Touch-Free Remote-Controlled Object Detection

Practical 3

Author: Joseph Manfredi Cameron

Abstract

In this project, the aim was to exploit the interactive capabilities of the Leap Motion sensor [1] to create an effective and intuitive system for detecting objects where users are not required to touch physical objects but instead make use of their bodies to achieve interaction. The system for detecting objects additionally has the capability for remote control, as the detection rig makes use of a camera which provides the user with a live video feed, hence users do not necessarily need to be present with the objects being detected. The context of this object-detection system is that it may be used on-board robots or rovers [2] to explore new territory, or for high-pressure scenarios such as bomb diffusion where operators need to efficiently survey the surrounding area without being physically present.

There are a couple of main motivations for making a touch-free remote-controlled system in this project. Firstly, with the presence of the COVID-19 global pandemic, there has been a highlighted need for systems that minimise contamination spread via both lack of physical contact with objects, and remote working where the distance between people is large enough to prevent viral spread. This system successfully allows for both factors to be accounted for. Also, most systems that require expertise to operate, particularly in this field of remote-object detection via controlling robots and rovers etc., usually are operated via touching joysticks and buttons that require users to learn and memorise abstract and arbitrary interactions. However, the Leap Motion controller allows users to instead interact via natural touch-free interactions that we as humans are more naturally familiar with. Interacting with the Leap Motion controller can trick the human brain into "feeling" like part of the system, rather than traditional methods that very much reinforce that you are operating a foreign system. It is insightful to see the effects on users that these intuitive interactions (via the Leap Motion controller) allowed when operating a versatile object-detection system of this nature.

This project was developed with Processing [3], two Arduino Uno microcontrollers [4], and of course the Leap Motion sensor and its accompanying software libraries. The Leap Motion sensor acts as a main input to the system, where it can track users' hands to move/activate hardware components connected to the Arduino microcontrollers or change settings of the system software in Processing. The software developed in Processing acts as a central hub for the entire system, and also serves as one of the main outputs of the system as it displays a live camera feed and continuously provides feedback to users via a display screen.

Interaction Overview

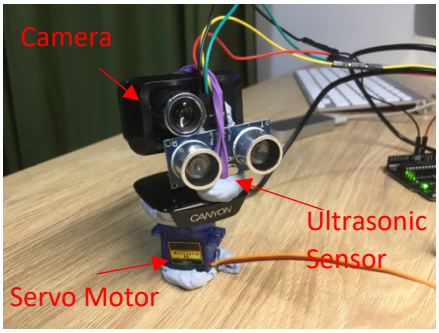

Figure 1: The Detection Rig

Note on the Detection-Rig Sub-System: The sub-system to detect objects is called the "detection-rig", it can be seen in Figure 2. The detection rig is a collection of multiple input/output hardware components that are collectively responsible for detecting objects and relaying that back and forth from the central software hub in Processing. It consists of a servo motor that rotates directly based on the user's hand input as detected by the Leap Motion sensor, a camera to allow for remote object detection as users can solely look through the camera at objects and do not necessarily need to be present, and an ultrasound sensor that measures the distance from the system to the objects in question. Two Arduino Uno microcontrollers are used to correctly handle the behaviour of these components and their communication to and from the system software in Processing.

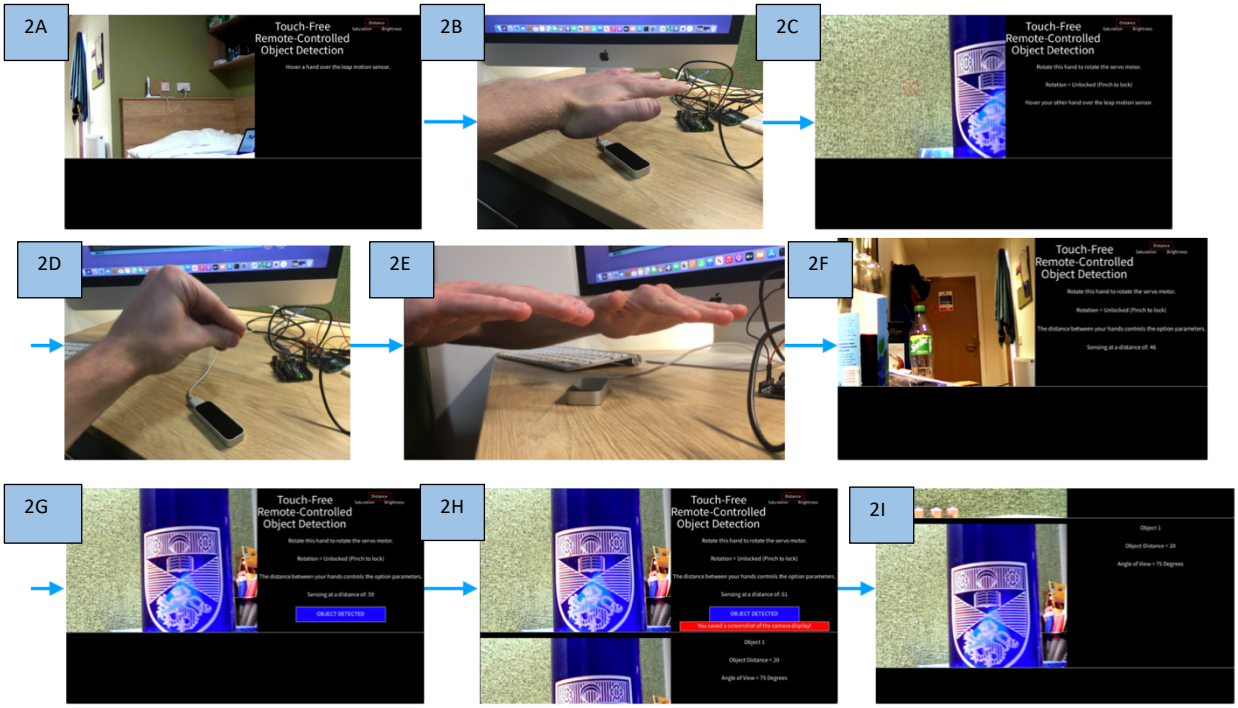

Figure 2

Figure 2 shows the standard flow of interaction with this object detection system when the default distance parameter is selected. Figure 2A shows the initial screen users are first presented with on the display monitor. The live camera video feed provided by the camera on the detection rig can be seen here in the top left corner. Furthermore, Figure 2A shows that there is an instruction to hover a hand over the Leap Motion controller. Figure 2B shows a user doing this. Once a user hovers their hand over the Leap Motion controller, this hand then becomes the controlling hand for the servo motor, where the angle of this hand around the axis beaming straight up from the Leap Motion controller directly corresponds to the angle of the servo motor. Since the detection rig is being rotated by the change of the servo motor's angle and is rotating around an axis, it felt natural to also make the interaction that achieves this to be rotational by nature, where the user rotates their hand. Furthermore, creating intuitive interaction in this way while using hand gestures can lead to extremely good accuracy, recall, and precision when compared to more traditional methods, as shown by Rady et al [5]. When the entire detection rig rotates with the servo motor, the live video feed shows this update in real time. Figure 2C shows the new screen displayed once a user hovers one hand over the Leap Motion controller. If a user wishes to lock the servo motor, thus stopping the motor's rotation with this hand, they can perform a pinch gesture. This gesture is shown in Figure 2D. The pinch gesture was chosen for this task because it is an extremely easy gesture for people to make while distracted by other scenarios or occupied with spatial thinking, as they would be while simultaneously controlling the servo motor. Bailey showed that natural gestures, such as this pinching gesture, result in lower cognitive load and higher instructional efficiency for users [6]. In the context of this system, both of these features are important, as people may need some time to learn how to use an object-detection system such as this in high-pressured scenarios such as remote bomb diffusion. To unlock the servo motor, users simply just make another pinch gesture with the same hand and go back to rotating the detection rig by rotating their hand.

Figure 2C also shows that once a single hand is hovered over the Leap Motion controller, another instruction for users to hover their other hand over the Leap Motion is displayed. Figure 2E shows a user doing this, and Figure 2F shows the new screen that is displayed on the monitor once there are two hands hovering over the Leap Motion controller. As seen in Figure 2F, the user is told that the distance between both hands controls the chosen parameter. In the case of the distance parameter, which is chosen by default, the distance between the user's hands equals the distance for which the ultrasound sensor is "looking" for objects. The further apart a user's hands are, the further the ultrasound sensor looks for objects. The sensing distance which equals this metric is displayed on screen as shown in Figure 2F. The interaction of users altering the distance between their

hands to alter the distance that is sensed by the detection rig is a similar example of the natural and intuitive interaction ideology that was introduced with the rotation of the hand equaling the rotation of the servo motor. Again, it is natural for users to think about altering a physical distance with their hands in front of them when tinkering with a distance attribute in the system. There is an intuitive nature of reasoning here where few users would question why they are performing the action, unlike many other arbitrary interactions that may occur with traditional setups such as the keyboard and mouse. There is high affordance here, meaning that the user instinctively knows what to do, because the mapping of using physical distance in the real world to alter distance in the system is a natural one. When the user spreads their hands far enough where the sensing distance equals, or is greater than, the distance between an object and the ultrasound sensor then the screen displayed in Figure 2G is shown, where there is a notification on screen that an object has been detected (shown in blue box). This notification is shown in a blue box so it can be easily distinguished in user's peripheral vision, and the user does not have to alter their focus away from what they are currently doing. Furthermore, when an object is detected like this, a tone is played from the piezo buzzer to re-enforce the object detection without forcing the user to take their focus entirely away from the task at hand which most likely involves looking at the live video feed on the screen.

At this stage, the user can then save a screenshot of the live video feed by making a pinch gesture like the gesture seen in Figure 2D with the second hand they hovered over the sensor. Once an image is saved, the screen in Figure 2H is displayed with a short-lasting notification saying that a screenshot was saved. Moreover, the red LED controlled by one of the Arduino microcontrollers is turned on for a few seconds to indicate that an image has been saved. The LED is turned on to potentially alert others that the controller has saved an image. In certain contexts, such as operating this system located on a rover in outer space remotely, it is very possible that there are a group of people relying on knowing what the actions of the object-detection system pilot are while watching. Hence, having the LED turn on is a useful way to alert others that the operator has taken an important action, such as saving an image. Along with the image screenshot being saved, details relating to the object are also saved, as they may be useful metrics related to the object that people want to remember for later. These details are the object identifier (a number derived from the order of the screenshot being taken), the object's distance from the detection rig at the time of the image being saved (this distance equals the real distance from the detection rig to the object in the crosshair, not the sensing distance), and the angle of rotation that the servo motor was at when the image was saved. Figure 2I shows the details of an object that are saved in the software application on Processing. Users can save as many images as they like, where saved images along with object details are simply displayed below the main application area in order of their save time. The user can look at all of these saved images and their associated details by simply scrolling vertically via a mouse. This action of scrolling to previously saved images is the only interaction that requires users to touch something, but it is not crucial to the task of detecting objects with the detection rig and can be done after a session of detection.

To select a different parameter from the default parameter of distance, the user simply performs a circle gesture. This circle gesture is shown in Figure 3. Users can perform either a clockwise directed circle gesture, or an anti-clockwise directed circle gesture, and the direction of this circle gesture dictates what parameter is selected. In Figure 2A, the top right-hand portion of the screen shows the parameters available, with the red rectangle surrounding the distance parameter indicating that it is currently selected. The three parameters (distance, brightness, and saturation) are organised in a circle. This organisation is there to correlate with the circle gestures. If the user performs a clockwise circle gesture with any hand, the parameter selection moves to the next parameter in the clockwise direction. If the user performs an anti-clockwise circle gesture with any hand, the parameter selection moves to the next parameter in the anti-clockwise direction. So, if the distance parameter is currently selected, the user should perform a clockwise circle gesture to move the parameter selection to the brightness parameter. Likewise, if the distance parameter is currently selected, the user should perform an anti-clockwise circle gesture to move the parameter selection to the saturation parameter.

Figure 3: Circle Gesture with Leap Motion

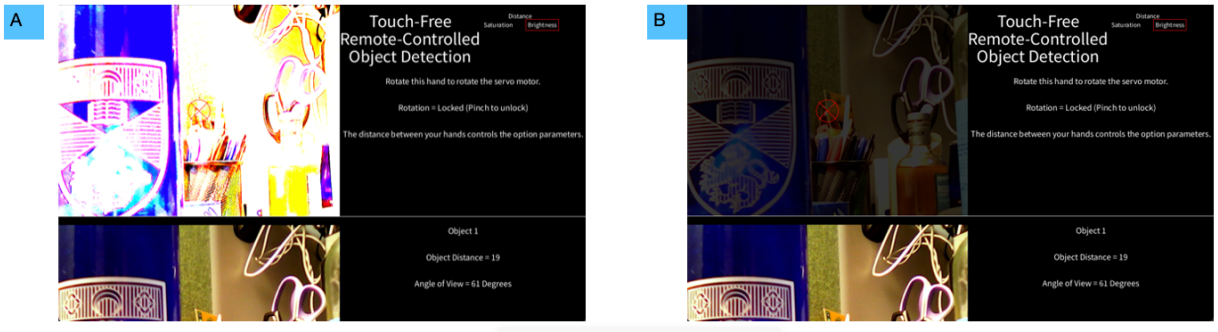

Figure 4: The Brightness Parameter

When the brightness parameter is selected, the interaction overview remains largely similar. The only aspects that change are the screens displayed to the user when two hands are hovered. When referring to Figure 2, Figure 2F and Figure 2G get replaced with Figure 4A and Figure 4B. With the brightness parameter the distance between the user's hands corresponds to the brightness of the live video feed, and the brightness of the video feed changes in real time as the user alters the distance between their hands. As a user's hands get further apart, the brightness of the live video feed increases. Figure 4A shows a high-brightness live video feed caused by the user keeping their hands far apart, and Figure 4B shows a low-brightness live video feed caused by the user keeping their hands closer together.

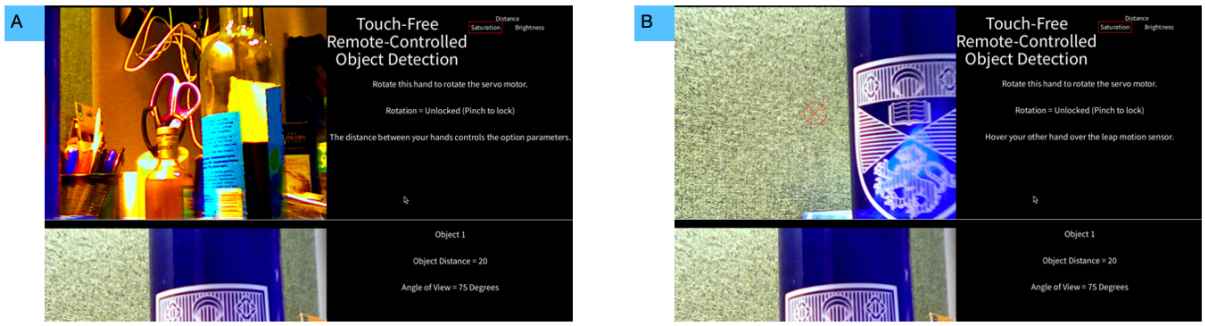

Figure 5: The Saturation Parameter

When the saturation parameter is selected, once again, the interaction overview remains largely similar. Similarly to the brightness parameter, the only aspects that change are the screens displayed to the user when two hands are hovered. When referring to Figure 2, Figure 2F and Figure 2G get replaced with Figure 5A and Figure 5B. With the saturation parameter the distance between the user's hands corresponds to the saturation of the images of the live video feed, and the saturation of the video feed changes in real time as the user alters the distance between their hands. To be clear, the saturation parameter in this system refers to the colour saturation of an image. It can also be described as the chromatic intensity of an image. As a user's hands get further apart, the saturation of the live video feed increases. Figure 5A shows a highly saturated live video feed caused by the user keeping their hands far apart, and Figure 5B shows a lowly saturated live video feed caused by the user keeping their hands closer together.

These interactions with the brightness and saturation parameters where users move their hands apart and towards each other, were chosen because it simulates a slider in mid-air, where the user can almost "feel" an invisible slider in front of them. This is an interesting and unique twist on the interaction offered by a familiar widget (the slider) where users can use their own bodies and almost feel the interaction instead. This is a great example of the novel and intuitive interactions that a controller like the Leap Motion controller can offer.

Parts List

Hardware	Other
• Leap Motion Controller x1 • Camera x1 • Arduino Uno Microcontroller x2 • Arduino USB Power Cable x2 • Breadboard x1 • Servo Motor SG90 x1 • Ultrasonic Sensor (HC-SR04) x1 • Passive/Piezo Buzzer x1 • Red LED x1 • 220 Ohm Resistor x1 • Breadboard Cable x16 • Display Screen x1	• Arduino IDE • Processing + Processing IDE • Leap Motion Software Developer Kit • Leap Motion for Processing Software Library

Process

To recreate this project, the first thing to do is to create the following circuits where the various hardware components are connected to the pins on the Arduino microcontrollers and breadboards.

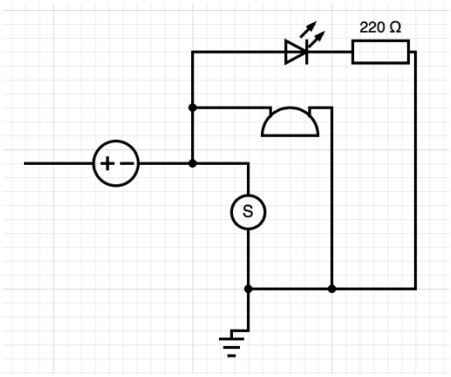

Figure 6: Circuit Diagram for Servo Motor Circuit

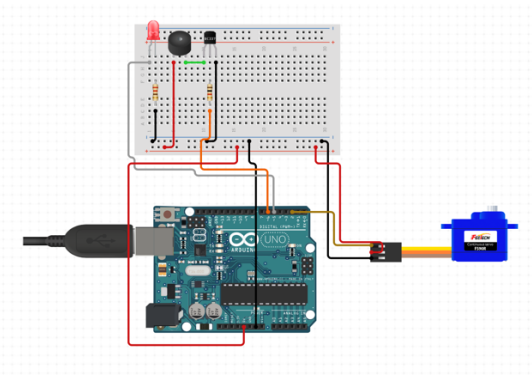

Figure 7: Example Wiring Diagram for Servo Motor Arduino

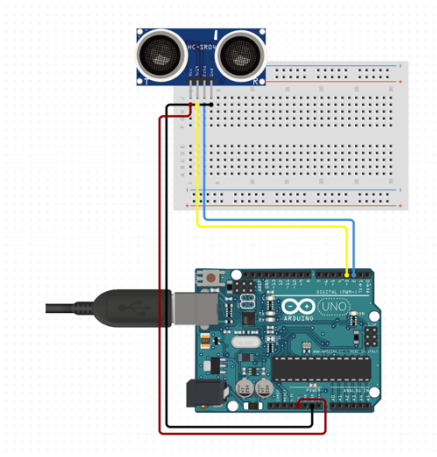

Figure 8: Example Wiring Diagram for Ultrasonic Arduino

One of the circuits (one of the Arduino microcontrollers) is solely responsible for managing the ultrasonic sensor and delivering its measurements to Processing over serial communication (shown in Figure 8). This functionality is required in an entirely separate circuit because the Arduino microcontroller implements a delay of 1 second overall for every reading that the ultrasonic sensor records. This is to smooth the readings of the ultrasonic sensor. If this delay was present in the other Arduino microcontroller circuit (shown in Figures 6 & 7) that controls the servo motor, the piezo buzzer, and the LED, then those delays would cause havoc with the interaction between users and the servo motor where instant and continuous communication without delays is required.

The hardware components connected to the Arduinos act as both inputs and outputs to the overall system and the software located in Processing. The ultrasonic sensor acts as an input that provides distance measurements. The code for the Arduino that controls this sensor can be seen in Appendix A. The other Arduino controls the output hardware components of the system, which are the servo motor, the red LED, and the piezo buzzer. The code that should be uploaded to this Arduino can be seen in Appendix B. The servo motor is used to

rotate the detection rig at precise angles reflective of the exact angle that users hold their hands at. A servo motor is an ideal motor for this exact function. The red LED is used to provide some output from the system that indicates that an image has been saved in the software running on processing. As mentioned before, this LED is implemented with consideration to the potential context that a system like this may be used in, where others in a group may find it useful to know when an important action is taken by the operator, such as saving an image on the system. The piezo buzzer is used as an output speaker that continuously plays sound when a user has detected an object at a certain distance. This addition of sound provides immediate feedback to users without taking their attention away from the task at hand while looking at the screen.

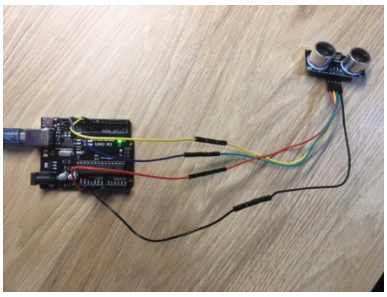

Figure 9: My Exact Ultrasound Arduino Setup

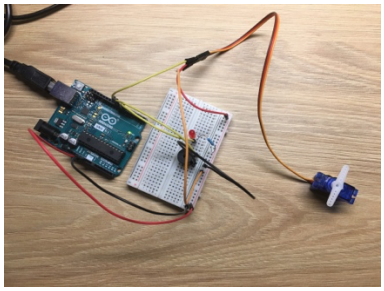

Figure 10: My Exact Servo Motor Arduino Setup

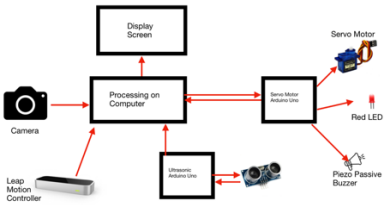

Figure 11: System Overview Diagram

At this point with most of the hardware setup, it's time to focus on developing the software for the detection system in Processing. The next step is to connect a camera (such as a webcam) via a USB cable that Processing can communicate with via serial communication. This camera provides the crucial live video feed that allows the system to be remote-controlled.

One of the most crucial parts of the software is to read the incoming data from the Leap Motion controller and perform actions accordingly. To use the Leap Motion as an input device for Processing, the Leap Motion software developer kit [7] must be downloaded in order for the computer to recognise the device, and the Leap Motion processing library [8] must also be downloaded so that the inbuilt code for Leap Motion can be used to allow for more efficient coding. After this, the Leap Motion controller is simply plugged in to the computer with a USB cable to be read over serial communication. The Leap Motion controller is capable of detecting how many hands are hovered over the sensor, what the positions of the hands and their respective fingers are, and what gestures the hands are making at any given time. In the Processing code that is provided with this report and project, it can be seen that the actions and instructions displayed to the user first rely on how many hands are present over the Leap Motion controller. When one hand is present, only servo motor actions are allowed. When two hands are present then the parameter controls and ability to save images are also allowed. The positions of both hands above the sensor are used to calculate the distance between the user's hands that dictates the parameter controls. The parameter selection is changed whenever the Leap Motion controller detects a circle gesture from any hand. The Java function responsible for this can be seen in Appendix C.

The code responsible for detecting objects at certain distances can be seen in Appendix D. Upon looking at the code, it can be seen that the distance between the user's hands is continuously compared to the actual distance recorded by the ultrasonic sensor.

The code responsible for altering the brightness of the live video feed from the camera can be seen in Appendix E. Upon looking at the code, it can be seen that the brightness is in fact altered by first reading every pixel from every image of the video feed, multiplying the intensity of each red, green, and blue pixel value with the new brightness value obtained from the distance between the user's hands, and then updating the pixels for every video frame with these values by their location on the entire application window.

The code responsible for altering the (colour) saturation of the live video feed from the camera can be seen in Appendix F. Upon looking at the code, it can be seen that the saturation is in fact altered by first reading the HSB (Hue, Saturation, Brightness) value for every pixel from every image of the video feed, multiplying the intensity of the saturation pixel value with the new saturation value obtained from the distance between the user's hands, and then updating the pixels for every video frame with these values by their location on the entire application window.

After completing these steps, the touch-free remote-controlled object detection system should have the system overview that is shown in Figure 11.

Challenges, Key Points & Conclusion

Overall, this touch-free remote-controlled object detection system allows users to achieve many tasks and interactions based on a small number of simple intuitive interactions. Furthermore, these interactions are all touch-free, and the system can be controlled remotely. For these reasons, this system offers interesting insight into using these intuitive interactions via the Leap Motion controller to attempt to reduce cognitive load on users that could otherwise be present in more traditional methods of interaction for similar systems (via joysticks to control a rover on Mars etc.).

One of the main challenges of this project involved calibrating the range of motion that the Leap Motion should read from user's hands. At first, the range of distance available for users to separate their hands without being lost by the Leap Motion provided little scope for altering the parameters. It was either too sensitive or not sensitive enough to the user input. Getting this right for the expected distance range (0cm – 100cm), brightness range, and saturation range, required some attention. One possible critique of this system that could be improved in future work, is that the ranges with which users can alter parameters are fairly fixed. For example, if the detection system was built for detecting objects in 0m – 1km range, the sensitivity for users to accurately detect objects may suffer.

Avenues for future work include building a more robust detection rig, that can provide better accuracy and stability. It may be intriguing to try and build movement into the detection rig also, in similar fashion to a car, and see if the intuitive touch-free interactions can also be used to move the vehicle of the detection rig. Another avenue for future work would be exploring more parameters for users to tweak. For example, it would be a natural extension to also implement zoom capability using the same style of interaction implemented for brightness/saturation if the camera had zoom capability. Finally, it would also be important to conduct future user research into exploring which specific hand gestures/movements result in the lowest cognitive load for these particular tasks.

References

[1] "Leap Motion Controller," Ultraleap, [Online]. Available: https://www.ultraleap.com/product/leap-motion-controller/. [Accessed 08 May 2022].

[2] "Nasa Mars Exploration Rovers," NASA, [Online]. Available: https://mars.nasa.gov/mer/. [Accessed 08 May 2022].

[3] "Processing," Processing Foundation, [Online]. Available: https://processing.org. [Accessed 08 May 2022].

[4] "Arduino Microcontrollers," Arduino, [Online]. Available: https://www.arduino.cc. [Accessed 08 May 2022].

[5] M. A. Rady, S. M. Youssef and S. F. Fayed, "Smart Gesture-based Control in Human Computer Interaction Applications for Special-need People," in *2019 Novel Intelligent and Leading Emerging Sciences Conference (NILES)*, Cairo, Egypt, 2019.

[6] S. Bailey, "Getting the Upper Hand: Natural Gesture Interfaces Improve Instructional Effifficiency on a Conceptual Computer Lesson," *Electronic Theses and Dissertations: University of Central Florida,* 2017.

[7] "Ultraleap For Developers," Ultraleap, [Online]. Available: https://developer.leapmotion.com. [Accessed 08 May 2022].

[8] D. Morawiec, "Leap Motion for Processing," [Online]. Available: https://github.com/nok/leap-motion-processing. [Accessed 08 May 2022].

Appendix A: Ultrasonic Arduino Code

```
// Arduino Code for Touch-Free Remote-Controlled Object Detection
// For the Ultrasound Arduino
// CS5041 P3 Special Project
// Author = 200036885

#define trigPin 10
#define echoPin 13

// Time spent for ultrasound signal to be sent and received
float duration;
// Calculated distance to object
int distance;

// ------------------------------------------------

void setup() {
  // Setup serial communication
  Serial.begin(9600);
  // Setup pins for the ultrasound sensor
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}

// ------------------------------------------------


void loop() {

  // Send an initial ultrasound pulse
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // Measure the duration of the pulse
  duration = pulseIn(echoPin, HIGH);

  // Calculate the distance based on speed of sound in air
  distance = (int) ((duration/2)*0.0343);

  // Ultrasound sensor only measures from 2cm to 400cm
  // So, eliminate all other readings
  if ((distance <= 2) || (distance >= 400)) {
    // Send a default message if there are no readings
    Serial.println("NA");
  } else {
    // Send the measured distance over serial
    Serial.println(distance);
    // Delay to get smoother readings
    delay(500);
  }

  // Another delay to get even smoother readings
  delay(500);

}
```

Appendix B: Servo Motor Arduino Code

```
// Arduino Code for Touch-Free Remote-Controlled Object Detection
// For the Servo Motor Arduino
// CS5041 P3 Special Project
// Author = 200036885

#include <Servo.h>

// Servo Motor
Servo myServo;

// For reading messages from Processing
byte incomingByte;

// LED for indicating that an image has been saved
const int redLEDPin = 7;

// ---------------------------------------------

void setup() {
  // Setup LED for indicating saved images
  pinMode(redLEDPin, OUTPUT);
  digitalWrite(redLEDPin, LOW);
  // Setup servo motor
  myServo.attach(9);
  Serial.begin(9600);
  myServo.write(0);
}

// ---------------------------------------------


void loop() {
  // Angle for the servo motor
  byte angle;
  if (Serial.available()) {
    // Read message from Processing
    incomingByte = Serial.read();
    if (incomingByte == byte(998)) {
      // Turn on LED
      digitalWrite(redLEDPin, HIGH);
    } else if (incomingByte == byte(999)) {
      // Turn off LED
      digitalWrite(redLEDPin, LOW);
    } else if (incomingByte == byte(997)) {
      // Play sound at 1000Hz from the piezo buzzer
      tone(8, 1000, 50);
    } else {
      // Turn servo motor to the angle sent from Processing
      angle = incomingByte;
      myServo.write(angle);
    }
    Serial.println(angle);
  }
}
```

Appendix C: Leap Motion Circle Gesture Detection Code

```
// This function is called whenever a circle gesture is recognised from the leap motion sensor
void leapOnCircleGesture(CircleGesture g, int state) {
  float durationSeconds = g.getDurationInSeconds();
  //println(durationSeconds);
  if ((!gestureMade) && (durationSeconds > 0.1)) {
    println("CIRCLE GESTURE DETECTED");
    int direction = g.getDirection();
    if ((parameterSelection == 1) && (direction == 1)) {
      // Clockwise from distance
      parameterSelection = 2;
    } else if ((parameterSelection == 2) && (direction == 1)) {
      // Clockwise from brightness
      parameterSelection = 3;
    } else if ((parameterSelection == 3) && (direction == 1)) {
      // Clockwise from saturation
      parameterSelection = 1;
    } else if ((parameterSelection == 1) && (direction == 0)) {
      // Anti-Clockwise from distance
      parameterSelection = 3;
    } else if ((parameterSelection == 2) && (direction == 0)) {
      // Anti-Clockwise from brightness
      parameterSelection = 1;
    } else if ((parameterSelection == 3) && (direction == 0)) {
      // Anti-Clockwise from saturation
      parameterSelection = 2;
    }
    gestureMade = true;
    gestureTimer = 120;
  }
}
```

Appendix D: Distance Parameter Code in Processing

```
// Distance between both hands is used for all mappings for all parameters
distanceBetweenHands = (int) dist(firstHand.getPosition().x, firstHand.getPosition().y, secondHand.getPosition().x, secondHand.getPosition().y);

if (parameterSelection == 1) {

  // DISTANCE PARAMETER

  int sensorDistance = (int) map(distanceBetweenHands, 150, 1000, 2, 100);
  if (sensorDistance > 5) {
    text("Sensing at a distance of: " + sensorDistance, 920, 360);
    if (sensorDistance >= parseUltrasoundDistance(ultrasoundDistance)) {
      fill(0, 0, 255);
      rectMode(CENTER);
      rect(920, 420, 300, 50);
      fill(255);
      text("OBJECT DETECTED", 920, 425);
      servoPort.write(997); // Send a message to the Servo Arduino that plays the piezo buzzer sound
    }
  }
```

Appendix E: Brightness Parameter Code in Processing

```
} else if (parameterSelection == 2) {

  // BRIGHTNESS PARAMETER

  loadPixels();

  for (int x = 0; x < video.width; x++ ) {
    for (int y = 0; y < video.height; y++ ) {

      // Calculate the pixel index for 1D array
      int pixelIndex = x + y*width;

      // Get the R,G,B values from pixel
      float r = red(pixels[pixelIndex]);
      float g = green(pixels[pixelIndex]);
      float b = blue(pixels[pixelIndex]);

      // Brightness adjustment
      float adjustBrightness = map(distanceBetweenHands, 150, 1000, 0, 8);

      // New brightness
      r *= adjustBrightness;
      g *= adjustBrightness;
      b *= adjustBrightness;
      r = constrain(r, 0, 255);
      g = constrain(g, 0, 255);
      b = constrain(b, 0, 255);

      // Replace pixel value with this new colour
      color c = color(r, g, b);
      pixels[pixelIndex] = c;
    }
  }

  updatePixels();
```

Appendix F: Saturation Parameter Code in Processing

```
} else if (parameterSelection == 3) {

  // SATURATION PARAMETER

  loadPixels();
  colorMode(HSB, 255);

  for (int x = 0; x < video.width; x++ ) {
    for (int y = 0; y < video.height; y++ ) {

      // Calculate the pixel index for 1D array
      int pixelIndex = x + y*width;

      // Get the H,S,B values from pixel
      color c = pixels[pixelIndex];
      float hue = hue(c);
      float saturation = saturation(c);
      float brightness = brightness(c);

      // Saturation adjustment
      float adjustSaturation = map(distanceBetweenHands, 150, 1000, 0, 8);

      // New saturation
      saturation *= adjustSaturation;
      saturation = constrain(saturation, 0, 255);

      // Replace pixel value with this new colour
      color newC = color(hue, saturation, brightness);
      pixels[pixelIndex] = newC;
    }
  }

  updatePixels();
  colorMode(RGB);
```