An integrated haptic-audio human-machine interface for environmental exploration in absence of visual clues

Edoardo Catenaro, *Politecnico di Milano*, Fabio Izzi, *Technische Universiteit Delft*

I. INTRODUCTION

Human-computer interaction is an emerging feature of modern society. Thanks to ongoing advancements in Electronics and Computer Science, mechatronics systems with an incredible powerful computational capacity (such as personal computers, smartphones, tablets) are nowadays available for an increasing fraction of the world population [1]. Even if a large variety of human-machine interfaces have been commercialized over the last years (touch screens, eye-trackers, EEG-based controllers, etc.) graphical user interfaces (GUIs) undoubtedly represent a dominant paradigm. By relying massively on visual perception, the usage of GUIs has set a large gap between users with and without visual impairments; the latter group being strongly penalized in terms of efficiency and accuracy in the usage of such products [1]. Visual impairment is a major health issue: according to an estimation of the World Health Organization, more than 39 million people were blind in the 2010 [2]. Paradoxically, people suffering of blindness or other optical diseases would benefit even more than healthy users from the usage of mobile smart-devices, since the intrinsic capacity of the latter in providing compensatory information about the surrounding environment. Different solutions have been proposed to improve interaction between visually impaired users and electronic applications. One of the most explored protocols considers the design of audio user interfaces (AUIs). Typical successful applications of this approach are text-to-speech softwares or instrumentation for color identification feeding audio-outputs to the users [1], [3]–[5]. An important limitation for these systems, however, is that audio feedback by external speakers might generate uncomfortable social situations, while the usage of earphones is not always convenient and/or ergonomic. An alternative interesting solution is to provide information by means of haptic signals and different research groups have started to explore similar interfaces [1], [3]–[5]. Tactile and proprioceptive receptors covers the totality of the human body, making almost infinite possibilities for creative designs. Eventually, a similar approach allows more reserved human-machine interactions, i.e. silent and perceived only by the user. The drawback of haptic transmission of information is that richness and clearness of the message is noticeably reduced if compared to audio feedbacks. In out project we decide to merge the advantages of AUI and Haptic communication protocols in order to improve the perception of the external environment for visually impaired people. In this regards, a hybrid Audio-Tactile human machine interface was designed. Its working principle was consist in a instrumented gloves, used by the subject in order to explore and reach information of the external environment. In this regards, a simple proof-of-concept experiment is conducted: the flexion of one finger is used to rotate a sonar, while flexing the middle finger makes the computer to convert angle of the sonar and measured distance (sonar to external environment) into a speech-based message. Eventually, a vibration motor provide an instant haptic signal to the user whether the sonar perceived an object at a distance lower than 10 cm (simulating thus a likely impact). Additionally, a GUIs is made in order to verify that the audio data is compatible with those actually sensed measured. The structure of this report is organized as follows: in section II the experimental setup/protocol, hardware and software are presented; in section III the achieved results are described; in section IV discussion and conclusion are offered. Finally, in appendix A and B it is possible to find the full code, used in Arduino and Processing.

II. METHODS

A. Protocol

The scope of the system is to detect the distance between a manually controlled hinge and the external environment and to provide audio-feedback (angular position and measured distance) when required by the user. Furthermore, in case of a perceived imminent "impact", i.e. when a object is detected to a distance less than 10 cm, a small actuator is used to provide a vibration-based alarm to the user. A radar-like GUI is displayed on the screen in order to compare information measured with those converted in audio format.

B. Hardware

The system is composed by two main hardware components: (1) a processing platform capable of perceiving the external environment; (2) an instrumented glove as user-computer interface. The platform is composed by an Arduino UNO, which is the control and information-processing unit, a servo-motor (Micro Motor Servo SG90) and a sonar (Ultrasonic Sensor HC-SR04) attached to the latter. The glove is equipped with two flexo-sensors (Spectra Symbol 002 17 2), which are individually connected to the fore-finger and middle finger, and a vibrator motor (HTC One M8) which provides haptic feedback - vibrations - at the wrist level. A detail depiction of sensors-actuator-controller connections (breadboard) is described in fig. 1. Fig. 2 and fig. 3 show the physical hardware implementation. Notice that the ergonomics of the instrumented glove was improved by adding layer of soft material and cardboard.

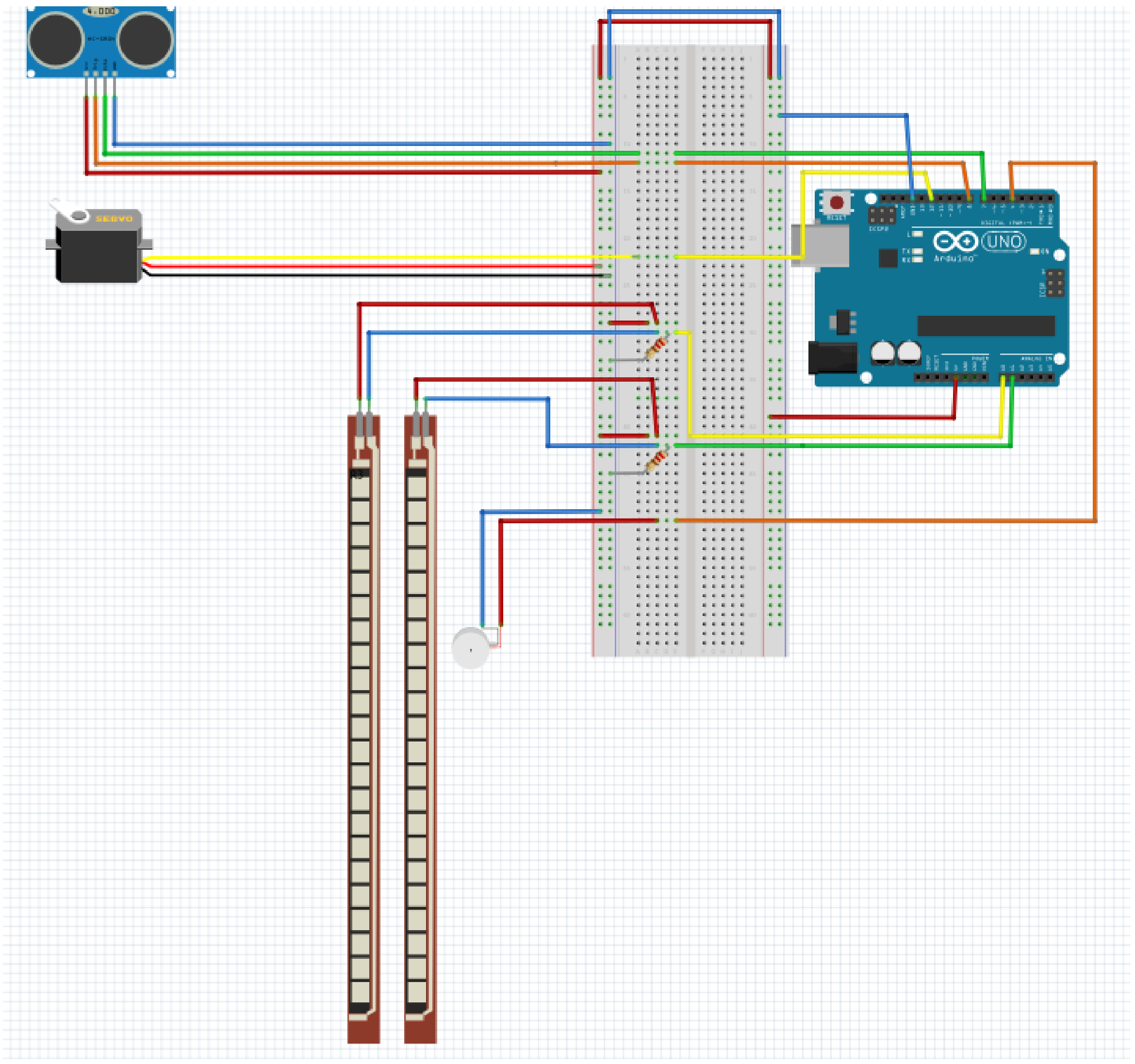

Fig. 1. Used hardware Fritzing sheme

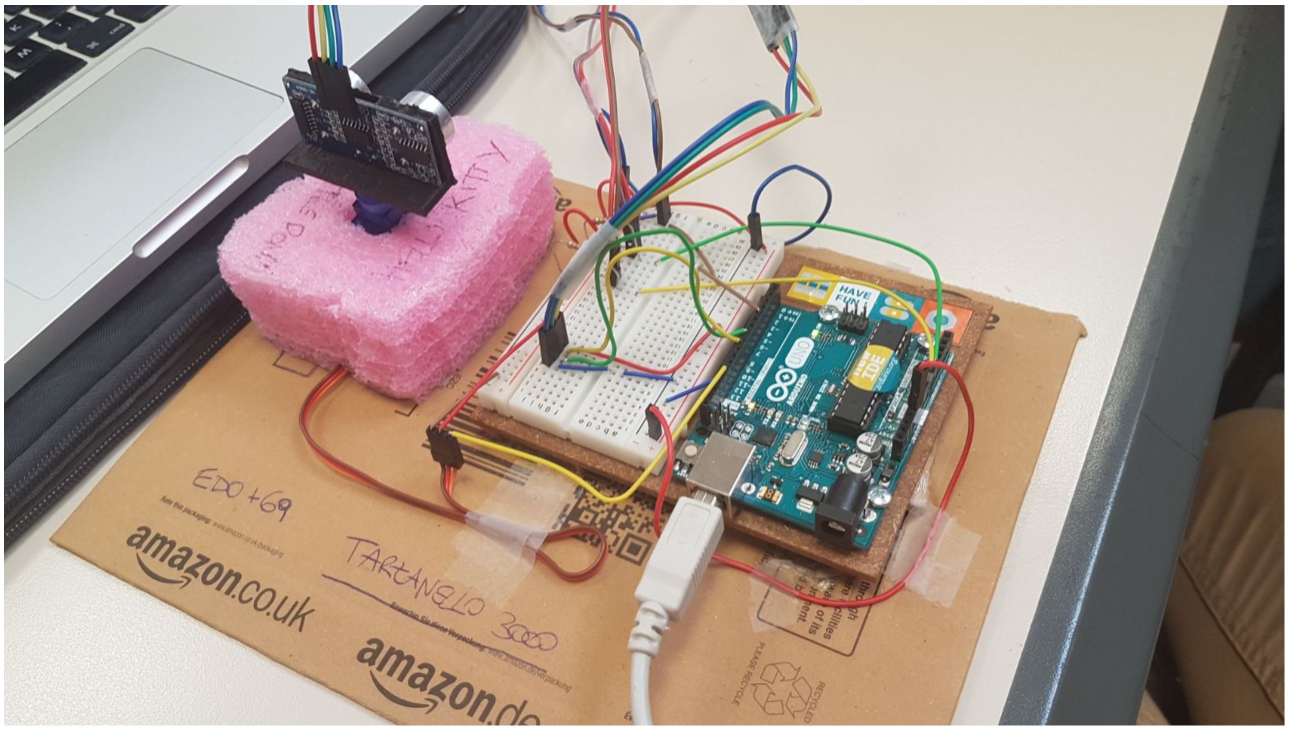

Fig. 2. physical hardware implementation

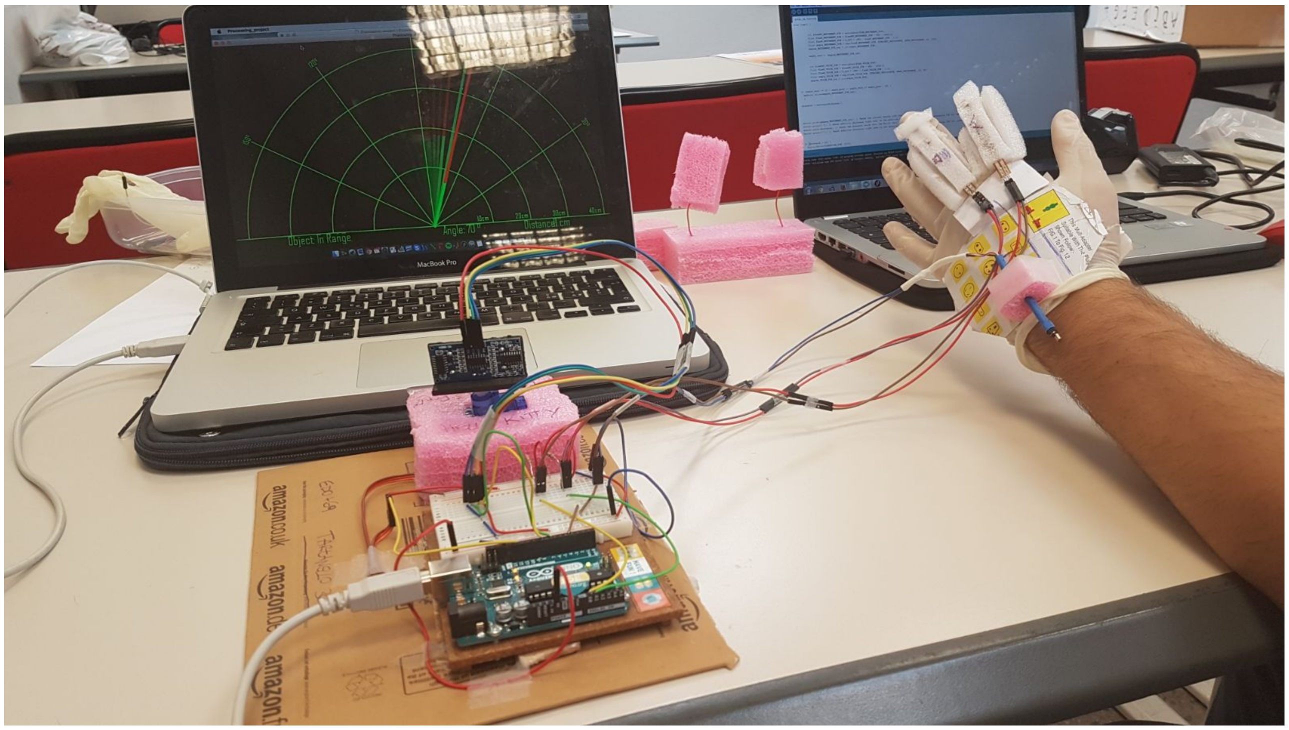

Fig. 3. physical hardware implementation

C. Software - Control Architecture

1) Arduino Code: The Arduino code controls all the sensors and actuators previously described. The Arduino script (Appendix A), after initializing all the needed variables, is structured in two parts. (1) SETUP: the objective of this part (from line 28 to 36) is to activate the required pin connections highlighting their related input or output behavior; (2) LOOP: this part of the script is cyclically executed. First of all (between lines 43 and 59) according to the Flexo-sensors resistance variations measurement (due to the forefinger and middle-finger movements), we compute the bending angles. This is done by applying the computational methods (“resistance-to-angle”), as described in the flexo-sensor datasheet [6]. Then, from line 62 to 65, if the angular position increment between two consecutive algorithm iterations is greater than 10 degrees (in absolute value), we use a function of the servo library to apply a servo-motor rotation imposed by the bending angle of the first Flexo-sensor(forefinger). In other words, we used a mapping between the finger angle to a range of motion of the servo-motor between 0 (sonar directed totally on the right) to 180 degrees (sonar directed totally on the left). The 10 degrees threshold, instead, represents a simple hard-coded solution to filter unintended sensor rotation due to muscular noise - small involuntary contractions - in the user finger. This specific value was reached by a trial-and-error approach aimed to optimal noise reduction. Once the angular position of the Servo motor (and of the Sonar sensor) is reached, we call a function called ”calculateDistance()” in order to receive the distance of the closest obstacle (along the sensor direction) measured by the Sonar sensor (line 68). From line 71 to 77, we use the distance value found in the last step to alarm the user about an imminent obstacle. As matter of facts, when the distance previously obtained is smaller than 10 cm, we activate the vibration actuator. Finally, the lines between 80 and 93 are aimed to create a communication between Arduino and Processing software through the serial port. We build up a protocol in order to easily identify each information sent by Arduino and received by Processing: the first sent value is the angular position of the Servo-motor (and of the Sonar sensor) which is ended by ”,” .The second is the distance measured by the sonar sensor which is ended by ”!”. The third is a boolean value related to bending angle of the second Flexo-sensor (middle-finger) which is equal to 1 only if the angle is bigger than 70 degrees (second trigger held) and zero otherwise. The end of the data package is then indicated by ”.”.

2) Processing Code: The main aspects of the processing code are: (1) Extract the information from the serial port about (a) middle-finger state (1 or 0), (b) angle of the servo-sensor system and (c) detected distance; (2) Convert state of the middle finger to a request for audio feedback; (3) When requested, convert information of servo angle and measured distance in the form of speech; (4) Display the same information by a graphical interface in order to check the accuracy of the algorithm.

The first part is based on the simple communication protocol that was already explained above. The ”,” and ”!” are used as two separate indexes, which are then used to separate angle, distance and triggering-action value. This is done by the function ”serialEvent” (see lines 61-80).

The second part is made by introducing a if-cycle which convert the flex motion of the user middle finger into a ”mouse click”-like action. This is coded between line 48 and 59 of the Appendix B. Notice that, the algorithm is made in such a way that a single (and not continuous) audio feedback is given when the middle finger is flexed (state = 1). In order to trigger a new ”reading”, the user must first move the finger back to its neutral state (flexion less then 70 degrees).

Text-to-speech action is made by adapting a previous library from [7]. In summary, it define a static class which allows to trigger the built-in ”say” (text-to-speech) command of a Macintosh. Additionally, it allows to choose different tonality and speed of speech (by properly defining the two variable ”voiceIndex” and ”voiceSpeed”). This can be found between line 181 and 240 of Appendix B. Then in line 79 a string containing the text to be read by the computer is constantly updated with the system information above described.

Part of the Processing code is aimed to the design of a visual interface. This let us check whether the Audio signal was consistent with the measured data. The graphic interface represents a simple radar screen, based on a library retrieved in [8]. The script is based on four functions: (1) ”drawRadar” (which draw the radar display); (2) ”drawText” (which draws the text and angular-distance information from the servo motor and sensor); (3) ”drawLine” (which draws the radar line according to the servo angle); (4) ” drawObject” (which color the radar line of red starting from the distance of a detected object). Notice that we limited the range of detectable object by the visual interface to 40 cm (even if our sensor can provide information for farther entities). All these functions can be find in the lines between 83 and 177 of Appendix B.

III. Results

As it is possible to see from the movie attached to this pdf file, the system worked correctly: the servo rotation was properly mapped to the flexion of the forefinger (in a continuous mapping approach); the graphic interface correctly visualize the system state; the speech information were consistent with those displayed and were triggered once per each flexion of the middle finger; the vibro-tactile feedback was well received by the user only when a distance of less the 10 cm was detected. Furthermore, unintended sensor rotation due to innate inability of human to maintain body segments perfectly steady - in this case muscular noise in the forefinger - were minimal.

IV. Discussion and Conclusion

The results obtained are shows that the thought device has the potential to tackle the interface issues described in the introduction part. However, it is important to remark a set of limitations encountered. First, even if we try to optimize as possible the glove ergonomics, issues in wearing it still remains. Future applications should present more reliable and intuitive wear in-off solution. Secondly, we limited the exploration of the external environment to 1 degrees of freedom problem, i.e. static rotation of a sensor. To allowing a human to be able to orientate him/herself in a complex environment, this system provide to limited information. However, the goal here was to provide a proof-of-concept experiment in order to test our haptic-audio feedback concept. In this regards, the solution provide very satisfactory results. Finally, the text-to-speech algorithm relies on the use of a Macintosh, since it is based on its built-in terminal function “say” (for more detail see [7]). Regardless of these limitations, our instrumented glove and sonar sensor shows that the usage of a haptic-based interface for interact with electronics might be an interesting approach for those people that can not rely on vision, and that this protocol can work in harmony with more rich signal such as audio feedback. Future work should be aimed to convert the software into a smartphone application, allowing a mobile version of the fixed platform. Additionally, decoupling the glove and the processing platform could be done by implementing a wireless connection, here not explored because of the time constraints. Richness of information could be achieved by implementing a parallel sensor (for instance one sensor for each shoulder and one glove per each hand), so that stereo-exploration of the external environment might be feasible. However, exploring mobile solutions requires also to deal with important aspects such as user-induced vibrations (such as during walking) and power supplies.

V. Appendix A: Arduino Full-Code

```
#include <Servo.h>

// Defines Tirg and Echo pins of the Ultrasonic Sensor
const int trigPin = 8;
const int echoPin = 7;

long duration;
int distance;
int degree_MOUVEMENT_PIN_int;
int degree_VOICE_PIN_int;

// Creates two int variables one for the current angle and one for the angle of the last step
int angle_curr = 0;
int angle_prec = 0;

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

const int FLEX_MOUVEMENT_PIN = A0;
const int FLEX_VOICE_PIN = A1;
const int VIBRATION_PIN = 4;
const float VCC = 4.98; // Measured voltage of Ardunio 5V line
const float R_DIV = 9800.0; // Measured resistance of 3.3k resistor
const float STRAIGHT_RESISTANCE = 28500.0;  // resistance when straight
const float BEND_RESISTANCE = 60000.0;  // resistance when finger is completely bended



void setup() {
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input
  pinMode(FLEX_MOUVEMENT_PIN, INPUT);
  pinMode(FLEX_VOICE_PIN, INPUT);
  pinMode(VIBRATION_PIN, OUTPUT);
  Serial.begin(9600);
  myServo.attach(12); // Defines on which pin is the servo motor attached
}




void loop() {

// Flexo_sensors: definition, resistance variation --> angle variation

  int flexADC_MOUVEMENT_PIN = analogRead(FLEX_MOUVEMENT_PIN);
  float flexV_MOUVEMENT_PIN = flexADC_MOUVEMENT_PIN * VCC / 1023.0;
  float flexR_MOUVEMENT_PIN = R_DIV * (VCC / flexV_MOUVEMENT_PIN - 1.0);
  float angle_MOUVEMENT_PIN = map(flexR_MOUVEMENT_PIN, STRAIGHT_RESISTANCE, BEND_RESISTANCE, 10, 170);
  degree_MOUVEMENT_PIN_int = int(angle_MOUVEMENT_PIN);

  angle_curr =  degree_MOUVEMENT_PIN_int;


  int flexADC_VOICE_PIN = analogRead(FLEX_VOICE_PIN);
  float flexV_VOICE_PIN = flexADC_VOICE_PIN * VCC / 1023.0;
  float flexR_VOICE_PIN = R_DIV * (VCC / flexV_VOICE_PIN - 1.0);
  float angle_VOICE_PIN = map(flexR_VOICE_PIN, STRAIGHT_RESISTANCE, BEND_RESISTANCE, 10, 90);
  degree_VOICE_PIN_int = int(angle_VOICE_PIN);



// We allow the Servo rotation only if the finger bending is clear: this is done in order to remove oscillations in servo motor positions
  if (angle_curr >= 10 + angle_prec || angle_curr <= angle_prec - 10) {
    myServo.write(degree_MOUVEMENT_PIN_int);
    }


  distance = calculateDistance();


// Vibration sensor
```

```
  if (distance < 10) {
    digitalWrite(VIBRATION_PIN, HIGH);
    delay(300);
   }

  digitalWrite(VIBRATION_PIN, LOW);


// Sends information through Serial Port
  Serial.print(degree_MOUVEMENT_PIN_int); // 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


// Audio feedback
  if(degree_VOICE_PIN_int > 70 ) {
          Serial.print("1");
          Serial.print(".");
          }else{
          Serial.print("0");
          Serial.print(".");}


  angle_prec = angle_curr;

  delay(50);
  }



// Function which calculates the distance
int calculateDistance(){
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10); // Sets the trigPin on HIGH state for 10 micro seconds
  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;
}
```

VI. Appendix B: Processing Full-Code

```
/*   Arduino Radar Project
 */
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

String angle="";
String distance="";
String data="";
String trigger;
String noObject;
float pixsDistance;
int iAngle, iDistance, iTrigger;
int iTrigger_new = 0;
int index1=0;
int index2=0;
PFont orcFont;
String script ;
int voiceIndex = 1;
int voiceSpeed = 230;

void setup() {

 size (1300, 700); // Initialization Java Window
 smooth();
 myPort = new Serial(this,Serial.list()[1], 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 and trigger.
 orcFont = loadFont("OCRAExtended-30.vlw");
}

void draw() {

  // Drawing a "radar" display  to verify the accuracy of the audio feedback //
  fill(98,245,31);
  textFont(orcFont);
  // simulating motion blur and slow fade of the moving line
  noStroke();
  fill(0,4);
  rect(0, 0, width, height-height*0.065);
  fill(98,245,31); // green color
  // calls the functions for drawing the radar
  drawRadar();
  drawLine();
  drawObject();
  drawText();

  // Detecting Audio-Trigger Signal//
  /* Functionality Explained: two variables are used in order to let the bending action of
    the middle finger
  to work as a "mouse click". iTrigger is the signal from the device: "1" means that the user
     is flexing is finger,
  while "0" indicates no action from the user. When the user flex is finger for the first
    time, an audio feeback is
  triggered. However, to avoid that the computer continues to talk for the all duration of
    figer flexion, iTrigger_new is used.
  Thanks to the below additional condition, each time the finger is flexed only one "read" is
     made by the PC. To trigger a
  new "reading", the user needs to move his or her middle finger back to the resting position
     (not flexed). */
  if (iTrigger == 1 && iTrigger_new != iTrigger){
    TextToSpeech.say(script, TextToSpeech.voices[voiceIndex], voiceSpeed); // This command
    makes the computer to convert the text into speech
    iTrigger_new = iTrigger;}
    else if (iTrigger == 0 && iTrigger_new != iTrigger){
      iTrigger_new = iTrigger;}
}

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
    "
  index2 = data.indexOf("!"); // find the character '!' and puts it into the variable "index2
    "
  angle= data.substring(0, index1); // read the data from position "0" to position of the
    variable index1 (thats the value of the angle)
  distance= data.substring(index1+1, index2); // read the data from position "index1" to the
    "index2" (thats the value of the distance)
  trigger= data.substring(index2+1, data.length()); // read the data from position "index2"
    to the end of the data (thats the triggering command)

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

  // The string text that is converted into speech is fed with current information about
    measured distance and servo-motor angular position
  script = angle+"degree and"+distance+"centimeter";
}


void drawRadar() {
  pushMatrix();
  translate(width/2,height-height*0.074); // moves the starting coordinats to new location
  noFill();
  strokeWeight(2);
  stroke(98,245,31);
  // draws the arc lines
  arc(0,0,(width-width*0.0625),(width-width*0.0625),PI,TWO_PI);
  arc(0,0,(width-width*0.27),(width-width*0.27),PI,TWO_PI);
  arc(0,0,(width-width*0.479),(width-width*0.479),PI,TWO_PI);
  arc(0,0,(width-width*0.687),(width-width*0.687),PI,TWO_PI);
  // draws the angle lines
  line(-width/2,0,width/2,0);
  line(0,0,(-width/2)*cos(radians(30)),(-width/2)*sin(radians(30)));
  line(0,0,(-width/2)*cos(radians(60)),(-width/2)*sin(radians(60)));
  line(0,0,(-width/2)*cos(radians(90)),(-width/2)*sin(radians(90)));
  line(0,0,(-width/2)*cos(radians(120)),(-width/2)*sin(radians(120)));
  line(0,0,(-width/2)*cos(radians(150)),(-width/2)*sin(radians(150)));
  line((-width/2)*cos(radians(30)),0,width/2,0);
  popMatrix();
}

void drawObject() {
  pushMatrix();
  translate(width/2,height-height*0.074); // moves the starting coordinats to new location
  strokeWeight(9);
  stroke(255,10,10); // red color
  pixsDistance = iDistance*((height-height*0.1666)*0.025); // covers the distance from the
    sensor from cm to pixels
  // limiting the range to 40 cms
  if(iDistance<40){
    // draws the object according to the angle and the distance
  line(pixsDistance*cos(radians(iAngle)),-pixsDistance*sin(radians(iAngle)),(width-width
    *0.505)*cos(radians(iAngle)),-(width-width*0.505)*sin(radians(iAngle)));
  }
  popMatrix();
}
```

```

void drawLine() {
  pushMatrix();
  strokeWeight(9);
  stroke(30,250,60);
  translate(width/2,height-height*0.074); // moves the starting coordinats to new location
  line(0,0,(height-height*0.12)*cos(radians(iAngle)),-(height-height*0.12)*sin(radians(iAngle))); // draws the line according to the angle
  popMatrix();
}


void drawText() { // draws the texts on the screen

  pushMatrix();
  if(iDistance>40) {
  noObject = "Out of Range";
  }
  else {
  noObject = "In Range";
  }
  fill(0,0,0);
  noStroke();
  rect(0, height-height*0.0648, width, height);
  fill(98,245,31);
  textSize(25);

  text("10cm",width-width*0.3854,height-height*0.0833);
  text("20cm",width-width*0.281,height-height*0.0833);
  text("30cm",width-width*0.177,height-height*0.0833);
  text("40cm",width-width*0.0729,height-height*0.0833);
  textSize(40);
  text("Object: " + noObject, width-width*0.875, height-height*0.0277);
  text("Angle: " + iAngle +"   ", width-width*0.48, height-height*0.0277);
  text("Distance: ", width-width*0.26, height-height*0.0277);
  if(iDistance<40) {
  text("        " + iDistance +" cm", width-width*0.225, height-height*0.0277);
  }
  textSize(25);
  fill(98,245,60);
  translate((width-width*0.4994)+width/2*cos(radians(30)),(height-height*0.0907)-width/2*sin(radians(30)));
  rotate(-radians(-60));
  text("30  ",0,0);
  resetMatrix();
  translate((width-width*0.503)+width/2*cos(radians(60)),(height-height*0.0888)-width/2*sin(radians(60)));
  rotate(-radians(-30));
  text("60  ",0,0);
  resetMatrix();
  translate((width-width*0.507)+width/2*cos(radians(90)),(height-height*0.0833)-width/2*sin(radians(90)));
  rotate(radians(0));
  text("90  ",0,0);
  resetMatrix();
  translate(width-width*0.513+width/2*cos(radians(120)),(height-height*0.07129)-width/2*sin(radians(120)));
  rotate(radians(-30));
  text("120  ",0,0);
  resetMatrix();
  translate((width-width*0.5104)+width/2*cos(radians(150)),(height-height*0.0574)-width/2*sin(radians(150)));
  rotate(radians(-60));
  text("150  ",0,0);
  popMatrix();
}

// the text to speech class //
```

```java
import java.io.IOException;

static class TextToSpeech extends Object {

  // Store the voices, makes for nice auto-complete in Eclipse

  // male voices
  static final String ALEX = "Alex";
  static final String BRUCE = "Bruce";
  static final String FRED = "Fred";
  static final String JUNIOR = "Junior";
  static final String RALPH = "Ralph";

  // female voices
  static final String AGNES = "Agnes";
  static final String KATHY = "Kathy";
  static final String PRINCESS = "Princess";
  static final String VICKI = "Vicki";
  static final String VICTORIA = "Victoria";

  // novelty voices
  static final String ALBERT = "Albert";
  static final String BAD_NEWS = "Bad News";
  static final String BAHH = "Bahh";
  static final String BELLS = "Bells";
  static final String BOING = "Boing";
  static final String BUBBLES = "Bubbles";
  static final String CELLOS = "Cellos";
  static final String DERANGED = "Deranged";
  static final String GOOD_NEWS = "Good News";
  static final String HYSTERICAL = "Hysterical";
  static final String PIPE_ORGAN = "Pipe Organ";
  static final String TRINOIDS = "Trinoids";
  static final String WHISPER = "Whisper";
  static final String ZARVOX = "Zarvox";

  // throw them in an array so we can iterate over them / pick at random
  static String[] voices = {
    ALEX, BRUCE, FRED, JUNIOR, RALPH, AGNES, KATHY,
    PRINCESS, VICKI, VICTORIA, ALBERT, BAD_NEWS, BAHH,
    BELLS, BOING, BUBBLES, CELLOS, DERANGED, GOOD_NEWS,
    HYSTERICAL, PIPE_ORGAN, TRINOIDS, WHISPER, ZARVOX
  };

  // this sends the "say" command to the terminal with the appropriate args
  static void say(String script, String voice, int speed) {
    try {
      Runtime.getRuntime().exec(new String[] {"say", "-v", voice, "[[rate " + speed + "]]" +
    script});
    }
    catch (IOException e) {
      System.err.println("IOException");
    }
  }

  // Overload the say method so we can call it with fewer arguments and basic defaults
  static void say(String script) {
    // 200 seems like a resonable default speed
    say(script, ALEX, 200);
  }

}
```

References

[1] Soviak, A., Borodin, A., Ashok, V., Borodin, Y., Puzis, Y., & Ramakrishnan, I. V. (2016, October). Tactile Accessibility: Does Anyone Need a Haptic Glove?. In Proceedings of the 18th International ACM SIGACCESS Conference on Computers and Accessibility (pp. 101-109). ACM.

[2] Pascolini, D., & Mariotti, S. P. (2012). Global estimates of visual impairment: 2010. British Journal of Ophthalmology, 96(5), 614-618.

[3] Soviak, A. (2015, October). Haptic Gloves Prototype for Audio-Tactile Web Browsing. In Proceedings of the 17th International ACM SIGACCESS Conference on Computers & Accessibility (pp. 363-364). ACM.

[4] Woniak, P., Knaving, K., Obaid, M., Carcedo, M. G., nler, A., & Fjeld, M. (2015, March). ChromaGlove: a wearable haptic feedback device for colour recognition. In Proceedings of the 6th Augmented Human International Conference (pp. 219-220). ACM.

[5] Soviak, A. (2015, May). Haptic gloves for audio-tactile web accessibility. In Proceedings of the 12th Web for All Conference (p. 40). ACM.

[6] Flex Sensor 2.2". (n.d.). Retrieved November 16, 2017, from https://learn.sparkfun.com/tutorials/flex-sensor-hookup-guide

[7] Frontier Nerds: An ITP Blog. (n.d.). Retrieved November 16, 2017, from http://www.frontiernerds.com/text-to-speech-in-processing

[8] 45. Arduino sonar radar con processing. (n.d.). Retrieved November 16, 2017, from https://www.progettiarduino.com/45-arduino-sonar-radar-con-processing.html