial.print(","); // Sends addition character right next to the previous value needed later in the Processing IDE for indexing
Serial.print(distance); // Sends the distance value into the Serial Port
Serial.print("."); // Sends addition character right next to the previous value needed later in the Processing IDE for indexing
}
// Repeats the previous lines from 355 to 15 degrees
for(int i=355;i>5;i--){
myServo.write(i);
delay(30);
distance = calculateDistance();
Serial.print(i);
Serial.print(",");
Serial.print(distance);
Serial.print(".");
}
}
// Function for calculating the distance measured by the Ultrasonic sensor
int calculateDistance(){

digitalWrite(trigPin, LOW);
delayMicroseconds(2);
// Sets the trigPin on HIGH state for 10 micro seconds
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);
duration = pulseIn(echoPin, HIGH); // Reads the echoPin, returns the sound wave travel time in microseconds
distance= duration*0.034/2;
return distance;
}
```

The second code is for a sensor designed to remain stationary at its set angle (in this case 90 degrees). The code is almost identical to the rotating system, but without the loop to continuously rotate back and forth. It simply sits facing forward and calculates the distance between it and the object ahead of it. It is designed to be a mounted sensor, such as at the front of a car or on the surface of expensive equipment, in order to prevent collisions.

The complete source code for the stationary mounted sensor is as displayed:

```
// Includes the Servo library
#include <Servo.h>.

// Defines Tirg and Echo pins of the Ultrasonic Sensor
const int trigPin = 10;
const int echoPin = 11;
// Variables for the duration and the distance
long duration;
int distance;

Servo myServo; // Creates a servo object for controlling the servo motor

void setup() {
pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
pinMode(echoPin, INPUT); // Sets the echoPin as an Input
Serial.begin(9600);
mServo.attach(12); // Defines on which pin is the servo motor attached
}
void loop() {
int i = 90 ;
myServo.write(i);
delay(30);
distance = calculateDistance();
Serial.print(i);
Serial.print(",");
Serial.print(distance);
Serial.print(".");
}

// Function for calculating the distance measured by the Ultrasonic sensor
int calculateDistance(){

digitalWrite(trigPin, LOW);
delayMicroseconds(2);
// Sets the trigPin on HIGH state for 10 micro seconds
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);
duration = pulseIn(echoPin, HIGH); // Reads the echoPin, returns the sound wave travel time in microseconds
distance= duration*0.034/2;
return distance;
}
```


© 2019 by ASME

2.2c Processing Code

Both the stationary and the rotating sensor use the same processing code, as the chosen proximity distance for this project was the same for both applications – for testing this distance was 30 cm. The code is designed to read the immediate calculated distance from the Arduino sensor and play an alarm sound through the computer whenever an object moves in range of the sensor. The code essentially reads the measured angle and distance of the sensor continually, which are both already calculated and stored in the Arduino's own code. It then contains an if statement that, whenever an object is detected by the sensor, tells the computer to begin playing an alarm .mp3 file [4]. There is an additional if statement, helped written by generative AI, embedded within the first to ensure that the alarm will not replay and restart if it is already running. This prevents the system from overloading and playing the file on a loop.

The complete processing code for both proximity sensors is as displayed:

```
import processing.serial.*; // imports library for serial communication
import java.awt.event.KeyEvent; // imports library for reading the data from the serial port
import java.io.IOException;

Serial myPort; // defines Object Serial
// defines variables
String angle = "";
String distance = "";
String data = "";
String noObject;
float pixsDistance;
int iAngle, iDistance;
int index1 = 0;
int index2 = 0;
PFont orcFont;
import processing.sound.*;
SoundFile file; // Sound file variable

void setup() {
  size(1920, 1080);
  smooth();
  myPort = new Serial(this, "COM3", 9600); // starts the serial communication
  myPort.bufferUntil('.'); // reads the data from the serial port up to the character '.'. So actually it reads this: angle, distance.

}

void serialEvent(Serial myPort) { // starts reading data from the Serial Port
  // reads the data from the Serial Port up to the character '.' and puts it into the String variable "data".
  data = myPort.readStringUntil('.');
  data = data.substring(0, data.length() - 1);

  index1 = data.indexOf(","); // find the character ',' and puts it into the variable "index1"
  angle = data.substring(0, index1); // read the data from position "0" to position of the variable index1 or that's the value of the angle the Arduino Board sent into the Serial Port
  distance = data.substring(index1 + 1, data.length()); // read the data from position "index1" to the end of the data, or that's the value of the distance

  // converts the String variables into Integer
  iAngle = int(angle);
  iDistance = int(distance);
 }

 void draw() {
  // Continuously check the distance in the draw function
  if (iDistance < 30) {
   // Load a soundfile from the /data folder of the sketch and play it back


   if (file == null || !file.isPlaying()) {  // Check if the sound is not already playing
    file = new SoundFile(this, "Alarm Sound Effect.mp3");
    file.play();
   }
  }
}
```

3. RESULTS AND DISCUSSION

Both codes worked almost exactly as designed – although each had their own flaws as well. The rotating sensor was tested by placing it down in the center of an empty room and moving objects around it. Any time a person or an object was in front of the sensor, and in range, the alarm would begin to sound. However, the sensor only rotated about 180 degrees – it was unable to do a full circle, which is likely attributed to limitations on the Servo Motor itself. To account for this, the rotating sensor should only be used as an alarm system when facing away from a wall. The code should also be adjusted to rotate between 5 and 175 degrees (rather than its current 5 and 355) to ensure that the sensor's motion is continuous. Otherwise, its range of motion can simply be circumvented, and its effectiveness as a deterrent would fall to zero. In a real world application, it is unlikely that this current design would be effective – as it begins the alert every time there is *anything* in its range. For example, a wall within the sensor's range would currently set off the alarm, as would furniture such as chairs or desks. If implementing as a real security measure, the code must first be updated to ignore objects

© 2019 by ASME

that are supposed to be there – and only alert if something moves or if a new object appears. This could be done by setting an additional if statement in the processing code that tells the alarm to ignore the angles and distances of each object that is meant to be in the sensor's range. With these obstacles in mind, the use of the rotational proximity alarm system would likely be less effective than a simple camera or motion detector, especially in typically crowded areas.

In order to implement the rotating sensor to a real-world application, such as home security, it could implement a new if loop to sound the alarm only with motion. On its first rotation, the sensor can simply record the distances and positions at each angle. On each future rotation, it continues to record – but instead of having an if loop for the alarm to go off with any recorded obstacle the sensor's range, it would have one that subtracts the new data point from its stored one. If the difference is greater than 0, it means that an object has moved – and the alarm can then go off. If the difference is 0, the object remained where it was, and the computer can stay silent. This would allow for the rotating sensor to be used in a room where nothing should be moving – such as for home security when away, or a bank vault. It could also be coded to alert law enforcement when set off for a set amount of time, or when set off multiple times in a row. However, the system would need to be turned off every time that a person is meant to enter the room or house, which is true for every motion detector alarm in production.

×

Report Attributes	Details
Market Size in 2023	USD 2.52 Billion
Market Size by 2032	USD 5.91 Billion
CAGR	CAGR of 9.9% From 2024 to 2032
Report Scope & Coverage	Market Size, Segments Analysis, Competitive Landscape, Regional Analysis, DROC & SWOT Analysis, Forecast Outlook
Key Segments	• By Type (Wired Alarm System, Wireless Alarm System, Hybrid Alarm System) • By Component (Hardware, Alarm Sensors, RTU, Central Monitoring Receiver, Motion Detection Sensor, Others, Software, Services) • By Application (Residential, Commercial & Industrial)
Key Drivers	• Increasing Need for Home Intruder Alarm Systems • Increasing demand for intruder alarm systems is being fueled by advancements in technology.

FIGURE 3: INTRUDER ALARM SYSTEM MARKET PROJECTIONS [5]

According to Figure 3, the global market for home security systems in 2023 was valued at $2.52 billion – and is expected to more than double over the next eight years. With the necessary adjustments made to the code, this would provide consumers with a cheap alternative to motion detector alarms – but could also be implemented in other facets of home security. For example, many systems are designed to shine a floodlight whenever motion is detected. With minor adjustments, the alarm sound loop can be replaced with a loop to do anything from turning on lights to engaging electric locks.

The stationary sensor's testing suggests that this application is much more real-world ready than its rotating counterpart. For testing this, the sensor was faced forward and slowly walked directly towards a wall. As soon as the wall was in range, the alarm would sound. The primary issue with this design was that once the sensor's motion stopped, the alarm would continue to run because the wall in front of it was still in range. To account for this in a real world application, the code can be modified to only play the alarm when the distance is within range *and* smaller than the previous distance. This would ensure that once stopped, the alarm would stop sounding – but while approaching an object it would continue to run. Another test was conducted outdoors with the sensor facing the wind to ensure that the wind would not affect the sonic capabilities of the sensor. If the wind interfered with the distance tracking, then hope of this technology implemented in cars would be all but lost. However, moving air did not appear to affect the system at all – and so the second test was a success.

The market projections from Figure 1 display the estimated future of the automotive sensor market, but only for newly produced cars. The mandate that all new cars must have distance sensors embedded in their technology does not account for the vehicles already on the road. With the technology only becoming widespread in the early 2000s [6], there are countless cars on the road driving every day without any modern safety features. This is where the idea for this product comes in: post-market implementation.

The stationary proximity alarm as it stands right now, if implemented on the front of one of these older models, could be used as a warning for the driver to begin braking. Of course, the distance limit would need to be reconfigured to the actual braking distance of the car – but with a stronger sensor this would theoretically be possible. In future design projects, this technology could theoretically be wired directly into the cars themselves – and with further coding, moving within the sensor's range of another vehicle or obstacle could enable an automatic braking system to activate much like the car models being produced today do. If implementing the stationary sensor into a car, the code would need to be able to determine its own speed and constantly update its braking distance to account for the motion of the car – a car at 30 mph takes much less time and distance to brake than the same car at 75 mph. By creating a technology that improves the safety of vehicles already on the road, rather than only implementing the mandated technology on newly manufactured vehicles, the number of auto collisions (and injuries/deaths) would drastically decrease. This technology could also be used on cars for parking sensors, preventing the low points of a car from scraping the curb, or even in place of backup cameras.

© 2019 by ASME

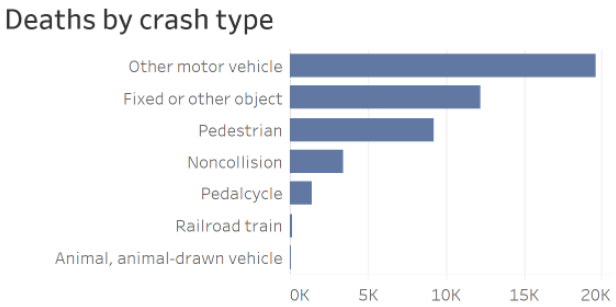


FIGURE 4: DEATHS BY COLLISION TYPE [7]

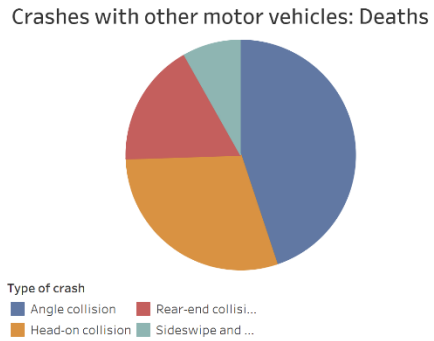


FIGURE 5: DEATHS BY COLLISION TYPE [7]

Figures 4 and 5 display the percentages of motor vehicle deaths by collision type. As displayed, crashing into other motor vehicles is by far the deadliest type of crash, accounting for 42.6 of all motor vehicle deaths in 2022. Of these vehicle-on-vehicle collisions, the deadliest are angular (44.9% of deaths), head-on (29.6%), and rear-end (17.3%) [7]. With this in mind, it can be concluded that the majority of collisions are avoidable by diligent driving, safe following distances, and safe speeds. However, it cannot be assumed that all drivers will follow these basic rules of the road, which is why the production mandate exists to compensate for poor or unsafe driving.

The majority of vehicular collisions are caused by one or both drivers failing to brake in time – which newer cars are able to combat with their own sensors and automatic braking systems. However, the older car models on the road do not have the same safety features, and so the only thing preventing them from rear-end collisions is their drivers' reaction time.

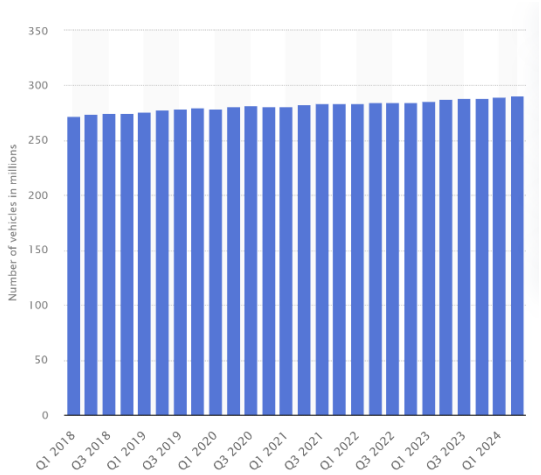


FIGURE 5: NUMBER OF CARS ON US ROADS BY QUARTER [9]

Furthermore, it is estimated that approximately 23% of vehicles on the road are over 20 years old [8]. This means that more than one-fifth of everyday drivers are without any modern safety features in their vehicles. According to Figure 5, there are approximately 291 million vehicles operating on the roads in the US, with this number increasing every quarter since 2018 [9]. This translates to at least 67 million cars without modern safety features, and likely many more that were produced before the safety features became mandated for all new productions. Although there is a finite number of cars without said features, it is a massive untapped market – and if the code can be adjusted to account for the vehicle's speed when calculating braking distance, the product would theoretically be implementable on every car.

This simple code can effectively be altered to cater to either of its two intended purposes. With minor modifications to either of the two Arduino codes, they would both be ready for real-world applications. There is a massive market for both applications tested in this project,

4. CONCLUSION

This project successfully created a proximity-based alarm system that can be implemented in a number of ways. Although it is less effective as a rotating security system, modifications can be made to the code in order to develop a system that detects motion rather than stationary objects. For the stationary sensor alarm, it can be implemented immediately on cars as a front-facing system with a set braking distance *or* modified to account for the car's immediate speed. With proper implementations, the usability of this simple technology is nearly limitless.

ACKNOWLEDGEMENTS

Arduino IDE C++ code was modified with help from generative AI.

© 2019 by ASME

Processing IDE JavaScript code was modified with suggestions from processing.org.

REFERENCES

1. Russell, R. *et al. Car accident statistics you need to know in 2024*, *Darrigo & Diaz*. Available at: https://www.ddlawtampa.com/resources/car-accident-statistics-you-need-to-know-in-2021/#:~:text=Total%20number%20of%20car%20accidents%20per%20year%20in%20the%20U.S.&text=This%20is%20up%20from%20238.6,losing%20their%20lives%20each%20year (2024).
2. Pangarkar, T. *Automotive sensors statistics 2024 by New Sensing Tech*, *Market.us Scoop*. Available at: https://scoop.market.us/automotive-sensors-statistics/ (2024).
3. *Arduino Radar Project*, *How To Mechatronics*. Available at: https://howtomechatronics.com/projects/arduino-radar-project/ (2022).
4. *Alarm Sound Effect Sound Effects*. Available at: https://www.youtube.com/watch?v=5nRgCabardA (2022).
5. SNS Insider. *Intruder alarm system market to reach USD 5.91 billion at 9.9% CAGR by 2032, driven by rising security concerns: Exclusive Report by SNS insider*, *GlobeNewswire News Room*. Available at: https://www.globenewswire.com/news-release/2024/10/03/2957738/0/en/Intruder-Alarm-System-Market-to-Reach-USD-5-91-Billion-at-9-9-CAGR-by-2032-Driven-by-Rising-Security-Concerns-Exclusive-Report-by-SNS-Insider.html (2024).
6. *The evolution of Vehicle Sensor Systems IAA MOBILITY*. Available at: https://www.iaa-mobility.com/en/newsroom/news/future-technology/the-evolution-of-vehicle-sensor-systems (no date).
7. *Motor Vehicle - Type of Crash - injury facts*, *NSC Injury Facts*. Available at: https://injuryfacts.nsc.org/motor-vehicle/overview/type-of-crash/ (2024).
8. *Analysis of average age of cars and trucks on the road [2024]*, *Hedges & Company*. Available at: https://hedgescompany.com/blog/2024/02/average-age-of-cars-trucks/#:~:text=About%2023%25%20of%20all%20passenger,20%20years%20old%20or%20older. (2024).
9. Carlier, M. *U.S.: Vehicles in operation 2023*, *Statista*. Available at: https://www.statista.com/statistics/859950/vehicles-in-operation-by-quarter-united-states/#:~:text=In%20the%20fourth%20quarter%20of,roads%20throughout%20the%20United%20States. (2024).

© 2019 by ASME