Object Detection using SONAR based system



Report submitted for

Mini Project (6th Semester)

in partial fulfillment of the degree of

Bachelor of Technology

Electronics and Communication Engineering

By

SHUBHAM DOGRA &VINIT JOON

Title:

Object Detection using SONAR based sys-

Theme:

Embedded Systems

Semester 6TH

Page Numbers: 42

Date of Completion:

20 May 2022

DPG INSTITUTE OF TECHNOLOGY AND MANAGEMANT

The goal is to develop an object detection system which could determine the presence of objects in it's vicinity

The system leverages the HC-SR04 [Il trasonic Sensor mounted on top of a Micro-Seno motor, both being controlled by an Ardnino LINO. The HC-SR04 detects its separation from an obstacle placed in front of it and sends the data to the Arduino, while the servo rotates the sensor about its axis in order to have a field of view that svveeps mer a sector with a radius specified within 4 metres.

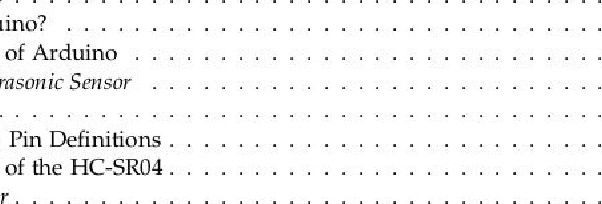
The Arduino recieves sensor readings (distance measurements) from the sensor and sends it via LISART to a computer for graphical display using the Processing

IDE.

The content of this wport is freely nvnilnble, but publication (with mfennce) may only bé pursued due to agreement apith the author.



Contents

1. Introd uc tion 2
   1. The UNO3
      1. Why Arduino? 3
      2. Operation of Arduino 5
   2. The HC-SR04 Ultrasonic Sensor 5
      1. Features5
      2. HC-SR04 : Pin Definitions 6
      3. Operation of the HC-SR04 . . 7
   3. Micro-Servo Motor . . 8

|  |  |
| --- | --- |
| 2 Circuit Schematics | 11 |
| 2.1 Arduino Pin Connections | 11 |

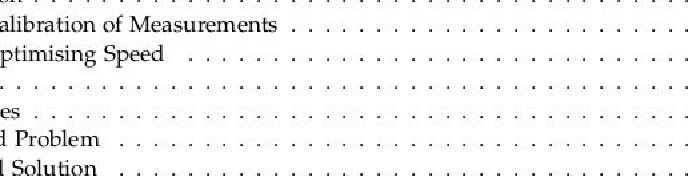
* 1. Breadboard View11
  2. Schematic view12

|  |  |
| --- | --- |
| 3 Development and Simulation | 13 |
| 3.1 IDE | 13 |

* 1. Simulation with Proteus 14
  2. Processing14

|  |  |  |
| --- | --- | --- |
| 4 Explaining the Codes |  | 16 |
| 4.1 The Arduino Code |  | 16 |
| 4.2 The Processing Code |  | 18 |
| 5 Observations |  | 19 |
| 5-1 Calibration . | | 19 |

19



5.1.1

Calibration

51.2

Optimising

20

* 1. Graphs 21
  2. Challe nges 24
  3. Identified Problem 24
  4. Proposed Solution 24

6 Conclusion 30

A A cknowle dgemen ts 31

V

Contents

[B The Team 32](#_Toc158049)

[C Appendix A : Arduino Code 33](#_Toc158050)

[D Appendix B Processing Code 36](#_Toc158051)

1 Introduction

The basic approach of our project, just like any other based on embedded systems (microcontrollers) largely depends on three main questions:

* Which microcontroller board would be appropriate?
* What sensor(s) should be used to fulfill the goals of the project?
* What actuators might be needed for introducing mechanical action, if needed?

We had a literature survey through the internet about deciding the components needed for our project. This included studying in detail, several project ideas and instructions posted on blogs and YouTube.

We selected the following major components for having a minimal cost effective and efficient apparatus that could do the job for us:

* Microcontroller Board - Arduino LINO
* Sensor - HC-SR04 Ultrasonic Sensor
* Actuator - Micro-Serva Motor

The next few sections will explain the basis of selection of hardware for this project.

2

1. 1. UNO

1.1 The Arduino UNO

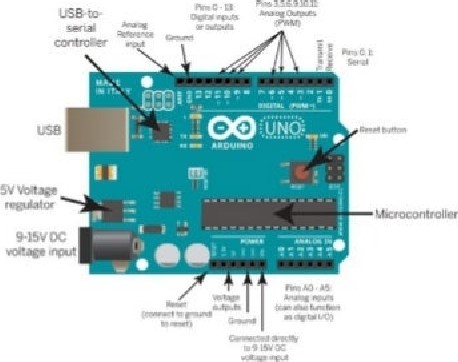


Figure The Arduino UNO

Arduino is an open source electronics platform for fast prototyping of projects for users with minimal knowledge or experience in electronics and programming. We have used the Arduina IINO, the most widely used variant of the Arduino.

Techmcally, Arduino Uno is a microcontroller board based on the 8-bit ATMega328P microcontroller. With 14 digital input/output pins (including 6 PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a povver jack, an ICSP header. and a reset button, it has everything needed to support the microcontroller; we simply have to connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

Arduino also comes with the Arduino IDE, in which we can write code and upload to the Arduino. The programming language used for an Arduino, called the Arduino Programming language, is very similar to C/C++ except that we can use inbuilt functions of the Arduino libraries which keep the code very simple and short.

1.1.1 Why Arduino?

We had the following reasons strongly backing our decision to choose the Arduino [INC):

Support - Thanks to its simple and accessible user experience, Arduino has been used in thousands of different projects and applications. The Arduino software is easy-to-use for beginners, yet flexible enough for advanced users. It runs on Mac, Windows, and Linux.

This has enabled a very large support base for Arduino, more than any other microcontroller board ever released in the market. Every problem/bug/question that we ever had throughout the course of this project was promptly answered by questions in online forums, as there was a very high probability that scores of people had already posted the same question and the issue was resolved by other experienced users.

1. Introduction

|  |  |  |
| --- | --- | --- |
|  | Microcontroller | ÄTmega328P(8-bit) |
|  | Operating Voltage | (DC)5V |
|  | Inpu t Voltage | (recommended) 7-12V; (limits) 6-20V |
|  | Digital 1/0 Pins | 14 (of which 6 provide PVVM output) |
|  | Analog Input Pins | 6 |
|  | DC Current per 1/0 Pin | 40 |
|  | DC Current for 3.3V Pin | 50 Ma |
|  | Flash Memory | 32 KB (ATmega328) of which 0.5 KB used by bootloader |
|  | SRAM | 2 KB (ATmega328) |
|  | EEPROM | 1 KB (ATmega328) |
|  | Clock Speed | 16 MHz |
|  | Need of External Programmer | NO |
|  | Built-in LED | pin13 |
|  | Length | 686 mm |
|  | Width | 53.4 mm |
|  | weight | 25 g |

Table 1.1: Features of Arduino UNO

Cost Effective - The Arduino UNO, its basic variant that every beginner to embedded systems usually starts with, is available for less than $25. Being an open source hardware and due to its simplicity, it is easily and widely replicable, which is also responsible for the growth of such a large community as mentioned in the earlier reason.

Features - It is generally observed that the computing power and inbuilt memory of the Arduino easily supports a wide variety of circuits with multiple components along with continuous serial communication with another device/computer. Compared to the system configuration of the board, our project would require fairly low amount of resources. The no. of GPIO pins on the UNO is more than sufficient considering the need for the circuitry in our project, as we shall see when we look at the pinouts of the servo motor and the sensor - the only two other devices being used in the project.

One important advantage of using the UNO is that unlike most previous programmable circuit boards, the Arduino UNO does not need a separate piece of hardware (called programmer) in order to load new code onto the board — we can simply use a USB cable. The ATmega328 on the Arduino/Genuino Uno comes preprogrammed with a bootloader that allows us to upload new code to it without the use of an external hardware programmer. Moreover, the components we have used work under the same operating voltage and current conditions as an arduino, especially considering the fact that such components are nowadays sold with their compatibility with arduino in mind.

Considering the above factors, we found Arduino UNO to be the most appropriate microcontroller board to work with for this project.

HC-SR04  Sensor

## 1.1.2 Operation of Arduino

Once a program written for an Ardllino, called a "Sketch" is uploaded to the board via the Arduino IDE, the Arduino environment performs some small transformations to make sure that the code is correct C/C++. It then gets passed to a compiler (avr-gcc), which turns the human readable code into machine readable instructions (object files). Then, the code gets combined with (linked against) the standard Arduino libraries that provide basic functions like digitalWrite() or Serial.print(). The result is a single Intel her file, which contains the specific bytes that need to be written to the memory of the chip on the Arduino board. This file is then uploaded to the board: transmitted over the USB or serial connection via the bootloader already on the chip or with external programming hardware.

1.2 The HC-SR04 Ultrasonic Sensor



Figure 1.2: HC-SR04 Ultrasonic sensor

The HC-SR04 ultrasonic sensor uses sonar to determine distance to an object like bats or dolphins do. It offers non-contact range detection from 2cm to 400 cm or 1" to 13 feet It's operation is not affected by sunlight or black material like sharp rangefinders are (although acoustically soft materials like cloth can be difficult to detect). It comes complete with ultrasonic transmitter and receiver module. The human ear can only detect sounds betwen 20Hz-20kHz. The sound waves beyond 20kHz are called ultrasonic waves or ultrasound.

The prmciple of ultrasonic distance measurement uses the velocity of ultrasonic waves spreading in air, measuring the time from launch to reflection when it encounters an Obstacle. We then calculate the distance between the transmitter and the obstacle according to the time and the velocity Thus, the principle of ultrasonic distance measurement is the same as with a radar.

## 1.2.1 Features

The reason to choose this sensor was due to its reasonable range of taking measurements of the presence of an object for very less cost and electrical/computational resources. It's

1. Introduction

|  |  |  |
| --- | --- | --- |
|  | Electrical Parameters | HC-SR04 Ultrasonic Module |
|  | Operating Voltage |  |
|  | Quiescent Current (inactivity) | 2mA(1.5-X5mA) |
|  | VVOrking Current | 15mA(10-20mA) |
|  | Operating Frequency | 40KHZ(u1trasonic) |
|  | Farthest Range |  |
|  | Nearest Range |  |
|  | Resolution | 0.3cm |
|  | Measuring Angle | 15 Degrees |
|  | Input Trigger Signal | Ious TTL pulse |
|  | Output Echo Signal | Output IT L level signal, proportional with range |
|  | Dimensions |  |

Table 1.2: Features Of HC-SR04

features are given in Table 1.2.

### Note - Use of only stationary objects

Since the sensor considers the distance to an obstacle that reflects the ultrasonic wave, cannot use it to make trustworthy measurements of the position of moving objects. This is majorly due to the fact that this sensor works on the principle of transmitting a set of pulses and recieving their reflection, before transmitting again, as we shall see later. Hence, there is no way that we can continuously track the position of an object with such a sensor As far as we know, that can only be done by some type of a camera, which would've introduced the concepts of computer vision and taken us beyond the scope and costs of the current project.

#### 1.2.2 HC-SR04 : Pin Definitions

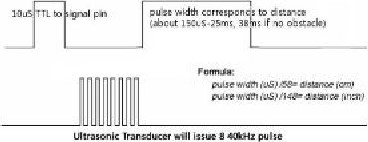
* Vcc - 5V Power Supply
* Trig - Trigger Pin (Input)
* Echo - Receive Pin (Output)  GND - Power Ground

The frig and Echo pins are used for tranmission and reception of ultrasonic waves, respectively

HC-SR04  Sensor

#### 1.2.3 Operation of the HC-SR04

Initiate Echo back



Signal

internal

Figure 13'. Timing Diagram Of HC-SR04 Ultrasonic sensor

The timing diagram of HC-SR04 is shown in Figure 1.3. To start measurement; Trig of SR04 must receive a HIGH pulse (5V) for at least 1 Ous, which will initiate the sensor to transmit S cycles of ultrasonic burst at 40kHz. It then waits for the reflected ultrasonic wave. When the sensor detects the ultrasonic wave through the receiver, it will set the Echo pin to HIGH (5V), for as long as it detects an incoming ultrasonic pulse at the receiver.

Calculation of distance

The delay between tranmission and reception of the final pulse (i.e., between when the Trig pm is set back to LOW and when the Echo pin turns HIGH) is a time period proportional to the distance travelled by it. TO obtain the distance, we start a counter the moment the Trig pin is set to LOW which keeps counting till the microcontroller detects a HIGH pulse at the Echo pin. Time = Width of pulse, in us (micro second).

* Distance in centimeters Time / 58
* Distance in inches = Time / 148

as mentioned in its datasheet. We can also utilize the speed of sound. which is 340m/ s in a generic case.

This entire process can be translated into the Arduino code as shown in exarnple 1.1.

1. Introduction

Example 1.1 (Arduino Code for calcuating distance from HC-SR04 sensor)

Function for calculating the distance measured by the U 1 trasonic s ensor\*/ float calcul ateDi stance unsigned long T 1 digital Write (trigPin , LOW) ; // trigPin needs a fresh LOW pulse before sending a HIGH pulse that can be detected echoPin del ayMi crosec onds • // DELAY #2 : time for which lov trig pulse is maintained bef ore making it high digital Write (trigPin , HIGH) ; del ayMicroseconds (10) ; //DELAY Sets the trigPin on

HIGH State for 10 micro scc onds digital Write (trigPin , LOW) duration pulse1n (ech0Pin , HIGH) / / Reads the echoPin ,

:returns the s ound vave travel time in microseconds di st ance (duration /2) /29. 1 ; // in cm . Calibrat ed

Datasheet shows "duration the formula return distance ;

### Need of calibration of the sensor

As we can notice in the given code, we had to change the formula for calculation of distance, as the values were not close to real values of distance of objects kept in front of it, as tested by us. We would discuss it in a later stage.

## 1.3 Micro-Servo Motor



Figure 1.4: Micro-Servo Motor

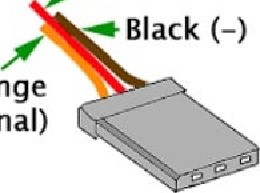
Micro-Serao Motor

We had to use a servo motor in order to rotate the HC-SR04 and give it a wider field of view over the area around it. This Micro-Serao can rotate approximately 180 degrees(90 in each side) with an operating speed of 0.1s/60degrees. Moreover, it has an operating voltage of 4.8V( 5V) which is in the same range as of the Arduino LINO. Other details:

* Weight: 9 g
* Dimension: 222 x 11.8 x 31 mm approx.
* Stall torque: 1.8 kgf.cm
* Operating speed: 0.1 s/60 degrees
* Operating voltage: V ( 5 V)
* Deadband width: 10 us (The servo will not move so long as subsequent commands to it are within the deadband timing pulse width)
* Temperature range: 0-55C

The Micro-Servo has a signal pin to which the arduino sends the value of angle that it should turn to. Figure 1.5 shows the pins of the Micro-Serva.

"Red (+)

Orange

(Signal)

Figure 1.5: Pins of Micro-Servo

The Ultrasonic Sensor was mounted on top of it and the apparatus connected to the arduino was kept in a modified transparent plastic case, as shown in Figure 1.6.

1. Introduction



Figure 1.6: The Apparatus

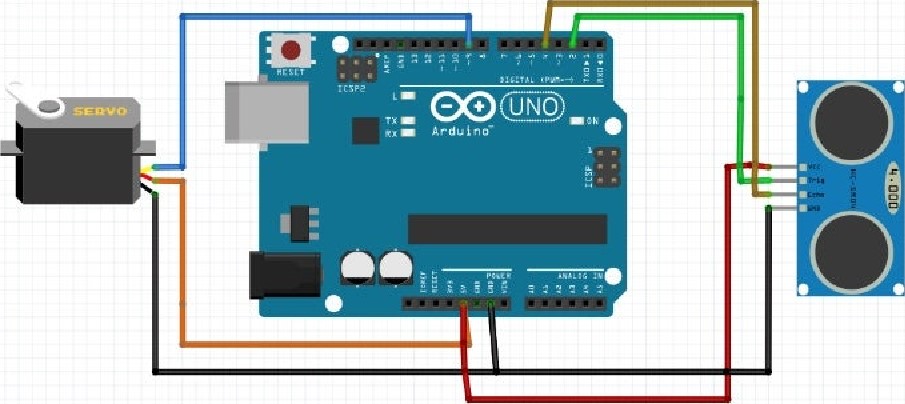
We then proceeded to deciding the circuit connections, as shown in the next section.

2 Circuit Schematics

## 2.1 Arduino Pin Connections

|  |  |  |
| --- | --- | --- |
|  | Pin of Arduino | Connected to |
|  | Digital 1/0 (2) | Trig of HC-SR04 |
|  | Digital 1/0 (4) | Echo of HC-SR04 |
|  | Dlgital 1/0 (9) | Signal pin of Servo(orange) |
|  | Vcc | vcc of HC-SR04 and Servo(Red) |
|  | GND Pin | GND of HC-SR04 and Servo(B1ack) |

### 2.2 Breadboard View



fritzing

Figure 2.1; Circuit : Breadboard View

11

Chapter 2. Circuit Schematics

2.3 Schematic view

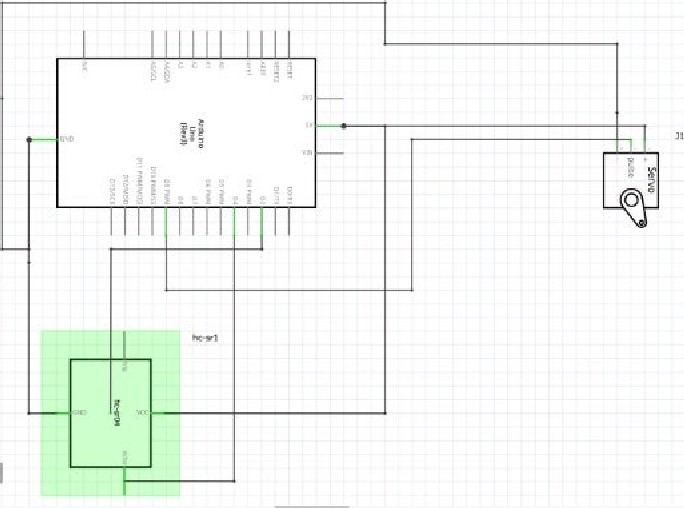
fritzing

Figure 2.2'. Circuit : Schematic view

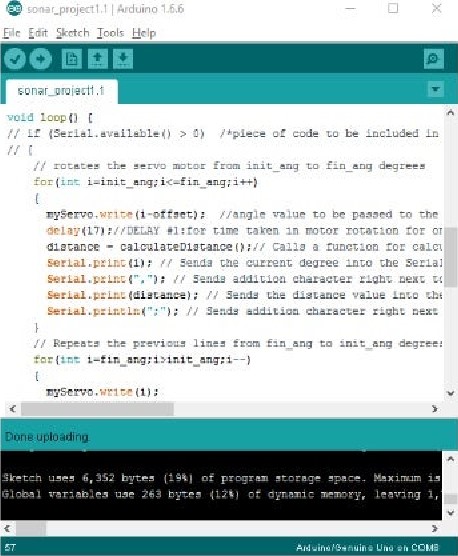
Above circuit schematics were obtained using the Fritzing software.

Once we decided the connections, we had to n.vo more steps left before connecting the actual circuit: setting up the Development Environment and Circuit Simulation.

3 Development and Simulation

## 3.1 Arduino IDE

The code to be burnt to the program memory of an Arduino is written and compiled in the Arduino IDE. The IDE also has a serial monitor, which displays values being recieved in real-time from the Arduino via serial communication through the serial port .COM ports). The IDE verifies for correct C/C++ syntax and compiles it, before linking it to Arduino Library files. This creates the hex file that contains the code in binary form. It can either be uploaded to the board directly, which the IDE does with the help of the Bootloader program pre-installed on the Arduino, or it can be fed to a simulator, which would simulate and show how the circuit would run and its associated parameters.



Figure

3

1:

The

Arduino

IDE

13

Chapter 3. Development and Simulation

## 3.2 Simulation with Proteus

It is a good practice to simulate a circuit before connecting it, in order to avoid hardware damage/debug issues in a circuit beforehand. We used Proteus Professional 8 for this purpose. Proteus has libraries for most commonly used electronics components, and it also allows us to make our own components, thanks to which we could download unavailable components from the internet, where many users have shared their own custom components.

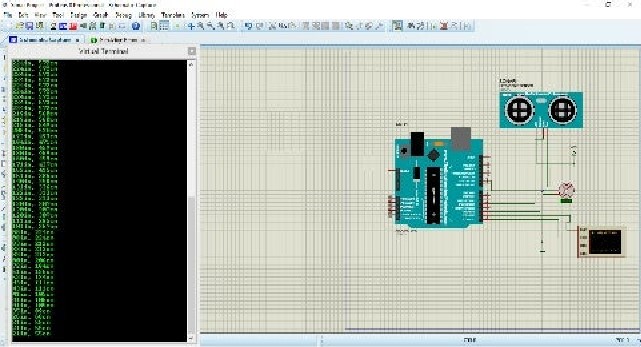


Figure 3.2: The Proteus Simulator

### 3.3 Processing

Processing is an interactive software to write programs with visual output. We have used Processing 3.2 for generating a 2D radar-map representation of the serial data being obtained from the Arduino through the computer's serial port.

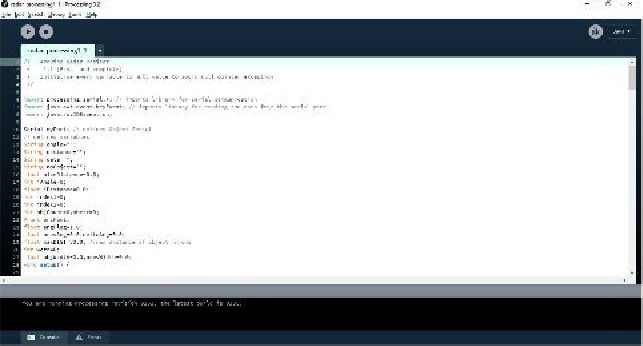


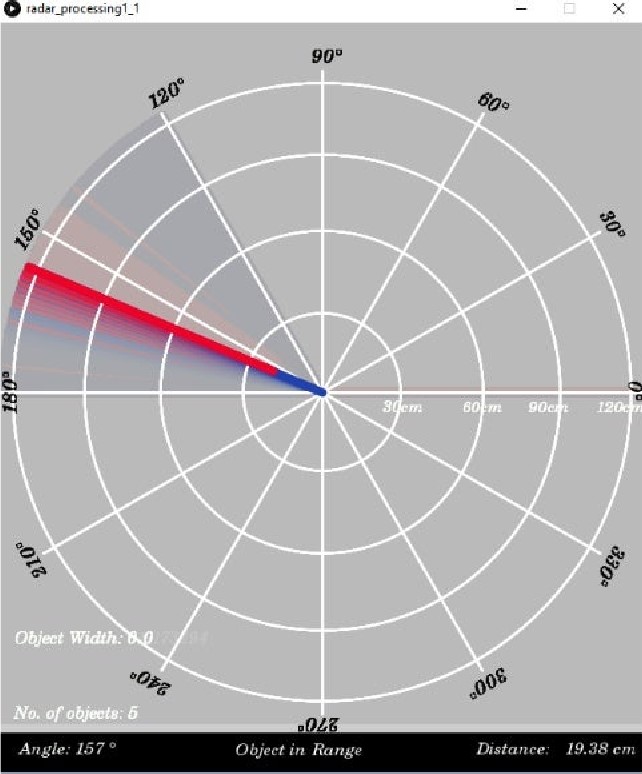
Figure 3.3: Processing

The code in Processing is written in and primarily consists of drawing the lines

3.3. Processing 15

and borders of the radar map, and plotting the tracker-bar on the map. We added functionalities such as counting the number of objects, and also tried to calculate the width of the object, apart from just showing the distance of the object.

We made a 360 degrees map so that we could view the area according to our convenience if the angle of the set-up is changed. This angle is the "offset" that we introduced in the Arduino code. The entire Arduino and Processing codes are shown in Appendix C and Appendix D, respectively.



Figure

3.4;

Running

Our

code

in

Processing

4 Explaining the Codes

Here we'd be briefly going through the flow of the codes written for Arduino and in Processing. The entire Arduino and Processing codes are given (explained with comments) in Appendix C and Appendix D, respectively

## 4.1 The Arduino Code

Every Arduihiü code consists of two standard functions - OOid setup() and void loop(). The former is used to define and initialise variables, pins and the baud rate of serial communication.

We have programmed the Arduino to control the circuit in the following flow:

1. Rotate the motor to an angle,
2. Use the ultrasonic sensor and calculate the distance,
3. Send both the above values via serial communication,
4. Rotate the motor to the next degree,
5. Repeat steps 2-5.

The following code snippets implement the above given steps:

Listing 4.1: Step 1

// rotates the servo motor from init \_ ang to fin \_ ang degrees for (int i = int \_ ang ; i < -i in \_ ang ; i++)

My Servo . write ( i -offset // angle value to be passed to the servo library object for writing into the motor del ay (1 7) ,' // DELAY #1 : for time taken in motor rotation for one degree before calculating distance

Listing 4.2'. Step 2

di stance calculate Distance o ; // Calls a function for

16

4.1. The Arduino Code 17

calculating the di stance measured by the sensor for each degree

Listing 4.3; Function called above

// Function for calculating the distance measured by the

Ultras on i c sensor float calculatel)istance igned I Ong TI micros digitalWrite (trigPin , LOW) , // trigPin needs a fresh LOW pulse before sending a HIGH pulse that can be detected from echo Pin del ayMi crose conds • // DELAY time for which lov trig pulse is ained before making it high digitalWrite (trigPin , HIGH) del croseconds (10)  sets the trigPin on HIGH state for 10 micro seconds digitalWrite (trigPin , LOW) dur at ion  (echoPin , HIGH) Reads the e choPin , returns the sound wave trav el time in microseconds / / distance = duration 034/2 ; distance (duration /2) / 29 . 1 ; // in cm , datasheet gives dur ation,/58 V" as the formula

Listing 4.4: Step 3

Serial . print (i) // Sends the curr ent degree into the

Serial Port for gr aphical representation

Serial . print // Sends addition char acter right next to the previous value needed later in the Processing IDE for indexing

Serial . print (distance // S ends the dist ance value into the Serial Port for the graph

. print / / Sends addit ion char right next to the previous value needed later in the Processing IDE for indexing

Keeping the above code in a void loop ( ) ensures that the steps get repeated in a loop, as desired.

18 Chapter 4. Explaining the Codes

### 4.2 The Processing Code

The JAVA code written for processing divides its work into a few main functions, which are customary for every code written in Processing. The functions are

* void getup ( )
* to initialise variables and baud rate of serial communication,
* void dratü()
* to make a 2D/3D drawing on the screen on which to show the output,
* void serialEvent ( )
* the function dealing with incoming "outgoing serial data, and operations on it.

Customised functions were used to draw several parts of the displayed graph, which vvere called from void draw(). These are

* void dratéRadar()
* to draw the basic circular layer of graphics
* void drauLine()  to draw a line with trails which would be used to sweep over the area implying scannmg
* void draüObject()
* to draw a colored line over the scanning figure to indicate presence of an object, whose length would be dependant on the distance of the object from the sensor
* void draaText ( )
* to place text at different places on the map to show the serial and calculated data values

5 Observations

We observed data from both the serial monitor of the Arduino IDE, as well as from the graph in processing. Since we wanted our device to give the best results as quickly as possible. we decided to calibrate it'





Figure 5.1: Data Observed in the Serial Monitor of the Arduino IDE

## 5.1 Calibration

### 5.1.1 Calibration of Measurements

We realised that actual measurements of distance/count/width of an object placed in the vicinity of the apparatus were different from the ones provided by our device. Hence, we proceeded to calibrate each part of our device that was capable of taking a measurement.

The quantities that were measured in our project before going for any further calculations were: Angle and Distance. The rest were only obtained by playing around with these two main quantities. Hence, we needed them to be as accurate as possible.

We did not notice any error in the value of angle that the motor was rotating to, hence did not have to do any sort of calibration for it.

To calibrate the distance, we placed several objects at fixed distances from the sensor on a white chart paper, with values of distances marked on it with the help of a ruler. We then made measurements keeping the sensor still, and made an adjustment to the formula in the code by observing the multiplication factor that needed to be introduced into the previous formula in order to get the correct values. Hence, the formula was changed from "duration/58V to "duration/ 58.2".

19

20 5.



Figure 5.2: Calibration on chart paper with marked distances

### 5.1.2 Optimising Speed

We wanted to optimise our device to work as quickly as possible, without affecting normal execution of the program. So we optimised our code, considering the delays in physical movement, time periods of recieved/transrnitted waves and time taken to process the data.

#### Graphs

We took into account the several delays introduced in the code, along with the delay in rotation of the motor(specified by its speed given in its datasheet, tested to be accurate) and delay in calculation of distance. Accordingly, we minimised each of the values given as arguments to the

delay C)

function used at different parts of the Arduino code, to the safest least value possible. Each of the delays with descriptions can be observed in the Arduino code given with comments in Appendix C

## 5.2 Graphs

We obtained the following [error vs. position] graphs for three different kinds of objects kept in front of the sensor. In each case, we noted down values for both the sensor being stationary, and rotating. The three different kinds of objects were

Objectl Hollow plastic bottle (width—6cm) Object2 Solid DC battery (width—3cm) Object3 = Metallic pen

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1.8  1.6 g  1.4 0  1  0.8  0.6  0.4  0.2 | 5 | 10 | avg. lerrorl vs position    15 20 25 | stationary  delay(17) • fastest  - Slow          position  30 35 (cm) |

Figure 5.3: Objectl-avg. I error vs position

5.

|  |  |  |  |
| --- | --- | --- | --- |
| 06  0.2 | 10 | avg. I errorl vs position  15 20 | stationary  delay(17 ) - fastest  delay(30) - slow  position  35 (cm) |

Figure 5.4: Object2-avg. I error I vs position

|  |  |  |  |
| --- | --- | --- | --- |
| 08  0.6 | 10 | avg. I errorl vs position  15 20 25 | delay(17) - fastest  delay(30) • slow  position  35 (cm) |

Figure 5.5: Object3-avg. I error [ vs position

Finally, we also plotted the error in data of width of object vs. position, which was

### Graphs

something like this:

|  |  |  |  |
| --- | --- | --- | --- |
| (obiect at 20cm)  3  2 | out of ra nee  avg, lerrorl vs | Object  2 | out ot range  stationary  delay(17) - fastest    delay(30) - Slow  object  3 |

Figure 5.6: Object-wise avg. I error I in width at

24 5.

### 5.3 Challenges

As observed from the graphs plotted in the previous section, we are not able to determine any particular increasing/decreasing order being followed by the errors in measurement of the "distancerr quantity Moreover, by increasing the speed of measurement of the device (by reducing the manual delays introduced inside the Arduino code within safe limits), the error does not necessarily seem to increase with speed. The "width of the object" quantity seems to be correct at one instant and completely wrong at any other instant with very random error values.

However, what we did observe were sudden jumps in errors. Among several trials, their occurrence increased with the speed. That is, the device was detecting stray values, and the amount of stray values per experiment increased with the speed of the device.

#### 5.4 Identified Problem

With the help of ideas from several online electronics forums, we carefully studied the flow of the code, which exposed the flaw in our logic. The problem lied in the fact that the code took variable amount of time in calculating the distance of an object, proportional to the distance itself. This fact could be derived from the line in the code

duration = pulseIn(echoPin, HIGH) ;

where the function pulseli-u starts a counter which counts till the value at echoPin is HIGH. Here, our program only waited to recieve the first returning pulse, before quickly sending the output and turning the motor to the next angle. At some instances, this could mean that by the time the sensor was ready to take a new measurement; the older echo pulses vvere still arriving at the receiver, hence a wrong measurement. The older echo pulses could still be around due to

(d) The program having quickly moved to the next set of commands and having made the sensor ready to receive ultrasonic waves - this, within the time in which the remaining of the 8 pulses were still being recieved at the reciever, or

(b) Due to the waves being spherical in nature and the field of view of the sensor being 15 degrees, the waves could've been reflected by some undesired obstacles in the field of view. It could also be the result of multiple reflections.

#### 5.5 Proposed Solution

Therefore, we edited the algorithm in such a way, that between each degree of rotation of the motor, the program takes the exact amount of time to run. That is, the program makes up for the time left out compared to the extreme-time-taking case by waiting.

We did this by using the fact to our advantage that the maximum titne taken by the calculateDistance() function is when there is no obstacle. The FIC-SR04 sensor waits for 38ms in case no object is detected, before giving a HIGH echo signal. Hence, we ensure that the program always takes a total of 38ms for each iteration of the loop. For this, we used the

micros() function inside the calculateDistnnce() function of the arduino code, which calculates the number of microseconds elapsed till the code execution started. Here's the added portion of code which did our solution:

Listing 5.1: Corrected calculate Distance() function

// Funct :for  the distance measured by the

Ultras on i c sensor float calculateDistance igned I Ong TI micros digitalWrite (trigPin , LOW) // trigPin needs a fresh LOW pulse before sending a HIGH pulse that can be detected from echoPin del ayMi crose conds ' //DELAY time for which lov trig pulse is maint ained before making it high digitalWrite (trigPin , HIGH) del conds CIO)  Sets the trigPin on HIGH state for 10 micro seconds digitalWrite (trigPin , LOW) dur at ion pulseIn (echoPin , HIGH // Reads the e choPin , r eturns the sound wave travel time in microseconds //distance= duration 034/2 ; distance - (duration /2) /29 . 1 ; // in cm , datasheet gives dur ation,/58 " as the formula

// To avoid sending data at variable time intervals due to varying time duration taken between execution of above code inside this function depending on distance of

##### obstacle

// if no obj ect , echo pulse is HI CH after 38ms while (micros Tl <38000)

return distance ;

Here are the graphs replotted for the data obtained after correction

26 5.

|  |  |  |  |
| --- | --- | --- | --- |
| 06  0.2 | 10 | avg. I errorl vs position  15 20 | delay(17) - fastest  delay(30) • slow    position  35 (cm) |
| 08  06  0.4 | 10 | After Correction avg, I errorl vs position  15 20 25 | delay( 17) - fastest  delay(30) - slow    position  35 (cm) |

5.7: I vs, after correction

5.5.

|  |  |  |  |
| --- | --- | --- | --- |
| 1.8  1.2  0.6  0.4  0.2 | 10 | avg. lerrorl vs position      15 25 | stationary  delay(17) - fastest  delay(30) - slow  position  35 (cm) |
| 1.6  1.2  0.8  0.4 | 10 | After Correction avg, I errorl vs position      15 25 | delay(17) - fastest  delay(30) - slow    position  35 (cm) |

5.8: I vs. after correction

5.

|  |  |  |  |
| --- | --- | --- | --- |
| 06  0.2 | 10 | avg. I errorl vs position  15 20 | stationary  delay(17 ) - fastest  delay(30) - slow  position  35 (cm) |
| 1.6  08  06  0.4 |  | After Correction avg. I errorl vs position  15 20 25 | delay(17) - fastest  delay(30) • slow    position  (cm) |

5.9: I vs, after correction

5.5.

|  |  |  |
| --- | --- | --- |
| (obiect at 20cm)  8  2 | out of range  avg, lerrorl vs Object    2 | out of range  stationary  delay(17) - fastest  delay(30) - slow  object  3 |
| 8  6  4  2 | After Correction avg. lerrorl vs object    2 | |  |  | | --- | --- | |  | 0b ect at 20cm |   delay(17) - fastest    delay(30) • slow  object  3 |

Figure 5.10; Object-wise:avg. I error I in width at x-=20C1ü. before vs, after correction

6 Conclusion

We continue to observe that there are sewral challenges, some unexplainable, in obtaining a stable and accurate reading using the HC-SR04 ultrasonic sensor. After we curbed some of the error, we tried to understand what more could've possibly gone wrong with the algorithm/circuitry- This lead us to raise questions at the depth of its internal working, which was a dead end, as we did not have any understanding of the complicated circuitry in it.

We conclude by stating that the HC-SR04 Ultrasonic Sensor is a good device for detecting objects, but measurements taken from it can at best be suggestive, but not conclusive of the data about present objects/obstacle.

We believe that for best results, especially in a robust application, this device must be used for detecting vrpresence" of objects and so far as to get a fair idea, the data that it provides. But, to measure exact distances of objects lying in front of a sensor. especially if in motion, we must use a different type of sensor, possibly a camera. and even a depth sensor like devices running on Augmented-Reality incorporate.

A Acknowledgements

We are deeply grateful to our teacher, mentor and team leader; Dr. Deepak Nair, who held us through the course of the semester and made us learn things in a very systematic way. This in itself was solely responsible for teaching us with deep practice, new skills and methods which could have been easily missed out; otherwise.

Thanks to sir's guidance, not a single portion of the project, however rudimentary it could've sounded, was devoid of new knowledge and appropriate professional methodology

We'd like to thank our institute, the LNMIIT Jaipur, for having granted us the resources,freedom and opportunity to embark on a project of our choice, which has taught us so much.

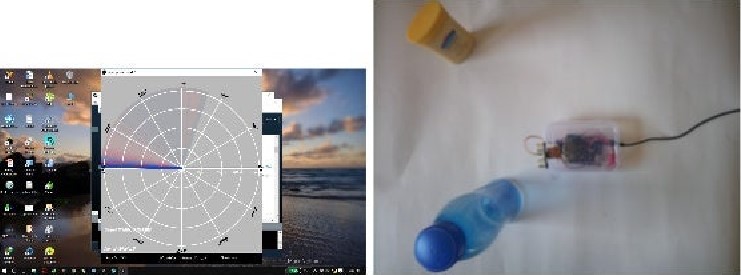


Figure B.2; The project

# C Appendix A : Arduino Code

Here is the code uploaded on to the Arduino IINO:

Listing C. 1; The Arduino Code

1 . 1 (Final )

// Includes the Servo 1 ibrary

\*include . h) 

//  Trig and Echo pins the Ultrasonic Sensor const int trigPin - 2 ; const int echoPin =

|  |  |
| --- | --- |
| // Variables for the duration & distance measured from the float duration ; float distance ;  // Angle offset between aph and motor XY axe s int offset  // Initial and final angles of motor i s rotation  // Note that the motor can only take values 0-180 .  // Make sure init\_angle - offset to avoid jumper    obstruction .  int init \_ ang 70+ offset ; int fin \_ ang - IBO\* offset ;  // Decl ares a structure vari able / object of the type S ervo , | sensor |
| // (defined in the servo ary) for controlling the servo  Servo my Servo ; void setup | motor . |

pinM0de (trigPin , OUTPUT) // Sets the tr igPin as an Output

(echoPin , // Sets the echoPin an Input Serial begin // Sets baud rate of serial communication my Servo . attach D on which pin is the SEEVO motor

### attached

34 Appendix C. Appendix A : Arduino Code

void loop

// if ( Seri al . avail able () 0) /•piece of code to be included in case of serial communi cation vith 10T board\* /

// rotates the servo motor from init \_ang to fin\_ang degrees for (int i = init \_ <

my Servo . write (i offset// angle value to be passed to the servo

Library object for it ing into the motor de 1 ay (17) • , // DELAY for time taken in motor rotation for one degree before at ing dist di stance cal culat eDist ance • // Calls a function for calculating the dist ance measured by the Ultr asonic sensor for each

degr e e

Serial print (i) // Sends the current degree into the Ser i al P ort for aphical represent ation

Serial print // S ends addition character r ight next to the previ ous value ne eded later in the Processing IDE for indexing

Seri al . print (distance) // Sends the distance value intO the

Serial Port for the graph

Serial . print  // S ends addition character ight next to the previ ous value ne eded later in the Pro cessing IDE for indexing

// Repeats the previous lines fin \_ ang to init\_ ang degrees for (int i -fin \_

my Servo . write (i de 1 ay // DELAY // can be minimised to 17 1667 ml crosec cui at eDist an C e

Serial print (i)

Seri al pr int

|  |  |
| --- | --- |
| Serial | print (distanc e |
| Serial | pr int |



// Function for cal culat ing the di st ance measured by the

Ultrasonic sensor float calculateDistance unsigned long T 1 micros dig i t al Write igPin , LOW) // trigPin needs fresh LOW pulse before s ending a HIGH pulse that c an be detected from echoPin del ayMicros econds ' // DELAY time for which lov trig pulse is ained before making it high digital Write (tr igPin , HIGH)

(1 0) ; // DELAY sets the on HIGH state for 10 mi cro seconds digital Write (trigPin , L 01/) 

duration  ( echoPin , HIGH the sound wave travel time in / / distance: dur at ion . 034/2 ; distance dur at ion/ 2) /29 . 1 ; durati on/ 58 v' as the formula

// Reads the e choPin , micr oseconds

#### // in cm, datasheet gives

// To av oid sending data at variab le time intervals due to vary ing time duration taken between execution of above code inside this function depending on distance of obstacle // if no object echo pulse is HIGH after 38ms while (micros



ret urn dist

# D Appendix B : Processing Code

The Java code that we ran in Processing is given below :

Listing D. 1: The Processing Code

 Arduino Radar Project

 1 . 1 (Final and completeJ

 Initial ise every variable to null value to avoid null pointer exception

import processing serial imports library for serial c ommunic at i on import java. . event KeyEvent // imports library for reading the data from the serial port import java. i o . I OException ;

Serial my Port ; // defines Obj ect Serial

// defines variables

### String angle ,

String distance = i' i' ;

String data— String float pixsDistance =O. O ' int iAng1e float i Distance -O . O ;

int indexl -0 ; int index2 int 0b j Count

PF ont orc Font •

-1 . 0 , fl oat prevAng -0 . 0 , de 1 aA ng float maxDIST=50. O ; //max distance of object In cms int wctr fl oat objWidth- 0 . 0 , prevWidth void setup

#### 36

size (600 , // CHANGE THIS TO YOUR SCREEN

smo 0th 

myP ore - nev Serial (this , "/dev/ttyACMO" 9600) // Enter the COM

Port address as COI•14 or COM 22 . starts the serial communi cation

myP ort . buff ertjnt il  // reads the data from the s erial port up to the character  so  it reads this : angle , distance .

orc Font loadFont ( l' Century SchL —Ital —20 .

void dr

#### (237 , 13 , 245) ; textFont ( or CF ont

// simulating motion blur and slow fade of the mov ng line noStroke fill (184 , 13) ; rect (0 , O , width , he ight -height . 06S)

 co Ior

// calls the functions for drawing the r adar dravRadar drawLine () dr av0bject o ; draw Text

void serial Event (Seri a I myP ort) { // starts reading data from the Serial P ort

// reads the data from the Serial Port up to the char and puts it into the String variable Ii dat a VI try {

data — myPort  ; 3 data . sub string (O data. length

i nd ex 1data inderOf // find the character and puts it into the variable indexl data . sub (0 , rid ex I ) // read the data position

0 v' to position of the var iable indexl or thats the value of the angle the Arduino Board sent into the Serial P ort distance- data . substring indexl+l , data . length / / re ad the data from position "indexl to the end of the data pr that s the value of the distance

// conv erts the String variables into Integer i Angle int (angle i Di stance float (distance) 

de  -prevAng

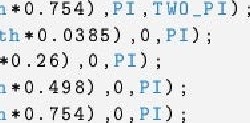
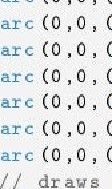
if (del taAng\* angF lag obi CountobjWidth

// antic lockvi se --de ItaAng>0 // clockwise --de ItaAng<0 if (deltaAng angF 1 ag

angF 1 —

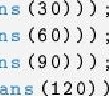
catch (Eyception e) { pr  par sing ; e . printStackTrace

void drawRadar () pushMatrix tr ansla•te (vidth/2 , height -he ight 507) • mov eg the art ing coordinates to new location noFi11 strokeWeight stroke (407 , 409 , 305) , dravs the arc lines

(0 , 0  (width-vidth 0385) , (vidth-vidth TWO\_PI) arc (0 , o (vidth Vidth . 2 6) (vidth-vidth \*0 . 26) PI , TWO\_PI ar c (0 , 0 vid th Vid th\*0 .498) , (width - vi dth 498) , PI , TWO\_PI ) ,

(width-uidth 754) , (width-vidth vid th vidth\*0 . 0385) , vidth -vidth (width-vidth 26) , (vidth-width vid th Vidth\*0 .498) , (width-vidth

(vidth vid th 754) , (Vi dth-vidth the angLe in e vidth vi dth

1 in e (0 , 0 COS radi (30) )- 'Aidth sin (radi ans line (0 , 0 width cos radi ans (60) ) , —Vidth /2) \*sin (radi ans line (0 , 0 width cos radi ansvidthsin (radi ans

1 in e (0 , 0 radi (120) ) , -Vidthsin radi

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 in e (0 , 0 |  |  | COS | radi | (150) ) | -Vidth |  | radi | (150) ) |

cos radi ans (180) )vid thsłn radi ans (180) ) line cos radi ans (210) ) ,vid thsłn radi ans (210) )

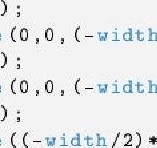


line

#### line (0 , 0 cos radi ans (240) ) , ( -ťidth/2) \* sin radi ans (240) )

line ( 0 , 0 -width COS radi( 270) ) -width sin( 270) )

line cos radi ans ( 300) ) , vidth sin radians ( 300) )



vidth

Width

line cos radi ans ( 330) ) , width' 2) \*sin r ad i ans ( 330) )

line cos ( radi ans, vidth /2 , popMatrix o ;

void pushMatrir s t rok.eWeight

str Oke ( 32  // color for blue line translate (width /2 , he ight -hei ght \*0 . 507 ) , moves the start ing coordinates to new IOC line height -heightcos (r adians ( i Angle n ,  ( he i ght he ight S in (radi  draws the according to the angle popMatrix

void draw0bj ect pushMatri% translate (width /2 , he ight -hei ght \*0 . 508 ) , moves start ing coordinats to new location strokeWeight str Oke ( 240 , 40) // red pixsDistance i Dist ance height he ight \*0 . 16 66) \*0 . 013 /3) ; covers the di stance from the s ensor from cm to pixels if ( i Distance < -max DIST if (vctr > 2) { // As suming that an object will be thick enough to be detected for 2 degr e es Of  . objWidth-

// draws the object according to the angle and the di stance line (p ixsDi stance s ad i ans ( i Anglen , -pixsDistance\*sin (radians iAng1e ) ) , ( width-vidth \*Ci . SOS ) \*cos (r adians ( iA ngle n , width width obctr

popMatrix

void dr awT ext  // draws the texts on the screen pushMatrix if ( i Distance >maxDIST) { noObject [i NO Object tÅithin if (vctr--O) objWidth-pr evWidth

else objWidth-f10at (vctr )ance \* 0 . 0174 ; / / vi dth of obj ect cm pr evWidth- objWidth ; vctr--o ; if (obctr 0b j Count++ ; 0b ctr

else { wctr++ ; if (wctr // Assuming that an object vi ll be thi Ck en o u gh detected for 2 degrees Of rotation .

|  |  |
| --- | --- |
| no Object | Obj ect in Range |

fil l/ / black backgr ound of b {3 t tom t ext no Stroke rect (0 , h e ight -h ei ghe \*0 . 0621 , width h ei gh t



text Size 

t ext30cm , width - width height -he ight \*0 . 4793) , text60cm wi dth vidth \*0 . 281 , he ight -height text90cm , width - width . 177 , he ight -height\* 0 . 479 2) text120 cm , width -vidth\*0 . 07 29 , height -height \*0 . 4792) ; text Size (16 ) , text (n00bject Width -Width\* 0 . 634 , height -height \* 0 . 021 B)

•text  vidth -vi dth h e ight -h ei ght \*0 . 02 32) ; text "Distance : vidth - width\*0 . 26 , he i ght — h e i Eh t\*0 . 0238) text Size ( 16 ) text Of object s : + Obj +width -width\*0 . 986 , he i ght he ight \*0 . 0714) ; text Object Width : + 0b j Width + ' iwidth-width\*0 . 986 , height he ight \*0 . 1714) ; if (iDist ance if (wctr text + iDistance  width width \*0 . 185 , height he ight \*0 . 0237) ;

text Size C 19 ) ,

// color degrees text tr anslate ( (vidth -width\*0 . 5020) \*Width/ c os r ad i an s height he ight  -vi dth/2•sin (r ad i ans rotate ( —radians

#### text ( t' 30 a

resetMatrix translate ( (vidth -width son + vi dth / 2 •cos (r ad i ans  (he ight he ight \*0 . 5139) width sxn radi ans rot ate ( —radians

text 600 resetMatrix translate ( (vidth -width so 7) \*Width/ 2 \*cos (radianshe igh t he ight \*0 . 5149) -width/ 2\* sin (radi rot ate (radians text GO \ degr ee resetMatrix translate (vidth width \*O . 613+vidth/2\* cos (radians (1 20) ) , (height he ight -vidth/2\*Sin ( radians (120) ) ) ; rot ate (radians ( -30) ) ; text (" 1 200 resetMatri%. 

translate (Width -width\* 0 . 8298 ) + vidth / cos (r adians (1 son , ( h e i ght 

height . 4803) -width/2\* sin radi (150) ) ) , rot ate (radians ( -60) ) • text (" 1 600 resetMatrix 

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | translate (vidth width . 47S+vidth/2\* cos (radians (1 80) ) , (height he ight \*0 . 47791) -width/ 2\* sin (radians (IBO) ) ) | | | rot ate (radians     text (" 1 800     resetMatrix   transl at e (vi dth - width         height . 4797) -width/ Sin (radi rot ate (radians ( -120) ) ; text (" 2 100   resetMatrix   transl ate (width width         he ight \*0 . 4867) -width/ 2\* sin (radi rot ate (radians ( -150) ) ; t eit (" 2400     resetMatrix   transl ate (vi dth width | cos (radians (2 10) ) , height  (210) ) ) ,  cos (radians (140) ) , (height  (240) ) ) ,  cos (radians (2 70) ) , height | | he ight \*0 . 50151) -width/ sin (radians (270) ) ) ; rot ate (radians ( -180) ) • text (" 2 700    resetMatrix | | |  |
|  |
|  |
|  |

transl ate (wi dth -width cos radians (3 00) ) , (height he ight \*0 . 51167) -width/ sin (radians (300) ) ) ; rot ate (radians ( -210) ) ;

text resetMatrix translate (width -width \* 0 . 478+vidth/2\* cos (radians (3 30) ) , (height he ight \*0 . 5174) -width/ 2\* sin (radi (330) ) ) , rot ate (radians ( -240) ) • text 

resetMatrix tr anslate (vid•th -width \*O . 623+ width cos (radians ( 36 0) ) he i ght he ight \*0. 52132) width sin (r adians (3 60 ) ) ) ; rotate ( radi ans (-270) ) •



text popMatr ix pr evAng — iAng1e