**ITSP Documentary**

PROJECT NAME:**3D Hologram controlled by hand gestures**

**TEAM NAME**:TGAV

**TEAM MEMBERS**:

Anshul Tomar(Team Leader), Anugole Sai Gaurav, Anwesh Mohanty, Vishwas Bharti

**WEEK 1:**

* We have firstly divided the work among us wherein Anshul and Vishwas would together design the sensor board circuit, Anwesh would make a model in Unity and Gaurav would interface Arduino with keyboard.
* Gaurav had searched for controlling keyboard with Arduino which was necessary as the controls for rotating left/right and zooming in/out of Unity model made by Anwesh was typing-key controlled one. So, Gaurav came up with <https://www.arduino.cc/reference/en/language/functions/usb/keyboard/keyboardwrite/> site wherein it was possible for arduino to send a keystroke to the connected computer using the function Keyboard.Write(). He looked up to some on-site available examples and designed a algorithm which he planned to code it later. Meanwhile he came across some related functions like:

Keyboard.Begin()- <https://www.arduino.cc/reference/en/language/functions/usb/keyboard/keyboardbegin/>

Keyboard.End()-

<https://www.arduino.cc/reference/en/language/functions/usb/keyboard/keyboardend/> Keyboard.Print()- <https://www.arduino.cc/reference/en/language/functions/usb/keyboard/keyboardprint/>

Similarly for Mouse like: Begin(), End() and the one unique from keyboard was

Mouse.Click()- <https://www.arduino.cc/reference/en/language/functions/usb/mouse/mouseclick/>

But Next day, when he made a sample code and uploaded it on Arduino, he got errors in Keyboard functions saying that when the library (keyboard.h) was included in the code in spite of including it. Later, on troubleshooting the issue, he realised that these core libraries (including mouse.h) allow the Atmega32u4 processor and SAMD based boards only like Leonardo, Esplora, Zero, Due and MKR Family to appear as a native Mouse and/or Keyboard to a connected computer but he was using Arduino Uno which doesn’t feature FTDI USB to serial driver chip, instead uses an Atmega16U2 programmed USB to serial converter.

He then searched on some similar links like <https://youtu.be/pFoqUhn1m5w> , <https://youtu.be/AKWheG8wKxM> , etc which probably woudln’t work. He tried understanding to interface a keypad with Arduino in here: <https://circuitdigest.com/microcontroller-projects/keypad-interfacing-with-arduino-uno> . He then came across <http://www.instructables.com/id/Arduino-Uno-Into-Usb-Keyboard/> wherein it was first told to install unojoy firmware and atmel flip software, Upload 3-4 files like simplejoystick.ino file and finally install JoyToKey software that emulates a keyboard. When he was about to do so, he came across some user under the comments section who got messed up in middle and couldn’t reset back his Arduino to normal. So, he gave up this idea too. He then tried controlling at least the OnScreen Keyboard with Arduino but couldn’t do so. He then came across <http://mitchtech.net/arduino-usb-hid-keyboard/> wherein it was told to download a USB HID keyboard firmware and update the Arduino through DFU(Device Firmware Update- A Special USB protocol). Keyboard keys were then read through a report buffer specifically designed for this file. However, after compiling and uploading there was no output. This one too proved futile. He also came across the same in Github which however was Linux based , using ubuntu repositories. He finally decided to buy Leonardo or Mega.

For two days , he learnt to develop a simple Unity based game <https://www.youtube.com/693c3692-ff3e-4b4c-a0a1-7571de7fcbd7> from here.

The following day he came up with the idea of integrating Arduino to Unity instead of keyboard. This video <https://www.youtube.com/watch?v=of_oLAvWfSI> has a step wise procedure of how to do so. In Unity firstly the Project API configuration settings has to be changed to NET2.0 from its subset. Later in the script, a library System.IO.Ports has to be included, an object of type SerialPort has to be made which was dynamically allocated and given its port name( like COM3) and baud rate(9600) as parameters. Later instead of using keyboard keys as controls he made a function that takes in serially read parameter from Arduino file and hence re-made the previous code as desired. He also changed the Arduino code and took outputs from IR sensors which Vishwas made on a breadboard. It worked.

* Anwesh’s first week task was to find a suitable 3D model for our hologram and learn the basics of unity like how to rotate and scale and object in unity.

The model that Anwesh used in the project is taken from the unity asset store(<https://assetstore.unity.com/packages/essentials/tutorial-projects/survival-shooter-tutorial-40756>). To give a proper holographic display, Anwesh had to duplicate the game object 4 times and keep them at 90 degree to each other and ensure that when they rotate they were in sync with each other.

After setting up the initial scene, Anwesh had to write individual scripts for rotating and scaling the object. Initially the keys were mapped to only the keyboard with the up and down arrows used for scaling the object and the right and left arrow used for rotating the object.

The unity(which uses C# language) commands-

1. Player.gameObject.transform.Rotate(omega\*Time.deltaTime)-

<https://docs.unity3d.com/ScriptReference/Transform.Rotate.html>

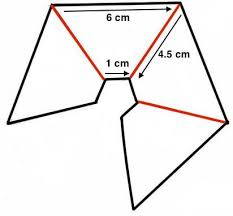
This rotates the player about the X,Y and Z axes at an angular speed of omega (omega is a 3D vector). Time.delta ensures that in every frame the angular speed remains omega so that the motion is either not too fast nor too slow. This function was mapped to the right and left arrow keys.

2. Player.gameObject.transform.localScale+=new Vector3 (scaleFactor,scaleFactor,scaleFactor)-

<https://docs.unity3d.com/ScriptReference/Transform-localScale.html>

This uniformly scales the game object by a factor(=scaleFactor) in all the X,Y and Z directions. This function was mapped to the up and down arrow keys.

These two scripts were attached to the all the 4 game objects at first and when tested, though it worked but there was some problem in syncing the rotation. After a few hit and trials we correctly found the axis direction about which the objects should rotate so as to form a proper holographic image.

* Anshul found out the maximum array of ir sensors we will be able to use given we have 2 Arduino’s.Also we found out the size and material of the frustum we will be using for creating the hologram.The dimensions are as shown below and any other multiple of the dimensions.The final dimension will be selected based on the screen size. For now we think that we will be able to use both digital and analog pins for input(but if not we will have to shrink the array further).
* We are ordering acrylic board for the frustum.Anshul searched for the dimensions of the frustum.We will order it.Vishwas planned to laser cut the faces but since the laser cutter is not working we have to think for other method.Also he is thinking to use wood for the IR sensor board.To avoid the clumsiness over the board Vishwas advised to keep the resistors below the board.By doing this the board will look good and there will be no obstacle in the IR signal.Wood was chosen because ,it’s easy to cut holes in it and it’s insulating as well.We have also decided to update our board design from a 16\*16 sensor grid to 4\*4 sensor grid.It’s because we can give 26 inputs to 2 arduinos.This is just to avoid clumsiness in our model.We can add more sensors later.

**WEEK 2:**

* This week Gaurav explained to Anwesh regarding Unity-Arduino interfacing. Gaurav found a Unity library called System.IO.Ports which enables the unity engine to receive inputs from the COM5 port in the laptop via the arduino. Gaurav and Vishwas had made a simplistic model consisting of 4 IR LEDs which gave their outputs to the arduino.

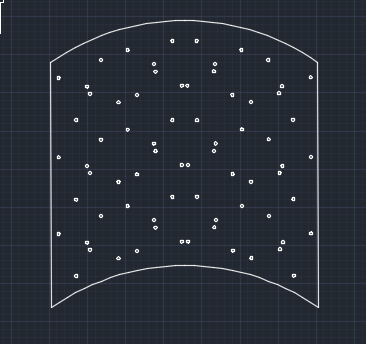
The unity engine then checks if the serial port is open or not using the function sp.IsOpen. If the port is open then the game checks which of the four IR LED is active and transmits the info to the unity game engine. Based on the data received by the arduino, the game uses one of the four functions(rotate left, rotate right, zoom in and zoom out) which have been changed according to accommodate the interfacing between the arduino and the unity game engine.

We faced quite some problems in implementing this. Initially we had applied scripts to individual objects in the game which led to creation of multiple instances during execution. This meant only one of the 4 objects was rotating and scaling whereas the other 3 objects didn’t move. We tried the same method in Gaurav’s game but unfortunately we encountered the same problem of “**Access Denied**”.

On browsing through the internet, Anwesh found that instead of creating single references to an object in multiple scripts, it’s better to create multiple references to objects in a single script. So Anwesh wrote a single script for both zooming and rotating and it worked.

* Vishwas and Anshul made the presentation for the 1st week.We will be ordering the display screen also <https://www.amazon.in/Lap-Gadgets-15-N208TX-SVE151B11W-replacement/dp/B01ELKEBR2/ref=sr_1_4?s=computers&ie=UTF8&qid=1528453747&sr=1-4&keywords=display+screen+for+laptop> in the next week .We are not hurrying in buying it.For now we will test our codes and frustum on the laptop screen that we have.We will buy the acrylic board from a nearby shop.

**WEEK 3:**

* Gaurav has purchased acrylic board from *Swastik Aluminium & Hardware store* opposite IIT main gate. Then, on the day when the laser cutting machine at tinkerer’s lab was ready to use, we laser-cut the sheet into 4 trapeziums of dimensions 38.1cm, 6.35cm (for parallel sides and 27.3cm for non parallel sides . For laser cutting we had to first create the template of the trapezium to be cut in Autocad and save it as .dxf file. We then had to open it in the pc connected to the machine and upload it after selecting suitable speed of the laser cutter and its power range. After uploading, we had to select a suitable origin and then check the frame size. Then the sheet has to be laser cut. After the sheet has been cut, we removed the insulation and made the pyramid by sticking with an insulation tape. We then tested the hologram of the model made by Anwesh and we were able to see the 3d perspective of it as we desired. We then tested the features of rotation and zooming with the ir sensors made on breadboard and we indeed were able to control the same of that model. Later that day, we also saw some hologram videos from youtube through the pyramid. We enjoyed watching those.
* Vishwas has purchased plywood for our sensorboard and some set of ir sensors(pairs of ir leds and ir photodiodes) and blue leds from *Mangaldeep* at IIT Market. We then designed the sensor board in Autocad. We decided to use 12 sets of IR sensors with each set have 4 leds connected in parallel so that it behaves like a proximity sensors. We dimensioned the board accordingly with breadth being 35cm and made the length curved for better looking. We then made it as a 4 x 3 matrix as shown below 

We then decided to laser cut the same on plywood. In Spite of maximising the power, the laser couldn’t cut through the plywood. We repeated the same 5-6 times, but it was in vain. The laser has cut only through one third of the plywood.We then had to manually cut out the board and all the holes were drilled using the drill machine. But there was one benefit of laser cutting that it had provided the boundary of the board and the location of the holes

**WEEK 4:**

* Vishwas started soldering the connections in the circuit board. For each set of sensors, the ir led with 220Ω resistor in series was connected parallel to ir photodiode(reverse biased) with 4.7kΩ in series. All 4 leds were connected in parallel to the 4.7kΩ resistor thereby acting lke a proximity sensor. The first two columns had common Vcc and ground and similarly for the other two. This was done so that each would go into either Arduino (uno) and they can be serially communicated. Inorder to make one set it used to take 30-40 minutes for him. He used the solder wire to connect the components which were later insulated. Jumpers were soldered to each set one before ir led where voltage would be provided through digital pins of Arduino. The photodiode was directly connected to Vcc. Another jumper was soldered between the photodiode and the resistor to analog read the values to Arduino. Inorder to lengthen the jumpers, female and male jumpers were connected to the male ones which were soldered. By weekend, he managed to complete the sensor board.
* Anshul: Initially the code consisted of two parts the receiver part and the sender part. The left six ir sensors(3x2) were connected to the receiver arduino and the right six ir sensors(3x2) were connected to the sender arduino. From each ir sensor there were four wires coming out:-1)Vcc 2)Gnd 3)Output of the ir sensor taken at the analog input of arduino uno 4)Digital input to the ir led given from the digital pins of the arduino set at output mode.The first thing in the code was to calibrate the ir sensors.This was done by first sending a digital low to the ir led. This implied that the analog input given by the ir sensor was the ambient ir present in the room.Then a digital high was given to the ir led which meant that the ir sensor was now measuring the intensity of the total infrared radiation(let’s say it as obstacle intensity) coming to it after getting reflected from the walls.The ambient intensity was now subtracted from the obstacle intensity to get the true amount of ir led light being reflected from the walls. This measurement was taken 5 times for each ir led,sensor pair and their average was stored in their respective variable calibrateIR\_XX.This value will be used to depict the condition when no object is present in front.The next was the readIR function whose function was to read the intensity at all the pins and store it in a array of 100 integer values.If the value of all the photodiodes are zerothen the function readIR sets the array top index to zero and the array is net to null.If any of the photodiodes detect an obstacle (i.e the intensity detected > calibrateIR\_XX + 50 ( 50 is used here so as to clearly distinguish between an obstacle and a no obstacle condition)) the readIR function starts filling up its array until it overflows after which it resets the top array index to zero. The expected gesture function determines the gesture based on the4 array values. For each photosensor two new values are calculated total positive distance and total negative distance.Total positive distance is calculated as the sum of difference of the distances(l[j] - l[j-1]) where l[j] is greater than l[j-1] denoting an increase in intensity or a decrease in distance.Similarly total negative distance is calculated as the sum of difference of the distances(l[j-1] - l[j]) where l[j] is smaller than l[j-1[ denoting a decrease in intensity or increase in distance.Now based on these values the gesture is determined as either east or west.You can see the cut-off values for the variables distancePos and distanceNeg in the code. For communicating between the two arduino’s we used one arduino as the sender which was used to send the input to other arduino using two digital pins.In the default state(when no gesture is given or a random gesture is given) the sender arduino sends digital low on both pins.When east gesture is given it sends 01 and when west gesture is given it sends 10 to the receiver arduino.According to the gesture conceived by the receiver arduino and the output of the sender arduino the four functions are determined. This method was not working as we were not able to synchronize both the arduinos such that if one sends then the other arduino reaches the exact line of code where it has read that value.the two code are as shown below:-
* Arduino Uno 1(sender):

#define IR\_RX\_TL A0 // Top Left IR receiver photodiode on analog pin A0

#define IR\_RX\_TR A1 // Top Right IR receiver photodiode on analog pin A1

#define IR\_RX\_BL A2 // IR receiver photodiode on analog pin A2

#define IR\_RX\_BR A3 // IR receiver photodiode on analog pin A3

#define IR\_RX\_L A4

#define IR\_RX\_R A5

#define IR\_TX\_TL 2 // Top Left IR emitter LED on digital pin 2

#define IR\_TX\_TR 3 // Top Right IR emitter LED on digital pin 3

#define IR\_TX\_BL 4 // Bottom Left IR emitter LED on digital pin 4

#define IR\_TX\_BR 5 // Bottom Right IR emitter LED on digital pin 5

#define IR\_TX\_L 6

#define IR\_TX\_R 7

bool readGesture = true; // Used to process the gesture

int ambientIR\_TL; // Stores the ambient IR value read from the TL IR receiver

int ambientIR\_TR; // Stores the ambient IR value read from the TR IR receiver

int ambientIR\_BL; // Stores the ambient IR value read from the BL IR receiver

int ambientIR\_BR; // Stores the ambient IR value read from the BR IR receiver

int ambientIR\_L;

int ambientIR\_R;

int obstacleIR\_TL; // Stores the raw object IR value read from the TL IR receiver

int obstacleIR\_TR; // Stores the raw object IR value read from the TR IR receiver

int obstacleIR\_BL; // Stores the raw object IR value read from the BL IR receiver

int obstacleIR\_BR; // Stores the raw object IR value read from the BR IR receiver

int obstacleIR\_L;

int obstacleIR\_R;

int value\_TL[10]; // Stores the IR value read from the TL IR receiver

int value\_TR[10]; // Stores the IR value read from the TR IR receiver

int value\_BL[10]; // Stores the IR value read from the BL IR receiver

int value\_BR[10]; // Stores the IR value read from the BR IR receiver

int value\_L[10];

int value\_R[10];

int distance\_TL; // Stores the mapped IR value read from the TL IR receiver

int distance\_TR; // Stores the mapped IR value read from the TR IR receiver

int distance\_BL; // Stores the mapped IR value read from the BL IR receiver

int distance\_BR; // Stores the mapped IR value read from the BR IR receiver

int distance\_L;

int distance\_R;

int calibration\_TL; // Stores the raw IR value read from the TL receiver when the Arduino is powered

int calibration\_TR; // Stores the raw IR value read from the TR receiver when the Arduino is powered

int calibration\_BL; // Stores the raw IR value read from the BL receiver when the Arduino is powered

int calibration\_BR; // Stores the raw IR value read from the BR receiver when the Arduino is powered

int calibration\_L;

int calibration\_R;

int lastDistanceHold\_TL; // The next variables are used to detect a "hold gesture" action

int lastDistanceHold\_TR;

int lastDistanceHold\_BL;

int lastDistanceHold\_BR;

int lastDistanceHold\_L;

int lastDistanceHold\_R;

int holdMillis = 300; // Used in a timer to determine a "hold gesture" action

uint8\_t distanceArray\_TL[100]; // Stores the last 100 IR values from the TL IR receiver to determine the gesture later

uint8\_t distanceArray\_TR[100]; // Stores the last 100 IR values from the TR IR receiver to determine the gesture later

uint8\_t distanceArray\_BL[100]; // Stores the last 100 IR values from the BL IR receiver to determine the gesture later

uint8\_t distanceArray\_BR[100]; // Stores the last 100 IR values from the BR IR receiver to determine the gesture

uint8\_t distanceArray\_L[100];

uint8\_t distanceArray\_R[100];

uint8\_t distanceArrayCounter = 0; // Used to access the previous array's content

uint8\_t gesture = 0; // Stores the gesture

uint32\_t lastHoldMillis = 0;

void setup() {

// put your setup code here, to run once:

Serial.begin(9600);

pinMode(IR\_TX\_TL, OUTPUT);

pinMode(IR\_TX\_TR, OUTPUT);

pinMode(IR\_TX\_BL, OUTPUT);

pinMode(IR\_TX\_BR, OUTPUT);

pinMode(IR\_TX\_L, OUTPUT);

pinMode(IR\_TX\_R, OUTPUT);

digitalWrite(IR\_TX\_TL, LOW);

digitalWrite(IR\_TX\_TR, LOW);

digitalWrite(IR\_TX\_BL, LOW);

digitalWrite(IR\_TX\_BR, LOW);

digitalWrite(IR\_TX\_L, LOW);

digitalWrite(IR\_TX\_R, LOW);

calibrateIR();

}

void loop() {

// put your main code here, to run repeatedly:

readIR(5); // Read 5 times the IR values and store the average

determineGesture(); // It determines the type of gesture made

// processHoldGesture(); // It checks if the hand is being hold

}

void calibrateIR(){

for(uint8\_t x = 0; x < 5; x++){

digitalWrite(IR\_TX\_TL,LOW);

digitalWrite(IR\_TX\_TR,LOW);

digitalWrite(IR\_TX\_BL,LOW);

digitalWrite(IR\_TX\_BR,LOW);

digitalWrite(IR\_TX\_L,LOW);

digitalWrite(IR\_TX\_R,LOW);

delay(2);

ambientIR\_TL = analogRead(IR\_RX\_TL);

ambientIR\_TR = analogRead(IR\_RX\_TR);

ambientIR\_BL = analogRead(IR\_RX\_BL);

ambientIR\_BR = analogRead(IR\_RX\_BR);

ambientIR\_L = analogRead(IR\_RX\_L);

ambientIR\_R = analogRead(IR\_RX\_R);

digitalWrite(IR\_TX\_TL,HIGH);

digitalWrite(IR\_TX\_TR,HIGH);

digitalWrite(IR\_TX\_BL,HIGH);

digitalWrite(IR\_TX\_BR,HIGH);

digitalWrite(IR\_TX\_L,HIGH);

digitalWrite(IR\_TX\_R,HIGH);

delay(1);

obstacleIR\_TL = analogRead(IR\_RX\_TL);

obstacleIR\_TR = analogRead(IR\_RX\_TR);

obstacleIR\_BL = analogRead(IR\_RX\_BL);

obstacleIR\_BR = analogRead(IR\_RX\_BR);

obstacleIR\_L = analogRead(IR\_RX\_L);

obstacleIR\_R = analogRead(IR\_RX\_R);

value\_TL[x] = - ambientIR\_TL + obstacleIR\_TL;

value\_TR[x] = - ambientIR\_TR + obstacleIR\_TR;

value\_BL[x] = - ambientIR\_BL + obstacleIR\_BL;

value\_BR[x] = - ambientIR\_BR + obstacleIR\_BR;

value\_L[x] = - ambientIR\_L + obstacleIR\_L;

value\_R[x] = - ambientIR\_R + obstacleIR\_R;

}

for(uint8\_t x = 0; x < 5; x++){

distance\_TL += value\_TL[x];

distance\_TR += value\_TR[x];

distance\_BL += value\_BL[x];

distance\_BR += value\_BR[x];

distance\_L += value\_L[x];

distance\_R += value\_R[x];

}

calibration\_TL = (distance\_TL / 5);

calibration\_TR = (distance\_TR / 5);

calibration\_BL = (distance\_BL / 5);

calibration\_BR = (distance\_BR / 5);

calibration\_L = (distance\_L / 5);

calibration\_R = (distance\_R / 5);

}

void readIR(uint8\_t times){

for(uint8\_t x = 0; x < times; x++){

digitalWrite(IR\_TX\_TL,LOW); // We need to turn off the IR LEDs to read the ambient IR

digitalWrite(IR\_TX\_TR,LOW);

digitalWrite(IR\_TX\_BL,LOW);

digitalWrite(IR\_TX\_BR,LOW);

digitalWrite(IR\_TX\_L,LOW);

digitalWrite(IR\_TX\_R,LOW);

delay(2); // Wait until the IR LEDs are completely turned off

ambientIR\_TL = analogRead(IR\_RX\_TL); // Stores the ambient IR light

ambientIR\_TR = analogRead(IR\_RX\_TR);

ambientIR\_BL = analogRead(IR\_RX\_BL);

ambientIR\_BR = analogRead(IR\_RX\_BR);

ambientIR\_L = analogRead(IR\_RX\_L);

ambientIR\_R = analogRead(IR\_RX\_R);

digitalWrite(IR\_TX\_TL,HIGH); // Turn on the IR LEDs to read the IR light reflected by the obstacle

digitalWrite(IR\_TX\_TR,HIGH);

digitalWrite(IR\_TX\_BL,HIGH);

digitalWrite(IR\_TX\_BR,HIGH);

digitalWrite(IR\_TX\_L,HIGH);

digitalWrite(IR\_TX\_R,HIGH);

delay(1); // Wait until the IR LEDs are completely turned on

obstacleIR\_TL = analogRead(IR\_RX\_TL); // Stores the IR light reflected by the obstacle

obstacleIR\_TR = analogRead(IR\_RX\_TR);

obstacleIR\_BL = analogRead(IR\_RX\_BL);

obstacleIR\_BR = analogRead(IR\_RX\_BR);

obstacleIR\_L = analogRead(IR\_RX\_L);

obstacleIR\_R = analogRead(IR\_RX\_R);

value\_TL[x] = - ambientIR\_TL + obstacleIR\_TL; // Store the IR value to later calculate the average

value\_TR[x] = - ambientIR\_TR + obstacleIR\_TR;

value\_BL[x] = - ambientIR\_BL + obstacleIR\_BL;

value\_BR[x] = - ambientIR\_BR + obstacleIR\_BR;

value\_L[x] = - ambientIR\_L + obstacleIR\_L;

value\_R[x] = - ambientIR\_R + obstacleIR\_R;

}

for(uint8\_t x = 0; x < times; x++){ // Calculate the average IR value per sensor

distance\_TL += value\_TL[x];

distance\_TR += value\_TR[x];

distance\_BL += value\_BL[x];

distance\_BR += value\_BR[x];

distance\_L += value\_L[x];

distance\_R += value\_R[x];

}

distance\_TL = (distance\_TL / times); // Save the distance value

distance\_TR = (distance\_TR / times);

distance\_BL = (distance\_BL / times);

distance\_BR = (distance\_BR / times);

distance\_L = (distance\_L / times);

distance\_R = (distance\_R / times);

if(calibration\_TL + 50 > distance\_TL){ // Calibrate the sensors

distance\_TL = 0;

}

else{

distance\_TL -= calibration\_TL + 50;

}

if(calibration\_TR + 50 > distance\_TR){

distance\_TR = 0;

}

else{

distance\_TR -= calibration\_TR + 50;

}

if(calibration\_BL + 50 > distance\_BL){

distance\_BL = 0;

}

else{

distance\_BL -= calibration\_BL + 50;

}

if(calibration\_BR + 50 > distance\_BR){

distance\_BR = 0;

}

else{

distance\_BR -= calibration\_BR + 50;

}

if(calibration\_L + 50 > distance\_L){

distance\_L = 0;

}

else{

distance\_L -= calibration\_L + 50;

}

if(calibration\_R + 50 > distance\_R){

distance\_R = 0;

}

else{

distance\_R -= calibration\_R + 50;

}

// If any distance is greater than 0 then map it from a 0-1023 range to a 0-255 range (to be able to store it in a 1B variable) and store it in an array

if(distance\_TL != 0 || distance\_TR != 0 || distance\_BL != 0 || distance\_BR != 0 || distance\_L != 0 || distance\_R != 0){

distance\_TR = map(constrain(distance\_TR, 0, 1100), 0, 1100, 0, 255);

distance\_TL = map(constrain(distance\_TL, 0, 1100), 0, 1100, 0, 255);

distance\_BR = map(constrain(distance\_BR, 0, 1100), 0, 1100, 0, 255);

distance\_BL = map(constrain(distance\_BL, 0, 1100), 0, 1100, 0, 255);

distance\_L = map(constrain(distance\_L, 0, 1100), 0, 1100, 0, 255);

distance\_R = map(constrain(distance\_R, 0, 1100), 0, 1100, 0, 255);

if(distanceArrayCounter < 100){

distanceArray\_TR[distanceArrayCounter] = distance\_TR;

distanceArray\_TL[distanceArrayCounter] = distance\_TL;

distanceArray\_BR[distanceArrayCounter] = distance\_BR;

distanceArray\_BL[distanceArrayCounter] = distance\_BL;

distanceArray\_R[distanceArrayCounter] = distance\_R;

distanceArray\_L[distanceArrayCounter] = distance\_L;

distanceArrayCounter += 1;

}

else{

distanceArray\_TR[0] = distance\_TR;

distanceArray\_TL[0] = distance\_TL;

distanceArray\_BR[0] = distance\_BR;

distanceArray\_BL[0] = distance\_BL;

distanceArray\_L[0] = distance\_L;

distanceArray\_R[0] = distance\_R;

distanceArrayCounter = 1;

}

}

else if(distance\_TL == 0 && distance\_TR == 0 && distance\_BL == 0 && distance\_BR == 0 && distance\_L == 0 && distance\_R == 0){ // If not then reset the arrays

readGesture = true;

distanceArrayCounter = 0;

distanceArray\_TR[0] = 0;

distanceArray\_TL[0] = 0;

distanceArray\_BR[0] = 0;

distanceArray\_BL[0] = 0;

distanceArray\_R[0] = 0;

distanceArray\_L[0] = 0;

}

}

void determineGesture(){

if(distanceArrayCounter > 2){ // If the arrays have more than 2 elements then determine the gesture

uint8\_t distancePost\_TL = 0; // The next "distancePost..." vars are the sum of it previous value plus

uint8\_t distancePost\_TR = 0; // the last element minus the previous element of the array

uint8\_t distancePost\_BL = 0;

uint8\_t distancePost\_BR = 0;

uint8\_t distancePost\_L = 0;

uint8\_t distancePost\_R = 0;

uint8\_t distancePostNeg\_TL = 0;

uint8\_t distancePostNeg\_TR = 0;

uint8\_t distancePostNeg\_BL = 0;

uint8\_t distancePostNeg\_BR = 0;

uint8\_t distancePostNeg\_R = 0;

uint8\_t distancePostNeg\_L = 0;

uint8\_t expectedGesture = 0; // Used later to determine the gesture made

bool increasing\_TL = false; // Used later to determine the gesture made. If the IR sensor's value

bool decreasing\_TL = false; // increases then "increasing\_xx" is true. Otherwise, if it decreasing

bool increasing\_TR = false; // then "decreasing\_xx" is true

bool decreasing\_TR = false;

bool increasing\_BL = false;

bool decreasing\_BL = false;

bool increasing\_BR = false;

bool decreasing\_BR = false;

bool increasing\_L = false;

bool decreasing\_L = false;

bool increasing\_R = false;

bool decreasing\_R = false;

// For every element in the distanceArray...

for(uint8\_t x = 1; x < distanceArrayCounter; x++){

// Update the "distancePost..." vars values

if(distanceArray\_TL[x] - distanceArray\_TL[x-1] > 0){

distancePost\_TL += distanceArray\_TL[x] - distanceArray\_TL[x-1];

}

else{

distancePostNeg\_TL += distanceArray\_TL[x-1] - distanceArray\_TL[x];

}

if(distanceArray\_TR[x] - distanceArray\_TR[x-1] > 0){

distancePost\_TR += distanceArray\_TR[x] - distanceArray\_TR[x-1];

}

else{

distancePostNeg\_TR += distanceArray\_TR[x-1] - distanceArray\_TR[x];

}

if(distanceArray\_BL[x] - distanceArray\_BL[x-1] > 0){

distancePost\_BL += distanceArray\_BL[x] - distanceArray\_BL[x-1];

}

else{

distancePostNeg\_BL += distanceArray\_BL[x-1] - distanceArray\_BL[x];

}

if(distanceArray\_BR[x] - distanceArray\_BR[x-1] > 0){

distancePost\_BR += distanceArray\_BR[x] - distanceArray\_BR[x-1];

}

else{

distancePostNeg\_BR += distanceArray\_BR[x-1] - distanceArray\_BR[x];

}

if(distanceArray\_L[x] - distanceArray\_L[x-1] > 0){

distancePost\_L += distanceArray\_L[x] - distanceArray\_L[x-1];

}

else{

distancePostNeg\_L += distanceArray\_L[x-1] - distanceArray\_L[x];

}

if(distanceArray\_R[x] - distanceArray\_R[x-1] > 0){

distancePost\_R += distanceArray\_R[x] - distanceArray\_R[x-1];

}

else{

distancePostNeg\_R += distanceArray\_R[x-1] - distanceArray\_R[x];

}

}

// Based on the "distancePost..." vars determine if the value is increasing or decreasing

if(distancePost\_TL > 10 && distancePostNeg\_TL < 10){

increasing\_TL = true;

decreasing\_TL = false;

}

else if(distancePostNeg\_TL > 10){

increasing\_TL = false;

decreasing\_TL = true;

}

if(distancePost\_TR > 10 && distancePostNeg\_TR < 10){

increasing\_TR = true;

decreasing\_TR = false;

}

else if(distancePostNeg\_TR > 10){

increasing\_TR = false;

decreasing\_TR = true;

}

if(distancePost\_BL > 10 && distancePostNeg\_BL < 10){

increasing\_BL = true;

decreasing\_BL = false;

}

else if(distancePostNeg\_BL > 10){

increasing\_BL = false;

decreasing\_BL = true;

}

if(distancePost\_BR > 10 && distancePostNeg\_BR < 10){

increasing\_BR = true;

decreasing\_BR = false;

}

else if(distancePostNeg\_BR > 10){

increasing\_BR = false;

decreasing\_BR = true;

}

if(distancePost\_L > 10 && distancePostNeg\_L < 10){

increasing\_L = true;

decreasing\_L = false;

}

else if(distancePostNeg\_L > 10){

increasing\_L = false;

decreasing\_L = true;

}

if(distancePost\_R > 10 && distancePostNeg\_R < 10){

increasing\_BR = true;

decreasing\_BR = false;

}

else if(distancePostNeg\_R > 10){

increasing\_R = false;

decreasing\_R = true;

}

// Determine the type of gesture //for present south and north gestures have not been updated

switch(expectedGesture){

case 0: // If "expectedGesture" is 0 then determine the expected gesture

// by analyzing if the sensors' values are increasing or decreasing

if((increasing\_TR && !increasing\_TL) || (increasing\_BR && !increasing\_BL) || (increasing\_R && !increasing\_L)){ // Left to right movement

expectedGesture = 1; // Expected an "east" gesture

}

else if((!increasing\_TR && increasing\_TL) || (!increasing\_BR && increasing\_BL) || (!increasing\_R && increasing\_L)){ // Right to left movement

expectedGesture = 2; // Expected a "west" gesture

}

else if(increasing\_TR && increasing\_TL && !increasing\_BL && !increasing\_BR){ // Top to bottom movement

expectedGesture = 3; // Expected a "south" gesture

}

else if(increasing\_BR && increasing\_BL && !increasing\_TL && !increasing\_TR){ // Bottom to top movement

expectedGesture = 4; // Expected a "north" gesture

}

// Check if the current "distance\_xx" minus the 2nd element in the array is greater than 50 then it is a "down" movement.

// The "increasing\_xx" vars are not used because they are very precise and they didn´t detect the "down" movement most of the time.

else if(distance\_TL - distanceArray\_TL[2] > 50 && distance\_TR - distanceArray\_TR[2] > 50 && distance\_BL - distanceArray\_BL[2] > 50 && distance\_BR - distanceArray\_BR[2] > 50){ // Up to down movementd

expectedGesture = 5; // Expected a "down" gesture

}

case 1:

// If "expectedGesture" is "east" then check if TR and BR sensors are increasing and TL and BL are decreasing.

// If this is true then save the current distance to the var "lastDistanceHold" to use it later to check a "hold

// gesture" action. Also, reset the "expectedGesture" var and set the "gesture" var to 1 ("east")

if((increasing\_TR && !increasing\_TL) || (increasing\_BR && !increasing\_BL) || (increasing\_R && !increasing\_L)){

lastDistanceHold\_TR = distance\_TR;

lastDistanceHold\_BR = distance\_BR;

lastDistanceHold\_TL = distance\_TL;

lastDistanceHold\_BL = distance\_BL;

lastDistanceHold\_R = distance\_R;

lastDistanceHold\_L = distance\_L;

if(readGesture){

Serial.println("EAST");

gesture = 1;

holdMillis = 300; // Change it to 300ms to add a small delay between the gesture "east" and the "hold gesture" action

lastHoldMillis = millis();

command[0]=1;

Serial.write(command,2);

}

expectedGesture = 0;

increasing\_TL = false;

decreasing\_TL = false;

increasing\_TR = false;

decreasing\_TR = false;

increasing\_BL = false;

decreasing\_BL = false;

increasing\_BR = false;

decreasing\_BR = false;

increasing\_R = false;

decreasing\_R = false;

increasing\_L = false;

decreasing\_L = false;

readGesture = false;

}

case 2:

if((!increasing\_TR && increasing\_TL) || (!increasing\_BR && increasing\_BL) || (!increasing\_R && increasing\_L)){

if(readGesture){

Serial.println("WEST");

gesture = 2;

holdMillis = 300;

lastHoldMillis = millis();

command[1]=1;

Serial.write(command,2);

}

expectedGesture = 0;

lastDistanceHold\_TR = distance\_TR;

lastDistanceHold\_BR = distance\_BR;

lastDistanceHold\_TL = distance\_TL;

lastDistanceHold\_BL = distance\_BL;

lastDistanceHold\_R = distance\_R;

lastDistanceHold\_L = distance\_L;

increasing\_TL = false;

decreasing\_TL = false;

increasing\_TR = false;

decreasing\_TR = false;

increasing\_BL = false;

decreasing\_BL = false;

increasing\_BR = false;

decreasing\_BR = false;

increasing\_R = false;

decreasing\_R = false;

increasing\_L = false;

decreasing\_L = false;

readGesture = false;

}

break;

/\* case 3:

if(increasing\_BR && increasing\_BL && decreasing\_TL && decreasing\_TR){

if(readGesture){

Serial.println("SOUTH");

gesture = 3;

holdMillis = 300;

lastHoldMillis = millis();

}

expectedGesture = 0;

lastDistanceHold\_TR = distance\_TR;

lastDistanceHold\_BR = distance\_BR;

lastDistanceHold\_TL = distance\_TL;

lastDistanceHold\_BL = distance\_BL;

increasing\_TL = false;

decreasing\_TL = false;

increasing\_TR = false;

decreasing\_TR = false;

increasing\_BL = false;

decreasing\_BL = false;

increasing\_BR = false;

decreasing\_BR = false;

readGesture = false;

}

break;

case 4:

if(increasing\_TR && increasing\_TL && decreasing\_BL && decreasing\_BR){

if(readGesture){

Serial.println("NORTH");

gesture = 4;

holdMillis = 300;

lastHoldMillis = millis();

}

expectedGesture = 0;

lastDistanceHold\_TR = distance\_TR;

lastDistanceHold\_BR = distance\_BR;

lastDistanceHold\_TL = distance\_TL;

lastDistanceHold\_BL = distance\_BL;

increasing\_TL = false;

decreasing\_TL = false;

increasing\_TR = false;

decreasing\_TR = false;

increasing\_BL = false;

decreasing\_BL = false;

increasing\_BR = false;

decreasing\_BR = false;

readGesture = false;

}

break;

case 5:

// The next if statement checks if 3 sensors are increasing its value and if it is true then store a "down" gesture

if((increasing\_BR && increasing\_BL && increasing\_TL) || (increasing\_TR && increasing\_BL && increasing\_TL) || (increasing\_TR && increasing\_BR && increasing\_TL) || (increasing\_TR && increasing\_BR && increasing\_BL)){

if(readGesture){

Serial.println("DOWN");

gesture = 5;

holdMillis = 300;

lastHoldMillis = millis();

}

expectedGesture = 0;

increasing\_TL = false;

decreasing\_TL = false;

increasing\_TR = false;

decreasing\_TR = false;

increasing\_BL = false;

decreasing\_BL = false;

increasing\_BR = false;

decreasing\_BR = false;

readGesture = false;

}

break;\*/

}

}

}

void processHoldGesture(){

// If the actual "distance\_xx" minus the "lastDistanceHold\_xx" (saved when the gesture was detected)

// is within a range (20) then make a "hold gesture" action

if(-20 <= distance\_TR - lastDistanceHold\_TR && distance\_TR - lastDistanceHold\_TR <= 20 && -20 <= distance\_TL - lastDistanceHold\_TL && distance\_TL - lastDistanceHold\_TL <= 20 && -20 <= distance\_BR - lastDistanceHold\_BR && distance\_BR - lastDistanceHold\_BR <= 20 && -20 <= distance\_BL - lastDistanceHold\_BL && distance\_BL - lastDistanceHold\_BL <= 20 && -20 <= distance\_R - lastDistanceHold\_R && distance\_R - lastDistanceHold\_R <= 20 && -20 <= distance\_L - lastDistanceHold\_L && distance\_L - lastDistanceHold\_L <= 20){

if(distance\_TR != 0 || distance\_TL != 0 || distance\_BR != 0 || distance\_BL != 0 || distance\_L != 0 || distance\_R != 0 ){

switch(gesture){

case 1:

if(millis() - lastHoldMillis > holdMillis){

holdMillis = 100;

Serial.println("HOLD EAST");

lastHoldMillis = millis();

}

break;

case 2:

if(millis() - lastHoldMillis > holdMillis){

holdMillis = 100;

Serial.println("HOLD WEST");

lastHoldMillis = millis();

}

break;

case 3:

if(millis() - lastHoldMillis > holdMillis){

holdMillis = 100;

Serial.println("HOLD SOUTH");

lastHoldMillis = millis();

}

break;

case 4:

if(millis() - lastHoldMillis > holdMillis){

holdMillis = 100;

Serial.println("HOLD NORTH");

lastHoldMillis = millis();

}

break;

}

}

else{

// Set a high numbenr for the "lastDistanceHold" vars to stop reading a "hold gesture" action

lastDistanceHold\_TL = 20000;

lastDistanceHold\_TR = 20000;

lastDistanceHold\_BR = 20000;

lastDistanceHold\_BL = 20000;

}

}

else{

lastDistanceHold\_TL = 20000;

lastDistanceHold\_TR = 20000;

lastDistanceHold\_BL = 20000;

lastDistanceHold\_BR = 20000;

lastDistanceHold\_R = 20000;

lastDistanceHold\_L = 20000;

}

}

* Arduino Uno 2(receiver):

#define IR\_RX\_TL A0 // Top Left IR receiver photodiode on analog pin A0

#define IR\_RX\_TR A1 // Top Right IR receiver photodiode on analog pin A1

#define IR\_RX\_BL A2 // IR receiver photodiode on analog pin A2

#define IR\_RX\_BR A3 // IR receiver photodiode on analog pin A3

#define IR\_RX\_L A4

#define IR\_RX\_R A5

#define IR\_TX\_TL 2 // Top Left IR emitter LED on digital pin 2

#define IR\_TX\_TR 3 // Top Right IR emitter LED on digital pin 3

#define IR\_TX\_BL 4 // Bottom Left IR emitter LED on digital pin 4

#define IR\_TX\_BR 5 // Bottom Right IR emitter LED on digital pin 5

#define IR\_TX\_L 6

#define IR\_TX\_R 7

bool readGesture = true; // Used to process the gesture

int ambientIR\_TL; // Stores the ambient IR value read from the TL IR receiver

int ambientIR\_TR; // Stores the ambient IR value read from the TR IR receiver

int ambientIR\_BL; // Stores the ambient IR value read from the BL IR receiver

int ambientIR\_BR; // Stores the ambient IR value read from the BR IR receiver

int ambientIR\_L;

int ambientIR\_R;

int obstacleIR\_TL; // Stores the raw object IR value read from the TL IR receiver

int obstacleIR\_TR; // Stores the raw object IR value read from the TR IR receiver

int obstacleIR\_BL; // Stores the raw object IR value read from the BL IR receiver

int obstacleIR\_BR; // Stores the raw object IR value read from the BR IR receiver

int obstacleIR\_L;

int obstacleIR\_R;

int value\_TL[10]; // Stores the IR value read from the TL IR receiver

int value\_TR[10]; // Stores the IR value read from the TR IR receiver

int value\_BL[10]; // Stores the IR value read from the BL IR receiver

int value\_BR[10]; // Stores the IR value read from the BR IR receiver

int value\_L[10];

int value\_R[10];

int distance\_TL; // Stores the mapped IR value read from the TL IR receiver

int distance\_TR; // Stores the mapped IR value read from the TR IR receiver

int distance\_BL; // Stores the mapped IR value read from the BL IR receiver

int distance\_BR; // Stores the mapped IR value read from the BR IR receiver

int distance\_L;

int distance\_R;

int calibration\_TL; // Stores the raw IR value read from the TL receiver when the Arduino is powered

int calibration\_TR; // Stores the raw IR value read from the TR receiver when the Arduino is powered

int calibration\_BL; // Stores the raw IR value read from the BL receiver when the Arduino is powered

int calibration\_BR; // Stores the raw IR value read from the BR receiver when the Arduino is powered

int calibration\_L;

int calibration\_R;

int lastDistanceHold\_TL; // The next variables are used to detect a "hold gesture" action

int lastDistanceHold\_TR;

int lastDistanceHold\_BL;

int lastDistanceHold\_BR;

int lastDistanceHold\_L;

int lastDistanceHold\_R;

int holdMillis = 300; // Used in a timer to determine a "hold gesture" action

uint8\_t distanceArray\_TL[100]; // Stores the last 100 IR values from the TL IR receiver to determine the gesture later

uint8\_t distanceArray\_TR[100]; // Stores the last 100 IR values from the TR IR receiver to determine the gesture later

uint8\_t distanceArray\_BL[100]; // Stores the last 100 IR values from the BL IR receiver to determine the gesture later

uint8\_t distanceArray\_BR[100]; // Stores the last 100 IR values from the BR IR receiver to determine the gesture

uint8\_t distanceArray\_L[100];

uint8\_t distanceArray\_R[100];

uint8\_t distanceArrayCounter = 0; // Used to access the previous array's content

uint8\_t gesture = 0; // Stores the gesture

uint32\_t lastHoldMillis = 0;

void setup() {

// put your setup code here, to run once:

Serial.begin(9600);

pinMode(IR\_TX\_TL, OUTPUT);

pinMode(IR\_TX\_TR, OUTPUT);

pinMode(IR\_TX\_BL, OUTPUT);

pinMode(IR\_TX\_BR, OUTPUT);

pinMode(IR\_TX\_L, OUTPUT);

pinMode(IR\_TX\_R, OUTPUT);

digitalWrite(IR\_TX\_TL, LOW);

digitalWrite(IR\_TX\_TR, LOW);

digitalWrite(IR\_TX\_BL, LOW);

digitalWrite(IR\_TX\_BR, LOW);

digitalWrite(IR\_TX\_L, LOW);

digitalWrite(IR\_TX\_R, LOW);

calibrateIR();

}

void loop() {

// put your main code here, to run repeatedly:

readIR(5); // Read 5 times the IR values and store the average

determineGesture(); // It determines the type of gesture made

// processHoldGesture(); // It checks if the hand is being hold

}

void calibrateIR(){

for(uint8\_t x = 0; x < 5; x++){

digitalWrite(IR\_TX\_TL,LOW);

digitalWrite(IR\_TX\_TR,LOW);

digitalWrite(IR\_TX\_BL,LOW);

digitalWrite(IR\_TX\_BR,LOW);

digitalWrite(IR\_TX\_L,LOW);

digitalWrite(IR\_TX\_R,LOW);

delay(2);

ambientIR\_TL = analogRead(IR\_RX\_TL);

ambientIR\_TR = analogRead(IR\_RX\_TR);

ambientIR\_BL = analogRead(IR\_RX\_BL);

ambientIR\_BR = analogRead(IR\_RX\_BR);

ambientIR\_L = analogRead(IR\_RX\_L);

ambientIR\_R = analogRead(IR\_RX\_R);

digitalWrite(IR\_TX\_TL,HIGH);

digitalWrite(IR\_TX\_TR,HIGH);

digitalWrite(IR\_TX\_BL,HIGH);

digitalWrite(IR\_TX\_BR,HIGH);

digitalWrite(IR\_TX\_L,HIGH);

digitalWrite(IR\_TX\_R,HIGH);

delay(1);

obstacleIR\_TL = analogRead(IR\_RX\_TL);

obstacleIR\_TR = analogRead(IR\_RX\_TR);

obstacleIR\_BL = analogRead(IR\_RX\_BL);

obstacleIR\_BR = analogRead(IR\_RX\_BR);

obstacleIR\_L = analogRead(IR\_RX\_L);

obstacleIR\_R = analogRead(IR\_RX\_R);

value\_TL[x] = - ambientIR\_TL + obstacleIR\_TL;

value\_TR[x] = - ambientIR\_TR + obstacleIR\_TR;

value\_BL[x] = - ambientIR\_BL + obstacleIR\_BL;

value\_BR[x] = - ambientIR\_BR + obstacleIR\_BR;

value\_L[x] = - ambientIR\_L + obstacleIR\_L;

value\_R[x] = - ambientIR\_R + obstacleIR\_R;

}

for(uint8\_t x = 0; x < 5; x++){

distance\_TL += value\_TL[x];

distance\_TR += value\_TR[x];

distance\_BL += value\_BL[x];

distance\_BR += value\_BR[x];

distance\_L += value\_L[x];

distance\_R += value\_R[x];

}

calibration\_TL = (distance\_TL / 5);

calibration\_TR = (distance\_TR / 5);

calibration\_BL = (distance\_BL / 5);

calibration\_BR = (distance\_BR / 5);

calibration\_L = (distance\_L / 5);

calibration\_R = (distance\_R / 5);

}

void readIR(uint8\_t times){

for(uint8\_t x = 0; x < times; x++){

digitalWrite(IR\_TX\_TL,LOW); // We need to turn off the IR LEDs to read the ambient IR

digitalWrite(IR\_TX\_TR,LOW);

digitalWrite(IR\_TX\_BL,LOW);

digitalWrite(IR\_TX\_BR,LOW);

digitalWrite(IR\_TX\_L,LOW);

digitalWrite(IR\_TX\_R,LOW);

delay(2); // Wait until the IR LEDs are completely turned off

ambientIR\_TL = analogRead(IR\_RX\_TL); // Stores the ambient IR light

ambientIR\_TR = analogRead(IR\_RX\_TR);

ambientIR\_BL = analogRead(IR\_RX\_BL);

ambientIR\_BR = analogRead(IR\_RX\_BR);

ambientIR\_L = analogRead(IR\_RX\_L);

ambientIR\_R = analogRead(IR\_RX\_R);

digitalWrite(IR\_TX\_TL,HIGH); // Turn on the IR LEDs to read the IR light reflected by the obstacle

digitalWrite(IR\_TX\_TR,HIGH);

digitalWrite(IR\_TX\_BL,HIGH);

digitalWrite(IR\_TX\_BR,HIGH);

digitalWrite(IR\_TX\_L,HIGH);

digitalWrite(IR\_TX\_R,HIGH);

delay(1); // Wait until the IR LEDs are completely turned on

obstacleIR\_TL = analogRead(IR\_RX\_TL); // Stores the IR light reflected by the obstacle

obstacleIR\_TR = analogRead(IR\_RX\_TR);

obstacleIR\_BL = analogRead(IR\_RX\_BL);

obstacleIR\_BR = analogRead(IR\_RX\_BR);

obstacleIR\_L = analogRead(IR\_RX\_L);

obstacleIR\_R = analogRead(IR\_RX\_R);

value\_TL[x] = - ambientIR\_TL + obstacleIR\_TL; // Store the IR value to later calculate the average

value\_TR[x] = - ambientIR\_TR + obstacleIR\_TR;

value\_BL[x] = - ambientIR\_BL + obstacleIR\_BL;

value\_BR[x] = - ambientIR\_BR + obstacleIR\_BR;

value\_L[x] = - ambientIR\_L + obstacleIR\_L;

value\_R[x] = - ambientIR\_R + obstacleIR\_R;

}

for(uint8\_t x = 0; x < times; x++){ // Calculate the average IR value per sensor

distance\_TL += value\_TL[x];

distance\_TR += value\_TR[x];

distance\_BL += value\_BL[x];

distance\_BR += value\_BR[x];

distance\_L += value\_L[x];

distance\_R += value\_R[x];

}

distance\_TL = (distance\_TL / times); // Save the distance value

distance\_TR = (distance\_TR / times);

distance\_BL = (distance\_BL / times);

distance\_BR = (distance\_BR / times);

distance\_L = (distance\_L / times);

distance\_R = (distance\_R / times);

if(calibration\_TL + 50 > distance\_TL){ // Calibrate the sensors

distance\_TL = 0;

}

else{

distance\_TL -= calibration\_TL + 50;

}

if(calibration\_TR + 50 > distance\_TR){

distance\_TR = 0;

}

else{

distance\_TR -= calibration\_TR + 50;

}

if(calibration\_BL + 50 > distance\_BL){

distance\_BL = 0;

}

else{

distance\_BL -= calibration\_BL + 50;

}

if(calibration\_BR + 50 > distance\_BR){

distance\_BR = 0;

}

else{

distance\_BR -= calibration\_BR + 50;

}

if(calibration\_L + 50 > distance\_L){

distance\_L = 0;

}

else{

distance\_L -= calibration\_L + 50;

}

if(calibration\_R + 50 > distance\_R){

distance\_R = 0;

}

else{

distance\_R -= calibration\_R + 50;

}

// If any distance is greater than 0 then map it from a 0-1023 range to a 0-255 range (to be able to store it in a 1B variable) and store it in an array

if(distance\_TL != 0 || distance\_TR != 0 || distance\_BL != 0 || distance\_BR != 0 || distance\_L != 0 || distance\_R != 0){

distance\_TR = map(constrain(distance\_TR, 0, 1100), 0, 1100, 0, 255);

distance\_TL = map(constrain(distance\_TL, 0, 1100), 0, 1100, 0, 255);

distance\_BR = map(constrain(distance\_BR, 0, 1100), 0, 1100, 0, 255);

distance\_BL = map(constrain(distance\_BL, 0, 1100), 0, 1100, 0, 255);

distance\_L = map(constrain(distance\_L, 0, 1100), 0, 1100, 0, 255);

distance\_R = map(constrain(distance\_R, 0, 1100), 0, 1100, 0, 255);

if(distanceArrayCounter < 100){

distanceArray\_TR[distanceArrayCounter] = distance\_TR;

distanceArray\_TL[distanceArrayCounter] = distance\_TL;

distanceArray\_BR[distanceArrayCounter] = distance\_BR;

distanceArray\_BL[distanceArrayCounter] = distance\_BL;

distanceArray\_R[distanceArrayCounter] = distance\_R;

distanceArray\_L[distanceArrayCounter] = distance\_L;

distanceArrayCounter += 1;

}

else{

distanceArray\_TR[0] = distance\_TR;

distanceArray\_TL[0] = distance\_TL;

distanceArray\_BR[0] = distance\_BR;

distanceArray\_BL[0] = distance\_BL;

distanceArray\_L[0] = distance\_L;

distanceArray\_R[0] = distance\_R;

distanceArrayCounter = 1;

}

}

else if(distance\_TL == 0 && distance\_TR == 0 && distance\_BL == 0 && distance\_BR == 0 && distance\_L == 0 && distance\_R == 0){ // If not then reset the arrays

readGesture = true;

distanceArrayCounter = 0;

distanceArray\_TR[0] = 0;

distanceArray\_TL[0] = 0;

distanceArray\_BR[0] = 0;

distanceArray\_BL[0] = 0;

distanceArray\_R[0] = 0;

distanceArray\_L[0] = 0;

}

}

void determineGesture(){

if(distanceArrayCounter > 2){ // If the arrays have more than 2 elements then determine the gesture

uint8\_t distancePost\_TL = 0; // The next "distancePost..." vars are the sum of it previous value plus

uint8\_t distancePost\_TR = 0; // the last element minus the previous element of the array

uint8\_t distancePost\_BL = 0;

uint8\_t distancePost\_BR = 0;

uint8\_t distancePost\_L = 0;

uint8\_t distancePost\_R = 0;

uint8\_t distancePostNeg\_TL = 0;

uint8\_t distancePostNeg\_TR = 0;

uint8\_t distancePostNeg\_BL = 0;

uint8\_t distancePostNeg\_BR = 0;

uint8\_t distancePostNeg\_R = 0;

uint8\_t distancePostNeg\_L = 0;

uint8\_t expectedGesture = 0; // Used later to determine the gesture made

bool increasing\_TL = false; // Used later to determine the gesture made. If the IR sensor's value

bool decreasing\_TL = false; // increases then "increasing\_xx" is true. Otherwise, if it decreasing

bool increasing\_TR = false; // then "decreasing\_xx" is true

bool decreasing\_TR = false;

bool increasing\_BL = false;

bool decreasing\_BL = false;

bool increasing\_BR = false;

bool decreasing\_BR = false;

bool increasing\_L = false;

bool decreasing\_L = false;

bool increasing\_R = false;

bool decreasing\_R = false;

// For every element in the distanceArray...

for(uint8\_t x = 1; x < distanceArrayCounter; x++){

// Update the "distancePost..." vars values

if(distanceArray\_TL[x] - distanceArray\_TL[x-1] > 0){

distancePost\_TL += distanceArray\_TL[x] - distanceArray\_TL[x-1];

}

else{

distancePostNeg\_TL += distanceArray\_TL[x-1] - distanceArray\_TL[x];

}

if(distanceArray\_TR[x] - distanceArray\_TR[x-1] > 0){

distancePost\_TR += distanceArray\_TR[x] - distanceArray\_TR[x-1];

}

else{

distancePostNeg\_TR += distanceArray\_TR[x-1] - distanceArray\_TR[x];

}

if(distanceArray\_BL[x] - distanceArray\_BL[x-1] > 0){

distancePost\_BL += distanceArray\_BL[x] - distanceArray\_BL[x-1];

}

else{

distancePostNeg\_BL += distanceArray\_BL[x-1] - distanceArray\_BL[x];

}

if(distanceArray\_BR[x] - distanceArray\_BR[x-1] > 0){

distancePost\_BR += distanceArray\_BR[x] - distanceArray\_BR[x-1];

}

else{

distancePostNeg\_BR += distanceArray\_BR[x-1] - distanceArray\_BR[x];

}

if(distanceArray\_L[x] - distanceArray\_L[x-1] > 0){

distancePost\_L += distanceArray\_L[x] - distanceArray\_L[x-1];

}

else{

distancePostNeg\_L += distanceArray\_L[x-1] - distanceArray\_L[x];

}

if(distanceArray\_R[x] - distanceArray\_R[x-1] > 0){

distancePost\_R += distanceArray\_R[x] - distanceArray\_R[x-1];

}

else{

distancePostNeg\_R += distanceArray\_R[x-1] - distanceArray\_R[x];

}

}

// Based on the "distancePost..." vars determine if the value is increasing or decreasing

if(distancePost\_TL > 10 && distancePostNeg\_TL < 10){

increasing\_TL = true;

decreasing\_TL = false;

}

else if(distancePostNeg\_TL > 10){

increasing\_TL = false;

decreasing\_TL = true;

}

if(distancePost\_TR > 10 && distancePostNeg\_TR < 10){

increasing\_TR = true;

decreasing\_TR = false;

}

else if(distancePostNeg\_TR > 10){

increasing\_TR = false;

decreasing\_TR = true;

}

if(distancePost\_BL > 10 && distancePostNeg\_BL < 10){

increasing\_BL = true;

decreasing\_BL = false;

}

else if(distancePostNeg\_BL > 10){

increasing\_BL = false;

decreasing\_BL = true;

}

if(distancePost\_BR > 10 && distancePostNeg\_BR < 10){

increasing\_BR = true;

decreasing\_BR = false;

}

else if(distancePostNeg\_BR > 10){

increasing\_BR = false;

decreasing\_BR = true;

}

if(distancePost\_L > 10 && distancePostNeg\_L < 10){

increasing\_L = true;

decreasing\_L = false;

}

else if(distancePostNeg\_L > 10){

increasing\_L = false;

decreasing\_L = true;

}

if(distancePost\_R > 10 && distancePostNeg\_R < 10){

increasing\_BR = true;

decreasing\_BR = false;

}

else if(distancePostNeg\_R > 10){

increasing\_R = false;

decreasing\_R = true;

}

// Determine the type of gesture //for present south and north gestures have not been updated

switch(expectedGesture){

case 0: // If "expectedGesture" is 0 then determine the expected gesture

// by analyzing if the sensors' values are increasing or decreasing

if((increasing\_TR && !increasing\_TL) || (increasing\_BR && !increasing\_BL) || (increasing\_R && !increasing\_L)){ // Left to right movement

expectedGesture = 1; // Expected an "east" gesture

}

else if((!increasing\_TR && increasing\_TL) || (!increasing\_BR && increasing\_BL) || (!increasing\_R && increasing\_L)){ // Right to left movement

expectedGesture = 2; // Expected a "west" gesture

}

else if(increasing\_TR && increasing\_TL && !increasing\_BL && !increasing\_BR){ // Top to bottom movement

expectedGesture = 3; // Expected a "south" gesture

}

else if(increasing\_BR && increasing\_BL && !increasing\_TL && !increasing\_TR){ // Bottom to top movement

expectedGesture = 4; // Expected a "north" gesture

}

// Check if the current "distance\_xx" minus the 2nd element in the array is greater than 50 then it is a "down" movement.

// The "increasing\_xx" vars are not used because they are very precise and they didn´t detect the "down" movement most of the time.

else if(distance\_TL - distanceArray\_TL[2] > 50 && distance\_TR - distanceArray\_TR[2] > 50 && distance\_BL - distanceArray\_BL[2] > 50 && distance\_BR - distanceArray\_BR[2] > 50){ // Up to down movementd

expectedGesture = 5; // Expected a "down" gesture

}

case 1:

// If "expectedGesture" is "east" then check if TR and BR sensors are increasing and TL and BL are decreasing.

// If this is true then save the current distance to the var "lastDistanceHold" to use it later to check a "hold

// gesture" action. Also, reset the "expectedGesture" var and set the "gesture" var to 1 ("east")

if((increasing\_TR && !increasing\_TL) || (increasing\_BR && !increasing\_BL) || (increasing\_R && !increasing\_L)){

lastDistanceHold\_TR = distance\_TR;

lastDistanceHold\_BR = distance\_BR;

lastDistanceHold\_TL = distance\_TL;

lastDistanceHold\_BL = distance\_BL;

lastDistanceHold\_R = distance\_R;

lastDistanceHold\_L = distance\_L;

if(readGesture){

Serial.println("EAST");

gesture = 1;

holdMillis = 300; // Change it to 300ms to add a small delay between the gesture "east" and the "hold gesture" action

lastHoldMillis = millis();

Serial.readBytes(command,2);

if (command[0] == '1' && command[1] == '0'){

Serial.println("ROTATE RIGHT");

}

else if (command[0] == '0' && command[0] == '1'){

Serial.println("ZOOM OUT");

}

}

expectedGesture = 0;

increasing\_TL = false;

decreasing\_TL = false;

increasing\_TR = false;

decreasing\_TR = false;

increasing\_BL = false;

decreasing\_BL = false;

increasing\_BR = false;

decreasing\_BR = false;

increasing\_R = false;

decreasing\_R = false;

increasing\_L = false;

decreasing\_L = false;

readGesture = false;

}

case 2:

if((!increasing\_TR && increasing\_TL) || (!increasing\_BR && increasing\_BL) || (!increasing\_R && increasing\_L)){

if(readGesture){

Serial.println("WEST");

gesture = 2;

holdMillis = 300;

lastHoldMillis = millis();

Serial.readBytes(command,2);

if (command[0] == '1' && command[1] == '0'){

Serial.println("ZOOM IN");

}

else if (command[0] == '0' && command[1] == '1'){

Serial.println("ROTATE LEFT");

}

}

expectedGesture = 0;

lastDistanceHold\_TR = distance\_TR;

lastDistanceHold\_BR = distance\_BR;

lastDistanceHold\_TL = distance\_TL;

lastDistanceHold\_BL = distance\_BL;

lastDistanceHold\_R = distance\_R;

lastDistanceHold\_L = distance\_L;

increasing\_TL = false;

decreasing\_TL = false;

increasing\_TR = false;

decreasing\_TR = false;

increasing\_BL = false;

decreasing\_BL = false;

increasing\_BR = false;

decreasing\_BR = false;

increasing\_R = false;

decreasing\_R = false;

increasing\_L = false;

decreasing\_L = false;

readGesture = false;

}

break;

/\* case 3:

if(increasing\_BR && increasing\_BL && decreasing\_TL && decreasing\_TR){

if(readGesture){

Serial.println("SOUTH");

gesture = 3;

holdMillis = 300;

lastHoldMillis = millis();

}

expectedGesture = 0;

lastDistanceHold\_TR = distance\_TR;

lastDistanceHold\_BR = distance\_BR;

lastDistanceHold\_TL = distance\_TL;

lastDistanceHold\_BL = distance\_BL;

increasing\_TL = false;

decreasing\_TL = false;

increasing\_TR = false;

decreasing\_TR = false;

increasing\_BL = false;

decreasing\_BL = false;

increasing\_BR = false;

decreasing\_BR = false;

readGesture = false;

}

break;

case 4:

if(increasing\_TR && increasing\_TL && decreasing\_BL && decreasing\_BR){

if(readGesture){

Serial.println("NORTH");

gesture = 4;

holdMillis = 300;

lastHoldMillis = millis();

}

expectedGesture = 0;

lastDistanceHold\_TR = distance\_TR;

lastDistanceHold\_BR = distance\_BR;

lastDistanceHold\_TL = distance\_TL;

lastDistanceHold\_BL = distance\_BL;

increasing\_TL = false;

decreasing\_TL = false;

increasing\_TR = false;

decreasing\_TR = false;

increasing\_BL = false;

decreasing\_BL = false;

increasing\_BR = false;

decreasing\_BR = false;

readGesture = false;

}

break;

case 5:

// The next if statement checks if 3 sensors are increasing its value and if it is true then store a "down" gesture

if((increasing\_BR && increasing\_BL && increasing\_TL) || (increasing\_TR && increasing\_BL && increasing\_TL) || (increasing\_TR && increasing\_BR && increasing\_TL) || (increasing\_TR && increasing\_BR && increasing\_BL)){

if(readGesture){

Serial.println("DOWN");

gesture = 5;

holdMillis = 300;

lastHoldMillis = millis();

}

expectedGesture = 0;

increasing\_TL = false;

decreasing\_TL = false;

increasing\_TR = false;

decreasing\_TR = false;

increasing\_BL = false;

decreasing\_BL = false;

increasing\_BR = false;

decreasing\_BR = false;

readGesture = false;

}

break;\*/

}

}

}

void processHoldGesture(){

// If the actual "distance\_xx" minus the "lastDistanceHold\_xx" (saved when the gesture was detected)

// is within a range (20) then make a "hold gesture" action

if(-20 <= distance\_TR - lastDistanceHold\_TR && distance\_TR - lastDistanceHold\_TR <= 20 && -20 <= distance\_TL - lastDistanceHold\_TL && distance\_TL - lastDistanceHold\_TL <= 20 && -20 <= distance\_BR - lastDistanceHold\_BR && distance\_BR - lastDistanceHold\_BR <= 20 && -20 <= distance\_BL - lastDistanceHold\_BL && distance\_BL - lastDistanceHold\_BL <= 20 && -20 <= distance\_R - lastDistanceHold\_R && distance\_R - lastDistanceHold\_R <= 20 && -20 <= distance\_L - lastDistanceHold\_L && distance\_L - lastDistanceHold\_L <= 20){

if(distance\_TR != 0 || distance\_TL != 0 || distance\_BR != 0 || distance\_BL != 0 || distance\_L != 0 || distance\_R != 0 ){

switch(gesture){

case 1:

if(millis() - lastHoldMillis > holdMillis){

holdMillis = 100;

Serial.println("HOLD EAST");

lastHoldMillis = millis();

}

break;

case 2:

if(millis() - lastHoldMillis > holdMillis){

holdMillis = 100;

Serial.println("HOLD WEST");

lastHoldMillis = millis();

}

break;

case 3:

if(millis() - lastHoldMillis > holdMillis){

holdMillis = 100;

Serial.println("HOLD SOUTH");

lastHoldMillis = millis();

}

break;

case 4:

if(millis() - lastHoldMillis > holdMillis){

holdMillis = 100;

Serial.println("HOLD NORTH");

lastHoldMillis = millis();

}

break;

}

}

else{

// Set a high numbenr for the "lastDistanceHold" vars to stop reading a "hold gesture" action

lastDistanceHold\_TL = 20000;

lastDistanceHold\_TR = 20000;

lastDistanceHold\_BR = 20000;

lastDistanceHold\_BL = 20000;

}

}

else{

lastDistanceHold\_TL = 20000;

lastDistanceHold\_TR = 20000;

lastDistanceHold\_BL = 20000;

lastDistanceHold\_BR = 20000;

lastDistanceHold\_R = 20000;

lastDistanceHold\_L = 20000;

}

}

* After Anshul completed the code, he, Anwesh and Gaurav firstly insulated some connections which weren’t done. Firstly, the sensor board was checked for a 2x2 matrix wherein we printed the output on serial monitor and thereby decided to check. Though it worked fine for some cases, sometimes the leds wouldn’t glow. Then we checked it for 3x2 matrix. There were many problems like wrong , erroneous output used to be printed onto the serial monitor, sometimes the Arduino wouldn’t work because Vcc and ground were shorted indicating improper insulation, leds would glow too much bright indicating it was directly connected to Vcc. So, Gaurav decided to debug and check the entire circuit of the board and Anshul decided to again recheck his code and also figure out how to communicate between the two Arduinos.

**WEEK 5:**

* Gaurav began testing the sensor board. Firstly he checked whether each set was working using a simple code on Arduino. Sometimes the ir sensors worked properly (which was checked through the analog values printed on the serial monitor and through mobile camera wherein the ir led would seem emitting light of violet colour), but the leds wouldn’t glow. Most of the times it was due to open circuit. The audible continuity in multimeter helped a lot to determine short and open connections. Most of the times, instead of re-soldering incase of open circuit, he would crimp the wires and insulate them properly. Finally after 2 days of testing, all sets of sensors managed to work. Those which needed to be soldered were done in tinkerer’s lab.
* Anshul: As the above method of communication between the two was not working we decided to use arduino mega .For this to work the two separate codes were merged into one single code with the same functions as above but all of them were present twice(one for each set of six photodiodes). One more gesture was added which was to change the model which was being projected . This was done by removing the bottom most layer of photodiodes from the main code and rewriting them as switches for changing the model(when the intensity recorded by the photodiode was more than a threshold intensity the model was changed according to the number of photodiode).The final code is as below:-

#define IR\_RX\_TL2 A3 // Top Left IR receiver photodiode on analog pin A0

#define IR\_RX\_TR2 A15 // Top Right IR receiver photodiode on analog pin A1

#define IR\_RX\_L2 A5

#define IR\_RX\_R2 A4

#define IR\_TX\_TL2 9 // Top Left IR emitter LED on digital pin 2

#define IR\_TX\_TR2 10 // Top Right IR emitter LED on digital pin 3

#define IR\_TX\_L2 7

#define IR\_TX\_R2 8

bool readGesture2 = true; // Used to process the gesture2

int a=0;

int b=0;

int ambientIR\_TL2; // Stores the ambient IR value read from the TL2 IR receiver

int ambientIR\_TR2; // Stores the ambient IR value read from the TR2 IR receiver

int ambientIR\_L2;

int ambientIR\_R2;

int obstacleIR\_TL2; // Stores the raw object IR value read from the TL2 IR receiver

int obstacleIR\_TR2; // Stores the raw object IR value read from the TR2 IR receiver

int obstacleIR\_L2;

int obstacleIR\_R2;

int value\_TL2[10]; // Stores the IR value read from the TL2 IR receiver

int value\_TR2[10]; // Stores the IR value read from the TR2 IR receiver

int value\_L2[10];

int value\_R2[10];

int distance\_TL2; // Stores the mapped IR value read from the TL2 IR receiver

int distance\_TR2; // Stores the mapped IR value read from the TR2 IR receiver

int distance\_L2;

int distance\_R2;

int caliBR2ation\_TL2; // Stores the raw IR value read from the TL2 receiver when the Arduino is powered

int caliBR2ation\_TR2; // Stores the raw IR value read from the TR2 receiver when the Arduino is powered

int caliBR2ation\_L2;

int caliBR2ation\_R2;

int lastDistanceHold\_TL2; // The next variaBL2es are used to detect a "hold gesture2" action

int lastDistanceHold\_TR2;

int lastDistanceHold\_L2;

int lastDistanceHold\_R2;

int holdMillis2 = 300; // Used in a timer to determine a "hold gesture2" action

uint8\_t distanceArray\_TL2[100]; // Stores the last 100 IR values from the TL2 IR receiver to determine the gesture2 later

uint8\_t distanceArray\_TR2[100]; // Stores the last 100 IR values from the TR2 IR receiver to determine the gesture2 later

uint8\_t distanceArray\_L2[100];

uint8\_t distanceArray\_R2[100];

uint8\_t distanceArrayCounter2 = 0; // Used to access the previous array's content

uint8\_t gesture2 = 0; // Stores the gesture2

uint32\_t lastHoldMillis2 = 0;

#define IR\_RX\_TL A9 // Top Left IR receiver photodiode on analog pin A0

#define IR\_RX\_TR A8 // Top Right IR receiver photodiode on analog pin A1

#define IR\_RX\_L A11

#define IR\_RX\_R A10

#define IR\_TX\_TL 3 // Top Left IR emitter LED on digital pin 2

#define IR\_TX\_TR 4 // Top Right IR emitter LED on digital pin 3

#define IR\_TX\_L 22

#define IR\_TX\_R 2

bool readGesture = true; // Used to process the gesture

int ambientIR\_TL; // Stores the ambient IR value read from the TL IR receiver

int ambientIR\_TR; // Stores the ambient IR value read from the TR IR receiver

int ambientIR\_L;

int ambientIR\_R;

int obstacleIR\_TL; // Stores the raw object IR value read from the TL IR receiver

int obstacleIR\_TR; // Stores the raw object IR value read from the TR IR receiver

int obstacleIR\_L;

int obstacleIR\_R;

int value\_TL[10]; // Stores the IR value read from the TL IR receiver

int value\_TR[10]; // Stores the IR value read from the TR IR receiver

int value\_L[10];

int value\_R[10];

int distance\_TL; // Stores the mapped IR value read from the TL IR receiver

int distance\_TR; // Stores the mapped IR value read from the TR IR receiver

int distance\_L;

int distance\_R;

int calibration\_TL; // Stores the raw IR value read from the TL receiver when the Arduino is powered

int calibration\_TR; // Stores the raw IR value read from the TR receiver when the Arduino is powered

int calibration\_L;

int calibration\_R;

int lastDistanceHold\_TL; // The next variables are used to detect a "hold gesture" action

int lastDistanceHold\_TR;

int lastDistanceHold\_L;

int lastDistanceHold\_R;

int holdMillis = 300; // Used in a timer to determine a "hold gesture" action

uint8\_t distanceArray\_TL[100]; // Stores the last 100 IR values from the TL IR receiver to determine the gesture later

uint8\_t distanceArray\_TR[100]; // Stores the last 100 IR values from the TR IR receiver to determine the gesture later

uint8\_t distanceArray\_L[100];

uint8\_t distanceArray\_R[100];

uint8\_t distanceArrayCounter = 0; // Used to access the previous array's content

uint8\_t gesture = 0; // Stores the gesture

uint32\_t lastHoldMillis = 0;

void setup() {

// put your setup code here, to run once:

Serial.begin(9600);

pinMode(IR\_TX\_TL, OUTPUT);

pinMode(IR\_TX\_TR, OUTPUT);

pinMode(IR\_TX\_L, OUTPUT);

pinMode(IR\_TX\_R, OUTPUT);

digitalWrite(IR\_TX\_TL, LOW);

digitalWrite(IR\_TX\_TR, LOW);

digitalWrite(IR\_TX\_L, LOW);

digitalWrite(IR\_TX\_R, LOW);

calibrateIR();

pinMode(IR\_TX\_TL2, OUTPUT);

pinMode(IR\_TX\_TR2, OUTPUT);

pinMode(IR\_TX\_L2, OUTPUT);

pinMode(IR\_TX\_R2, OUTPUT);

digitalWrite(IR\_TX\_TL2, LOW);

digitalWrite(IR\_TX\_TR2, LOW);

digitalWrite(IR\_TX\_L2, LOW);

digitalWrite(IR\_TX\_R2, LOW);

pinMode(5,OUTPUT);

pinMode(6,OUTPUT);

pinMode(11,OUTPUT);

pinMode(12,OUTPUT);

digitalWrite(5,HIGH);

digitalWrite(6,HIGH);

digitalWrite(11,HIGH);

digitalWrite(12,HIGH);

caliBR2ateIR();

}

void loop() {

// put your main code here, to run repeatedly:

readIR(5); // Read 5 times the IR values and store the average

readIR2(5);

determineGesture(); // It determines the type of gesture made

determinegesture2();

processHoldGesture(); // It checks if the hand is being hold

processHoldgesture2();

f();

g();

}

void calibrateIR(){

for(uint8\_t x = 0; x < 5; x++){

digitalWrite(IR\_TX\_TL,LOW);

digitalWrite(IR\_TX\_TR,LOW);

digitalWrite(IR\_TX\_L,LOW);

digitalWrite(IR\_TX\_R,LOW);

delay(2);

ambientIR\_TL = analogRead(IR\_RX\_TL);

ambientIR\_TR = analogRead(IR\_RX\_TR);

ambientIR\_L = analogRead(IR\_RX\_L);

ambientIR\_R = analogRead(IR\_RX\_R);

digitalWrite(IR\_TX\_TL,HIGH);

digitalWrite(IR\_TX\_TR,HIGH);

digitalWrite(IR\_TX\_L,HIGH);

digitalWrite(IR\_TX\_R,HIGH);

delay(1);

obstacleIR\_TL = analogRead(IR\_RX\_TL);

obstacleIR\_TR = analogRead(IR\_RX\_TR);

obstacleIR\_L = analogRead(IR\_RX\_L);

obstacleIR\_R = analogRead(IR\_RX\_R);

value\_TL[x] = - ambientIR\_TL + obstacleIR\_TL;

value\_TR[x] = - ambientIR\_TR + obstacleIR\_TR;

value\_L[x] = - ambientIR\_L + obstacleIR\_L;

value\_R[x] = - ambientIR\_R + obstacleIR\_R;

}

for(uint8\_t x = 0; x < 5; x++){

distance\_TL += value\_TL[x];

distance\_TR += value\_TR[x];

distance\_L += value\_L[x];

distance\_R += value\_R[x];

}

calibration\_TL = (distance\_TL / 5);

calibration\_TR = (distance\_TR / 5);

calibration\_L = (distance\_L / 5);

calibration\_R = (distance\_R / 5);

}

void caliBR2ateIR(){

for(uint8\_t x = 0; x < 5; x++){

digitalWrite(IR\_TX\_TL2,LOW);

digitalWrite(IR\_TX\_TR2,LOW);

digitalWrite(IR\_TX\_L2,LOW);

digitalWrite(IR\_TX\_R2,LOW);

delay(2);

ambientIR\_TL2 = analogRead(IR\_RX\_TL2);

ambientIR\_TR2 = analogRead(IR\_RX\_TR2);

ambientIR\_L2 = analogRead(IR\_RX\_L2);

ambientIR\_R2 = analogRead(IR\_RX\_R2);

digitalWrite(IR\_TX\_TL2,HIGH);

digitalWrite(IR\_TX\_TR2,HIGH);

digitalWrite(IR\_TX\_L2,HIGH);

digitalWrite(IR\_TX\_R2,HIGH);

delay(1);

obstacleIR\_TL2 = analogRead(IR\_RX\_TL2);

obstacleIR\_TR2 = analogRead(IR\_RX\_TR2);

obstacleIR\_L2 = analogRead(IR\_RX\_L2);

obstacleIR\_R2 = analogRead(IR\_RX\_R2);

value\_TL2[x] = - ambientIR\_TL2 + obstacleIR\_TL2;

value\_TR2[x] = - ambientIR\_TR2 + obstacleIR\_TR2;

value\_L2[x] = - ambientIR\_L2 + obstacleIR\_L2;

value\_R2[x] = - ambientIR\_R2 + obstacleIR\_R2;

}

for(uint8\_t x = 0; x < 5; x++){

distance\_TL2 += value\_TL2[x];

distance\_TR2 += value\_TR2[x];

distance\_L2 += value\_L2[x];

distance\_R2 += value\_R2[x];

}

caliBR2ation\_TL2 = (distance\_TL2 / 5);

caliBR2ation\_TR2 = (distance\_TR2 / 5);

caliBR2ation\_L2 = (distance\_L2 / 5);

caliBR2ation\_R2 = (distance\_R2 / 5);

}

void readIR(uint8\_t times){

for(uint8\_t x = 0; x < times; x++){

digitalWrite(IR\_TX\_TL,LOW); // We need to turn off the IR LEDs to read the ambient IR

digitalWrite(IR\_TX\_TR,LOW);

digitalWrite(IR\_TX\_L,LOW);

digitalWrite(IR\_TX\_R,LOW);

delay(2); // Wait until the IR LEDs are completely turned off

ambientIR\_TL = analogRead(IR\_RX\_TL); // Stores the ambient IR light

ambientIR\_TR = analogRead(IR\_RX\_TR);

ambientIR\_L = analogRead(IR\_RX\_L);

ambientIR\_R = analogRead(IR\_RX\_R);

digitalWrite(IR\_TX\_TL,HIGH); // Turn on the IR LEDs to read the IR light reflected by the obstacle

digitalWrite(IR\_TX\_TR,HIGH);

digitalWrite(IR\_TX\_L,HIGH);

digitalWrite(IR\_TX\_R,HIGH);

delay(1); // Wait until the IR LEDs are completely turned on

obstacleIR\_TL = analogRead(IR\_RX\_TL); // Stores the IR light reflected by the obstacle

obstacleIR\_TR = analogRead(IR\_RX\_TR);

obstacleIR\_L = analogRead(IR\_RX\_L);

obstacleIR\_R = analogRead(IR\_RX\_R);

value\_TL[x] = - ambientIR\_TL + obstacleIR\_TL; // Store the IR value to later calculate the average

value\_TR[x] = - ambientIR\_TR + obstacleIR\_TR;

value\_L[x] = - ambientIR\_L + obstacleIR\_L;

value\_R[x] = - ambientIR\_R + obstacleIR\_R;

}

for(uint8\_t x = 0; x < times; x++){ // Calculate the average IR value per sensor

distance\_TL += value\_TL[x];

distance\_TR += value\_TR[x];

distance\_L += value\_L[x];

distance\_R += value\_R[x];

}

distance\_TL = (distance\_TL / times); // Save the distance value

distance\_TR = (distance\_TR / times);

distance\_L = (distance\_L / times);

distance\_R = (distance\_R / times);

if(calibration\_TL + 50 > distance\_TL){ // Calibrate the sensors

distance\_TL = 0;

}

else{

distance\_TL -= calibration\_TL + 50;

}

if(calibration\_TR + 50 > distance\_TR){

distance\_TR = 0;

}

else{

distance\_TR -= calibration\_TR + 50;

}

if(calibration\_L + 50 > distance\_L){

distance\_L = 0;

}

else{

distance\_L -= calibration\_L + 50;

}

if(calibration\_R + 50 > distance\_R){

distance\_R = 0;

}

else{

distance\_R -= calibration\_R + 50;

}

// If any distance is greater than 0 then map it from a 0-1023 range to a 0-255 range (to be able to store it in a 1B variable) and store it in an array

if(distance\_TL != 0 || distance\_TR != 0 || distance\_L != 0 || distance\_R != 0){

distance\_TR = map(constrain(distance\_TR, 0, 1100), 0, 1100, 0, 255);

distance\_TL = map(constrain(distance\_TL, 0, 1100), 0, 1100, 0, 255);

distance\_L = map(constrain(distance\_L, 0, 1100), 0, 1100, 0, 255);

distance\_R = map(constrain(distance\_R, 0, 1100), 0, 1100, 0, 255);

if(distanceArrayCounter < 100){

distanceArray\_TR[distanceArrayCounter] = distance\_TR;

distanceArray\_TL[distanceArrayCounter] = distance\_TL;

distanceArray\_R[distanceArrayCounter] = distance\_R;

distanceArray\_L[distanceArrayCounter] = distance\_L;

distanceArrayCounter += 1;

}

else{

distanceArray\_TR[0] = distance\_TR;

distanceArray\_TL[0] = distance\_TL;

distanceArray\_L[0] = distance\_L;

distanceArray\_R[0] = distance\_R;

distanceArrayCounter = 1;

}

}

else if(distance\_TL == 0 && distance\_TR == 0 && distance\_L == 0 && distance\_R == 0){ // If not then reset the arrays

readGesture = true;

distanceArrayCounter = 0;

distanceArray\_TR[0] = 0;

distanceArray\_TL[0] = 0;

distanceArray\_R[0] = 0;

distanceArray\_L[0] = 0;

}

}

void determineGesture(){

if(distanceArrayCounter > 2){ // If the arrays have more than 2 elements then determine the gesture

uint8\_t distancePost\_TL = 0; // The next "distancePost..." vars are the sum of it previous value plus

uint8\_t distancePost\_TR = 0; // the last element minus the previous element of the array

uint8\_t distancePost\_L = 0;

uint8\_t distancePost\_R = 0;

uint8\_t distancePostNeg\_TL = 0;

uint8\_t distancePostNeg\_TR = 0;

uint8\_t distancePostNeg\_R = 0;

uint8\_t distancePostNeg\_L = 0;

uint8\_t expectedGesture = 0; // Used later to determine the gesture made

bool increasing\_TL = false; // Used later to determine the gesture made. If the IR sensor's value

bool decreasing\_TL = false; // increases then "increasing\_xx" is true. Otherwise, if it decreasing

bool increasing\_TR = false; // then "decreasing\_xx" is true

bool decreasing\_TR = false;

bool increasing\_L = false;

bool decreasing\_L = false;

bool increasing\_R = false;

bool decreasing\_R = false;

// For every element in the distanceArray...

for(uint8\_t x = 1; x < distanceArrayCounter; x++){

// Update the "distancePost..." vars values

if(distanceArray\_TL[x] - distanceArray\_TL[x-1] > 0){

distancePost\_TL += distanceArray\_TL[x] - distanceArray\_TL[x-1];

}

else{

distancePostNeg\_TL += distanceArray\_TL[x-1] - distanceArray\_TL[x];

}

if(distanceArray\_TR[x] - distanceArray\_TR[x-1] > 0){

distancePost\_TR += distanceArray\_TR[x] - distanceArray\_TR[x-1];

}

else{

distancePostNeg\_TR += distanceArray\_TR[x-1] - distanceArray\_TR[x];

}

if(distanceArray\_L[x] - distanceArray\_L[x-1] > 0){

distancePost\_L += distanceArray\_L[x] - distanceArray\_L[x-1];

}

else{

distancePostNeg\_L += distanceArray\_L[x-1] - distanceArray\_L[x];

}

if(distanceArray\_R[x] - distanceArray\_R[x-1] > 0){

distancePost\_R += distanceArray\_R[x] - distanceArray\_R[x-1];

}

else{

distancePostNeg\_R += distanceArray\_R[x-1] - distanceArray\_R[x];

}

}

// Based on the "distancePost..." vars determine if the value is increasing or decreasing

if(distancePost\_TL > 10 && distancePostNeg\_TL < 10){

increasing\_TL = true;

decreasing\_TL = false;

}

else if(distancePostNeg\_TL > 10){

increasing\_TL = false;

decreasing\_TL = true;

}

if(distancePost\_TR > 10 && distancePostNeg\_TR < 10){

increasing\_TR = true;

decreasing\_TR = false;

}

else if(distancePostNeg\_TR > 10){

increasing\_TR = false;

decreasing\_TR = true;

}

if(distancePost\_L > 10 && distancePostNeg\_L < 10){

increasing\_L = true;

decreasing\_L = false;

}

else if(distancePostNeg\_L > 10){

increasing\_L = false;

decreasing\_L = true;

}

if(distancePost\_R > 10 && distancePostNeg\_R < 10){

increasing\_R = true;

decreasing\_R = false;

}

else if(distancePostNeg\_R > 10){

increasing\_R = false;

decreasing\_R = true;

}

// Determine the type of gesture //for present south and north gestures have not been updated

switch(expectedGesture){

case 0: // If "expectedGesture" is 0 then determine the expected gesture

// by analyzing if the sensors' values are increasing or decreasing

if((increasing\_TR && !increasing\_TL) || (increasing\_R && !increasing\_L)){ // Left to right movement

expectedGesture = 1; // Expected an "east" gesture

}

else if((!increasing\_TR && increasing\_TL) || (!increasing\_R && increasing\_L)){ // Right to left movement

expectedGesture = 2; // Expected a "west" gesture

}

else if(increasing\_TR && increasing\_TL ){ // Top to bottom movement

expectedGesture = 3; // Expected a "south" gesture

}

else if( !increasing\_TL && !increasing\_TR){ // Bottom to top movement

expectedGesture = 4; // Expected a "north" gesture

}

// Check if the current "distance\_xx" minus the 2nd element in the array is greater than 50 then it is a "down" movement.

// The "increasing\_xx" vars are not used because they are very precise and they didn´t detect the "down" movement most of the time.

else if(distance\_TL - distanceArray\_TL[2] > 50 && distance\_TR - distanceArray\_TR[2] > 50){ // Up to down movementd

expectedGesture = 5; // Expected a "down" gesture

}

case 1:

// If "expectedGesture" is "east" then check if TR and BR sensors are increasing and TL and BL are decreasing.

// If this is true then save the current distance to the var "lastDistanceHold" to use it later to check a "hold

// gesture" action. Also, reset the "expectedGesture" var and set the "gesture" var to 1 ("east")

if((increasing\_TR && !increasing\_TL) || (increasing\_R && !increasing\_L)){

lastDistanceHold\_TR = distance\_TR;

lastDistanceHold\_TL = distance\_TL;

lastDistanceHold\_R = distance\_R;

lastDistanceHold\_L = distance\_L;

if(readGesture){

//Serial.println("EAST");

a=1;

gesture = 1;

holdMillis = 300; // Change it to 300ms to add a small delay between the gesture "east" and the "hold gesture" action

lastHoldMillis = millis();

}

expectedGesture = 0;

increasing\_TL = false;

decreasing\_TL = false;

increasing\_TR = false;

decreasing\_TR = false;

increasing\_R = false;

decreasing\_R = false;

increasing\_L = false;

decreasing\_L = false;

readGesture = false;

}

case 2:

if((!increasing\_TR && increasing\_TL) || (!increasing\_R && increasing\_L)){

if(readGesture){

//Serial.println("WEST");

a=2;

gesture = 2;

holdMillis = 300;

lastHoldMillis = millis();

}

expectedGesture = 0;

lastDistanceHold\_TR = distance\_TR;

lastDistanceHold\_TL = distance\_TL;

lastDistanceHold\_R = distance\_R;

lastDistanceHold\_L = distance\_L;

increasing\_TL = false;

decreasing\_TL = false;

increasing\_TR = false;

decreasing\_TR = false;

increasing\_R = false;

decreasing\_R = false;

increasing\_L = false;

decreasing\_L = false;

readGesture = false;

}

break;

/\* case 3:

if(increasing\_BR && increasing\_BL && decreasing\_TL && decreasing\_TR){

if(readGesture){

Serial.println("SOUTH");

gesture = 3;

holdMillis = 300;

lastHoldMillis = millis();

}

expectedGesture = 0;

lastDistanceHold\_TR = distance\_TR;

lastDistanceHold\_BR = distance\_BR;

lastDistanceHold\_TL = distance\_TL;

lastDistanceHold\_BL = distance\_BL;

increasing\_TL = false;

decreasing\_TL = false;

increasing\_TR = false;

decreasing\_TR = false;

increasing\_BL = false;

decreasing\_BL = false;

increasing\_BR = false;

decreasing\_BR = false;

readGesture = false;

}

break;

case 4:

if(increasing\_TR && increasing\_TL && decreasing\_BL && decreasing\_BR){

if(readGesture){

Serial.println("NORTH");

gesture = 4;

holdMillis = 300;

lastHoldMillis = millis();

}

expectedGesture = 0;

lastDistanceHold\_TR = distance\_TR;

lastDistanceHold\_BR = distance\_BR;

lastDistanceHold\_TL = distance\_TL;

lastDistanceHold\_BL = distance\_BL;

increasing\_TL = false;

decreasing\_TL = false;

increasing\_TR = false;

decreasing\_TR = false;

increasing\_BL = false;

decreasing\_BL = false;

increasing\_BR = false;

decreasing\_BR = false;

readGesture = false;

}

break;

case 5:

// The next if statement checks if 3 sensors are increasing its value and if it is true then store a "down" gesture

if((increasing\_BR && increasing\_BL && increasing\_TL) || (increasing\_TR && increasing\_BL && increasing\_TL) || (increasing\_TR && increasing\_BR && increasing\_TL) || (increasing\_TR && increasing\_BR && increasing\_BL)){

if(readGesture){

Serial.println("DOWN");

gesture = 5;

holdMillis = 300;

lastHoldMillis = millis();

}

expectedGesture = 0;

increasing\_TL = false;

decreasing\_TL = false;

increasing\_TR = false;

decreasing\_TR = false;

increasing\_BL = false;

decreasing\_BL = false;

increasing\_BR = false;

decreasing\_BR = false;

readGesture = false;

}

break;\*/

}

}

}

void processHoldGesture(){

// If the actual "distance\_xx" minus the "lastDistanceHold\_xx" (saved when the gesture was detected)

// is within a range (20) then make a "hold gesture" action

if(-20 <= distance\_TR - lastDistanceHold\_TR && distance\_TR - lastDistanceHold\_TR <= 20 && -20 <= distance\_TL - lastDistanceHold\_TL && distance\_TL - lastDistanceHold\_TL <= 20 && -20 <= distance\_R - lastDistanceHold\_R && distance\_R - lastDistanceHold\_R <= 20 && -20 <= distance\_L - lastDistanceHold\_L && distance\_L - lastDistanceHold\_L <= 20){

if(distance\_TR != 0 || distance\_TL != 0 || distance\_L != 0 || distance\_R != 0 ){

switch(gesture){

case 1:

if(millis() - lastHoldMillis > holdMillis){

holdMillis = 100;

//Serial.println("HOLD EAST");

a=1;

lastHoldMillis = millis();

}

break;

case 2:

if(millis() - lastHoldMillis > holdMillis){

holdMillis = 100;

//Serial.println("HOLD WEST");

a=2;

lastHoldMillis = millis();

}

break;

case 3:

if(millis() - lastHoldMillis > holdMillis){

holdMillis = 100;

Serial.println("HOLD SOUTH");

lastHoldMillis = millis();

}

break;

case 4:

if(millis() - lastHoldMillis > holdMillis){

holdMillis = 100;

Serial.println("HOLD NORTH");

lastHoldMillis = millis();

}

break;

}

}

else{

// Set a high numbenr for the "lastDistanceHold" vars to stop reading a "hold gesture" action

lastDistanceHold\_TL = 20000;

lastDistanceHold\_TR = 20000;

lastDistanceHold\_L = 20000;

lastDistanceHold\_R = 20000;

}

}

else{

lastDistanceHold\_TL = 20000;

lastDistanceHold\_TR = 20000;

lastDistanceHold\_R = 20000;

lastDistanceHold\_L = 20000;

}

}

void readIR2(uint8\_t times){

for(uint8\_t x = 0; x < times; x++){

digitalWrite(IR\_TX\_TL2,LOW); // We need to turn off the IR LEDs to read the ambient IR

digitalWrite(IR\_TX\_TR2,LOW);

digitalWrite(IR\_TX\_L2,LOW);

digitalWrite(IR\_TX\_R2,LOW);

delay(2); // Wait until the IR LEDs are completely turned off

ambientIR\_TL2 = analogRead(IR\_RX\_TL2); // Stores the ambient IR light

ambientIR\_TR2 = analogRead(IR\_RX\_TR2);

ambientIR\_L2 = analogRead(IR\_RX\_L2);

ambientIR\_R2 = analogRead(IR\_RX\_R2);

digitalWrite(IR\_TX\_TL2,HIGH); // Turn on the IR LEDs to read the IR light reflected by the obstacle

digitalWrite(IR\_TX\_TR2,HIGH);

digitalWrite(IR\_TX\_L2,HIGH);

digitalWrite(IR\_TX\_R2,HIGH);

delay(1); // Wait until the IR LEDs are completely turned on

obstacleIR\_TL2 = analogRead(IR\_RX\_TL2); // Stores the IR light reflected by the obstacle

obstacleIR\_TR2 = analogRead(IR\_RX\_TR2);

obstacleIR\_L2 = analogRead(IR\_RX\_L2);

obstacleIR\_R2 = analogRead(IR\_RX\_R2);

value\_TL2[x] = - ambientIR\_TL2 + obstacleIR\_TL2; // Store the IR value to later calculate the average

value\_TR2[x] = - ambientIR\_TR2 + obstacleIR\_TR2;

value\_L2[x] = - ambientIR\_L2 + obstacleIR\_L2;

value\_R2[x] = - ambientIR\_R2 + obstacleIR\_R2;

}

for(uint8\_t x = 0; x < times; x++){ // Calculate the average IR value per sensor

distance\_TL2 += value\_TL2[x];

distance\_TR2 += value\_TR2[x];

distance\_L2 += value\_L2[x];

distance\_R2 += value\_R2[x];

}

distance\_TL2 = (distance\_TL2 / times); // Save the distance value

distance\_TR2 = (distance\_TR2 / times);

distance\_L2 = (distance\_L2 / times);

distance\_R2 = (distance\_R2 / times);

if(caliBR2ation\_TL2 + 50 > distance\_TL2){ // CaliBR2ate the sensors

distance\_TL2 = 0;

}

else{

distance\_TL2 -= caliBR2ation\_TL2 + 50;

}

if(caliBR2ation\_TR2 + 50 > distance\_TR2){

distance\_TR2 = 0;

}

else{

distance\_TR2 -= caliBR2ation\_TR2 + 50;

}

if(caliBR2ation\_L2 + 50 > distance\_L2){

distance\_L2 = 0;

}

else{

distance\_L2 -= caliBR2ation\_L2 + 50;

}

if(caliBR2ation\_R2 + 50 > distance\_R2){

distance\_R2 = 0;

}

else{

distance\_R2 -= caliBR2ation\_R2 + 50;

}

// If any distance is greater than 0 then map it from a 0-1023 range to a 0-255 range (to be aBL2e to store it in a 1B variaBL2e) and store it in an array

if(distance\_TL2 != 0 || distance\_TR2 != 0 || distance\_L2 != 0 || distance\_R2 != 0){

distance\_TR2 = map(constrain(distance\_TR2, 0, 1100), 0, 1100, 0, 255);

distance\_TL2 = map(constrain(distance\_TL2, 0, 1100), 0, 1100, 0, 255);

distance\_L2 = map(constrain(distance\_L2, 0, 1100), 0, 1100, 0, 255);

distance\_R2 = map(constrain(distance\_R2, 0, 1100), 0, 1100, 0, 255);

if(distanceArrayCounter2 < 100){

distanceArray\_TR2[distanceArrayCounter2] = distance\_TR2;

distanceArray\_TL2[distanceArrayCounter2] = distance\_TL2;

distanceArray\_R2[distanceArrayCounter2] = distance\_R2;

distanceArray\_L2[distanceArrayCounter2] = distance\_L2;

distanceArrayCounter2 += 1;

}

else{

distanceArray\_TR2[0] = distance\_TR2;

distanceArray\_TL2[0] = distance\_TL2;

distanceArray\_L2[0] = distance\_L2;

distanceArray\_R2[0] = distance\_R2;

distanceArrayCounter2 = 1;

}

}

else if(distance\_TL2 == 0 && distance\_TR2 == 0 && distance\_L2 == 0 && distance\_R2 == 0){ // If not then reset the arrays

readGesture2 = true;

distanceArrayCounter2 = 0;

distanceArray\_TR2[0] = 0;

distanceArray\_TL2[0] = 0;

distanceArray\_R2[0] = 0;

distanceArray\_L2[0] = 0;

}

}

void determinegesture2(){

if(distanceArrayCounter2 > 2){ // If the arrays have more than 2 elements then determine the gesture2

uint8\_t distancePost\_TL2 = 0; // The next "distancePost..." vars are the sum of it previous value plus

uint8\_t distancePost\_TR2 = 0; // the last element minus the previous element of the array

uint8\_t distancePost\_L2 = 0;

uint8\_t distancePost\_R2 = 0;

uint8\_t distancePostNeg\_TL2 = 0;

uint8\_t distancePostNeg\_TR2 = 0;

uint8\_t distancePostNeg\_R2 = 0;

uint8\_t distancePostNeg\_L2 = 0;

uint8\_t expectedGesture2 = 0; // Used later to determine the gesture2 made

bool increasing\_TL2 = false; // Used later to determine the gesture2 made. If the IR sensor's value

bool decreasing\_TL2 = false; // increases then "increasing\_xx" is true. Otherwise, if it decreasing

bool increasing\_TR2 = false; // then "decreasing\_xx" is true

bool decreasing\_TR2 = false;

bool increasing\_L2 = false;

bool decreasing\_L2 = false;

bool increasing\_R2 = false;

bool decreasing\_R2 = false;

// For every element in the distanceArray...

for(uint8\_t x = 1; x < distanceArrayCounter2; x++){

// Update the "distancePost..." vars values

if(distanceArray\_TL2[x] - distanceArray\_TL2[x-1] > 0){

distancePost\_TL2 += distanceArray\_TL2[x] - distanceArray\_TL2[x-1];

}

else{

distancePostNeg\_TL2 += distanceArray\_TL2[x-1] - distanceArray\_TL2[x];

}

if(distanceArray\_TR2[x] - distanceArray\_TR2[x-1] > 0){

distancePost\_TR2 += distanceArray\_TR2[x] - distanceArray\_TR2[x-1];

}

else{

distancePostNeg\_TR2 += distanceArray\_TR2[x-1] - distanceArray\_TR2[x];

}

if(distanceArray\_L2[x] - distanceArray\_L2[x-1] > 0){

distancePost\_L2 += distanceArray\_L2[x] - distanceArray\_L2[x-1];

}

else{

distancePostNeg\_L2 += distanceArray\_L2[x-1] - distanceArray\_L2[x];

}

if(distanceArray\_R2[x] - distanceArray\_R2[x-1] > 0){

distancePost\_R2 += distanceArray\_R2[x] - distanceArray\_R2[x-1];

}

else{

distancePostNeg\_R2 += distanceArray\_R2[x-1] - distanceArray\_R2[x];

}

}

// Based on the "distancePost..." vars determine if the value is increasing or decreasing

if(distancePost\_TL2 > 10 && distancePostNeg\_TL2 < 10){

increasing\_TL2 = true;

decreasing\_TL2 = false;

}

else if(distancePostNeg\_TL2 > 10){

increasing\_TL2 = false;

decreasing\_TL2 = true;

}

if(distancePost\_TR2 > 10 && distancePostNeg\_TR2 < 10){

increasing\_TR2 = true;

decreasing\_TR2 = false;

}

else if(distancePostNeg\_TR2 > 10){

increasing\_TR2 = false;

decreasing\_TR2 = true;

}

if(distancePost\_L2 > 10 && distancePostNeg\_L2 < 10){

increasing\_L2 = true;

decreasing\_L2 = false;

}

else if(distancePostNeg\_L2 > 10){

increasing\_L2 = false;

decreasing\_L2 = true;

}

if(distancePost\_R2 > 10 && distancePostNeg\_R2 < 10){

increasing\_R2 = true;

decreasing\_R2 = false;

}

else if(distancePostNeg\_R2 > 10){

increasing\_R2 = false;

decreasing\_R2 = true;

}

// Determine the type of gesture2 //for present south and north gesture2s have not been updated

switch(expectedGesture2){

case 0: // If "expectedGesture2" is 0 then determine the expected gesture2

// by analyzing if the sensors' values are increasing or decreasing

if((increasing\_TR2 && !increasing\_TL2) || (increasing\_R2 && !increasing\_L2)){ // Left to right movement

expectedGesture2 = 1; // Expected an "east" gesture2

}

else if((!increasing\_TR2 && increasing\_TL2) || (!increasing\_R2 && increasing\_L2)){ // Right to left movement

expectedGesture2 = 2; // Expected a "west" gesture2

}

else if(increasing\_TR2 && increasing\_TL2 ){ // Top to bottom movement

expectedGesture2 = 3; // Expected a "south" gesture2

}

else if( !increasing\_TL2 && !increasing\_TR2){ // Bottom to top movement

expectedGesture2 = 4; // Expected a "north" gesture2

}

// Check if the current "distance\_xx" minus the 2nd element in the array is greater than 50 then it is a "down" movement.

// The "increasing\_xx" vars are not used because they are very precise and they didn´t detect the "down" movement most of the time.

else if(distance\_TL2 - distanceArray\_TL2[2] > 50 && distance\_TR2 - distanceArray\_TR2[2] > 50){ // Up to down movementd

expectedGesture2 = 5; // Expected a "down" gesture2

}

case 1:

// If "expectedGesture2" is "east" then check if TR2 and BR2 sensors are increasing and TL2 and BL2 are decreasing.

// If this is true then save the current distance to the var "lastDistanceHold" to use it later to check a "hold

// gesture2" action. Also, reset the "expectedGesture2" var and set the "gesture2" var to 1 ("east")

if((increasing\_TR2 && !increasing\_TL2) || (increasing\_R2 && !increasing\_L2)){

lastDistanceHold\_TR2 = distance\_TR2;

lastDistanceHold\_TL2 = distance\_TL2;

lastDistanceHold\_R2 = distance\_R2;

lastDistanceHold\_L2 = distance\_L2;

if(readGesture2){

//Serial.println("EAST");

b=1;

gesture2 = 1;

holdMillis2 = 300; // Change it to 300ms to add a small delay between the gesture2 "east" and the "hold gesture2" action

lastHoldMillis2 = millis();

}

expectedGesture2 = 0;

increasing\_TL2 = false;

decreasing\_TL2 = false;

increasing\_TR2 = false;

decreasing\_TR2 = false;

increasing\_R2 = false;

decreasing\_R2 = false;

increasing\_L2 = false;

decreasing\_L2 = false;

readGesture2 = false;

}

case 2:

if((!increasing\_TR2 && increasing\_TL2) || (!increasing\_R2 && increasing\_L2))

{

if(readGesture2){

// Serial.println("WEST");

b=2;

gesture2 = 2;

holdMillis2 = 300;

lastHoldMillis2 = millis();

}

expectedGesture2 = 0;

lastDistanceHold\_TR2 = distance\_TR2;

lastDistanceHold\_TL2 = distance\_TL2;

lastDistanceHold\_R2 = distance\_R2;

lastDistanceHold\_L2 = distance\_L2;

increasing\_TL2 = false;

decreasing\_TL2 = false;

increasing\_TR2 = false;

decreasing\_TR2 = false;

increasing\_R2 = false;

decreasing\_R2 = false;

increasing\_L2 = false;

decreasing\_L2 = false;

readGesture2 = false;

}

break;

/\* case 3:

if(increasing\_BR2 && increasing\_BL2 && decreasing\_TL2 && decreasing\_TR2){

if(readGesture2){

Serial.println("SOUTH");

gesture2 = 3;

holdMillis2 = 300;

lastHoldMillis2 = millis();

}

expectedGesture2 = 0;

lastDistanceHold\_TR2 = distance\_TR2;

lastDistanceHold\_BR2 = distance\_BR2;

lastDistanceHold\_TL2 = distance\_TL2;

lastDistanceHold\_BL2 = distance\_BL2;

increasing\_TL2 = false;

decreasing\_TL2 = false;

increasing\_TR2 = false;

decreasing\_TR2 = false;

increasing\_BL2 = false;

decreasing\_BL2 = false;

increasing\_BR2 = false;

decreasing\_BR2 = false;

readGesture2 = false;

}

case 4:

if(increasing\_TR2 && increasing\_TL2 && decreasing\_BL2 && decreasing\_BR2){

if(readGesture2){

Serial.println("NORTH");

gesture2 = 4;

holdMillis2 = 300;

lastHoldMillis2 = millis();

}

expectedGesture2 = 0;

lastDistanceHold\_TR2 = distance\_TR2;

lastDistanceHold\_BR2 = distance\_BR2;

lastDistanceHold\_TL2 = distance\_TL2;

lastDistanceHold\_BL2 = distance\_BL2;

increasing\_TL2 = false;

decreasing\_TL2 = false;

increasing\_TR2 = false;

decreasing\_TR2 = false;

increasing\_BL2 = false;

decreasing\_BL2 = false;

increasing\_BR2 = false;

decreasing\_BR2 = false;

readGesture2 = false;

}

break;

/\*case 5:

// The next if statement checks if 3 sensors are increasing its value and if it is true then store a "down" gesture2

if((increasing\_BR2 && increasing\_BL2 && increasing\_TL2) || (increasing\_TR2 && increasing\_BL2 && increasing\_TL2) || (increasing\_TR2 && increasing\_BR2 && increasing\_TL2) || (increasing\_TR2 && increasing\_BR2 && increasing\_BL2)){

if(readGesture2){

Serial.println("DOWN");

gesture2 = 5;

holdMillis2 = 300;

lastHoldMillis2 = millis();

}

expectedGesture2 = 0;

increasing\_TL2 = false;

decreasing\_TL2 = false;

increasing\_TR2 = false;

decreasing\_TR2 = false;

increasing\_BL2 = false;

decreasing\_BL2 = false;

increasing\_BR2 = false;

decreasing\_BR2 = false;

readGesture2 = false;

}

break;\*/

}

}

}

void processHoldgesture2(){

// If the actual "distance\_xx" minus the "lastDistanceHold\_xx" (saved when the gesture2 was detected)

// is within a range (20) then make a "hold gesture2" action

if(-20 <= distance\_TR2 - lastDistanceHold\_TR2 && distance\_TR2 - lastDistanceHold\_TR2 <= 20 && -20 <= distance\_TL2 - lastDistanceHold\_TL2 && distance\_TL2 - lastDistanceHold\_TL2 <= 20 && -20 <= distance\_R2 - lastDistanceHold\_R2 && distance\_R2 - lastDistanceHold\_R2 <= 20 && -20 <= distance\_L2 - lastDistanceHold\_L2 && distance\_L2 - lastDistanceHold\_L2 <= 20){

if(distance\_TR2 != 0 || distance\_TL2 != 0 || distance\_L2 != 0 || distance\_R2 != 0 ){

switch(gesture2){

case 1:

if(millis() - lastHoldMillis2 > holdMillis2){

holdMillis2 = 100;

//Serial.println("HOLD EAST");

b=1;

lastHoldMillis2 = millis();

}

break;

case 2:

if(millis() - lastHoldMillis2 > holdMillis2){

holdMillis2 = 100;

// Serial.println("HOLD WEST");

b=2;

lastHoldMillis2 = millis();

}

break;

case 3:

if(millis() - lastHoldMillis2 > holdMillis2){

holdMillis2 = 100;

Serial.println("HOLD SOUTH");

lastHoldMillis2 = millis();

}

break;

case 4:

if(millis() - lastHoldMillis2 > holdMillis2){

holdMillis2 = 100;

Serial.println("HOLD NORTH");

lastHoldMillis2 = millis();

}

break;

}

}

else{

// Set a high number for the "lastDistanceHold" vars to stop reading a "hold gesture2" action

lastDistanceHold\_TL2 = 20000;

lastDistanceHold\_TR2 = 20000;

lastDistanceHold\_L2 = 20000;

lastDistanceHold\_R2 = 20000;

}

}

else{

lastDistanceHold\_TL2 = 20000;

lastDistanceHold\_TR2 = 20000;

lastDistanceHold\_R2 = 20000;

lastDistanceHold\_L2 = 20000;

}

}

void f(){

if(a==1 && b==1){

Serial.println("rotateleft");

a=0;

b=0;

Serial.write(1);

Serial.flush();

delay(20);

}

else if(a==2 && b==2){

Serial.println("rotateright");

a=0;

b=0;

Serial.write(2);

Serial.flush();

delay(20);

}

else if(a==1 && b==2){

Serial.println("zoomout");

a=0;

b=0;

Serial.write(3);

Serial.flush();

delay(20);

}

else if(a==2 && b==1){

Serial.println("zoomin");

a=0;

b=0;

Serial.write(4);

Serial.flush();

delay(20);

}

}

void g(){

if(analogRead(A1)>700){

Serial.println("model2");

Serial.write(5);

Serial.flush();

delay(20);

}

else if(analogRead(A6)>600){

Serial.println("model3");

Serial.write(6);

Serial.flush();

delay(20);

}

//else if(analogRead(A7)>300){

// Serial.println("model4");

//}

}

* Anwesh: We decided to have more than one model in our game. Anwesh found another model in the asset store. Anwesh had to figure out a way to enable only model at a time and change them when required. At first, Anwesh tried using the SetActive command to enable and disable the models. But for some reason, it failed to work. Anwesh later thought of changing the mess renderer in both the models and after debugging a few errors, he got it to work.