Group 6 Project Report

Valentijn Giesbers (2028584), Luca Simonetti (2059256), Dominic Stamatoiu (2051167), Gabriela Vlahova (2063060)

Introduction

Nowadays robots are becoming an essential part of our society. It is not science fiction anymore, it is a reality. There are different types of robots such as behaviour-based robots, humanoid robots, cognitive robots and social robots. Social robots are meant to engage humans in a natural, interpersonal manner - frequently to accomplish beneficial outcomes in a variety of applications such as education, health, quality of life, and activities that need collaborative effort. In order to give meaningful social and task-related help to individuals, robots will need to engage us not only cognitively, but also emotionally (Breazeal et al., 2016). With the advance in robotic technology, human-computer interaction plays an important role in successfully developing a robot. Active decision making that takes human partners into account is required for robots to engage with people autonomously (Bütepage & Kragic, 2017).

Building the interaction between humans and robots is an essential part of the developing process. The field of study devoted to understanding, constructing and evaluating human-centred robotic systems is called Human-Robot Interaction. Human factors experts may clearly benefit from a better grasp of dynamics, control, and AI, but they should also look for methods to engage with engineers in these domains on research, conceptual design and assessment (Sheridan, 2016).

In this project, we used Arduino software, which provides an easy-to-use development environment as well as a variety of hardware and software resources. This enables rapid project development. It is a platform that includes a basic microcontroller and an interface development environment for developing apps that will be downloaded to the board. Arduino is a card that can handle digital and analogue signals. The embodiment takes an important part when it comes to the development of a successful robot, especially in socially interactive robots. This class includes robots that are capable of interacting socially with people, but also their social talents to achieve their desired duties.

Theoretical implementation

To get the robot to be more social in its appearance and behaviour we looked at the ways humans express themselves. The most common ways are through motion, language, sound and facial expressions. There were a few limitations on the feasibility of some ideas. Getting the robot to use facial expressions or human language was not possible. Since this agent is involved in a game we wanted to express the basic emotions involved when playing a game. Being happy and excited while the game is going on but also a bit of disappointment when the game is lost. If we can express this type of behaviour people will feel a better connection to the agent because they identify with those types of expressions during a game. That is why we chose for a rewarding sound, namely the sound from getting a coin in a mario game, when the agent was doing well and a game-over melody accompanied with the agent turning in its place when it lost the game.

Furthermore, appearance also plays a big role in human-computer interaction. Something familiar brings a better connection than something unknown like an arduino set up with a few cables and sensors. Making it too

familiar repulses people because of an effect known as the Uncanny Valley.

This effect states that the more humanoid a robot is the more familiar people are with it until it becomes close to a real human but not completely. This becomes eerie and uncomfortable to look at (Mori, MacDorman, & Kageki, 2012). This will not be feasible with this project because of multiple limitations but still an interesting phenomenon to keep in mind when building a social robot.

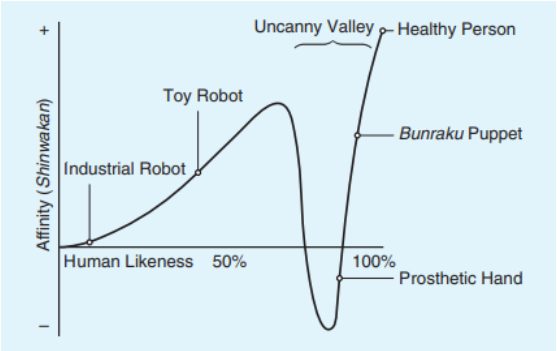


Figure 1. Affinity to Human likeness graph (Mori, MacDorman, & Kageki, 2012)

The agent will be used in a game that is supposed to be fun, because of that the interaction with it would benefit if the appearance was seen as cute. That is why we chose to convert the robot to a stuffed racoon and gave it the name Rocket Raccoon as a reference to the raccoon from the Marvel Cinematic Universe (see appendix A, Picture 1-4).

The game compliments the agent's familiarity because a real life wild raccoon does not want to be touched by a human, which is the basic idea of the game it will be playing. The game is a variant of the well known game tag, the agent will be in the middle of a group of human players. It will start driving forward until it gets close to a player, then turn around and drive away (see appendix A, video 1, 00:00-00:17). This will go on until a player touches the raccoon and the game is over (see appendix A, video 1, 00:17-00:21).

Practical implementation

Physically the agent consists mostly of arduino parts. The core part of the agent is the arduino Uno, the CPU of the robot. Connected directly to this is a 9 Volt battery to power it and there is a USB port so that new software can be written onto the board.

The agent is multimodal, it consists of multiple input and output sensors. To easily connect these sensors to the arduino, a breadboard is used. Four double A batteries are connected to the breadboard to power the sensors. One of the input sensors is an ultrasonic sensor that sends out high frequency soundwaves and measures when that sound bounces back to the sensor. It records the time it takes from sending out the wave to receiving an echo back. With this value it calculates the distance to the object in front of it in the *microsecondsToCentimeters* function (see appendix B, line 32 & 33). This is the way the agent can see and thus this sensor is mounted at the front of the agent. The other input sensor is a touch sensor. This has an open electrical loop that is closed when touched by, for example, our fingers because they are conductive. When the loop is closed a current will flow and the agent will get input (For a schematic overview of the hardware see Appendix C, Figure 1).

The agent will generate different outputs depending on what input it received. One of the outcomes can be turning and driving away. This is done by two servo motors on either side of the agent with a wheel attached to that motor. By powering only one motor and thus turning one wheel, the agent will revolve around its own axis. This way the agent can turn around when the sonar senses a player in front of itself before turning both motors on and driving forwards in a different direction. This way of revolving around its own axis is also used when the agent is touched and loses the game. It will momentarily spin around in place. This is accompanied by the

other output sensor, a buzzer. When the agent loses the game the buzzer will make a sound with different frequencies. During the game the buzzer imitates the sound of getting a coin in a mario game when it successfully turns away from a human player.

The game can be started by powering on the robot. This is when the software starts running and the sensors are turned on. The sonar will start measuring the distance from the front of the robot to the object (human player) in front of it and the wheels will both start turning (see appendix B, lines 44-58). The sonar will print the distance measured to the serial monitor so that developers can see if everything works. If the measured distance is shorter than the threshold the *turn()* function will be called. One of the motors will be turned off for a random amount of time between 600 and 1101 milliseconds to ensure that the turns the robot makes are varied and somewhat unpredictable. We found these values through repeated testing. Both motors will be turned on after this and the *sing()* function will be called which plays the mario coin sound (Dumais, 2019). This code has been adapted to fit our agent. (see appendix B, lines 107-143).

This loop of driving while measuring the distance and acting upon that distance keeps going while the touch sensor is constantly checked (see appendix B, lines 71-101). The state of the touch sensor is also sent to the serial monitor to promote ease of debugging. If the touch sensor measures an input, the game-over part of the game will start which turns one motor off and activates the buzzer which plays a game-over tune.

Discussion

During the building and testing phase of this project we came to find out that using only the buzzer as an output when the agent lost the game did not feel very social. It felt very bland and not how we had envisioned it. That is the reason we made the agent spin around its axis, for an added dramatic effect. This made the agent's behaviour when losing more enjoyable and familiar. The buzzer output itself also had to be tuned after the first tests to make it sound dramatic enough while also making sure that the tones were not too annoying to hear.

The appearance of an agent plays a big role in how we interpret it. We did not think of this when writing down ideas and building prototypes. During testing and building the disconnect with the agent became clear and we added the stuffed raccoon to completely overhaul the looks. This greatly increased the social aspect of the agent because now most of the exposed hardware is covered by the raccoon which is an embodiment that people are familiar with.

References

Breazeal, C., Dautenhahn, K., & Kanda, T. (2016). Social Robotics. In B. Siciliano & O. Khatib (Eds.), Springer Handbook of Robotics (pp. 1935–1972). Springer International Publishing. https://doi.org/10.1007/978-3-319-32552-1_72

Bütepage, J., & Kragic, D. (2017). Human-Robot Collaboration: From Psychology to Social Robotics (arXiv:1705.10146). arXiv. https://doi.org/10.48550/arXiv.1705.10146

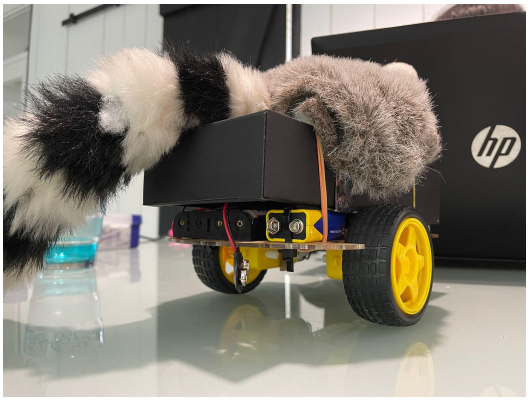
Dumais, D. (2019, 28 september). How To Make The Super Mario Coin Sound Effect. https://www.daviddumaisaudio.com/how-to-make-the-super-mario-coin-sound-effect/

Mori, M., MacDorman, K., & Kageki, N. (2012). The Uncanny Valley [From the Field]. IEEE Robotics & Automation Magazine, 19(2), 98–100. https://doi.org/10.1109/mra.2012.2192811

Plaza, P., Sancristobal, E., Carro, G., Blazquez, M., García-Loro, F., Martin, S., Perez, C., & Castro, M. (2018). Arduino as an Educational Tool to Introduce Robotics. 2018 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE), 1–8. https://doi.org/10.1109/TALE.2018.8615143

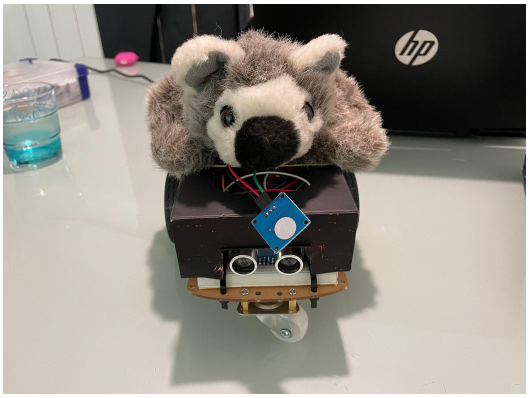
Sheridan, T. B. (2016). Human–Robot Interaction: Status and Challenges. Human Factors: The Journal of the Human Factors and Ergonomics Society, 58(4), 525–532. https://doi.org/10.1177/0018720816644364

Appendix A

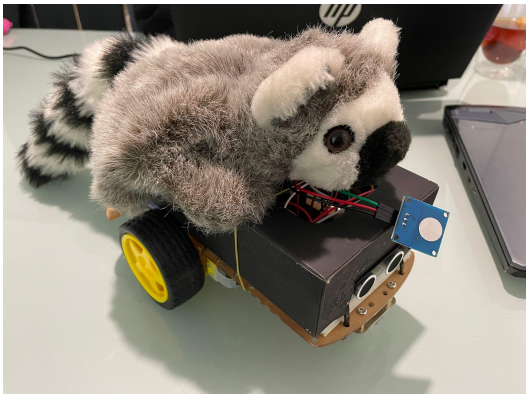


Picture 1.

Picture 2.



Picture 3.



Picture 4.

Video 1.

https://youtube.com/shorts/P6lkvhARYcE?feature=share

Appendix B

```
//#include <toneAC.h>

const int pingPin           = 7;  // Trigger Pin of Ultrasonic Sensor      pin 7
const int echoPin           = 6;  // Echo Pin of Ultrasonic Sensor         pin 6
const int threshold         = 40; // The threshold for the distance at which to turn
const int touch             = 5;  // Touch Sensor readout                  pin 5
const int buzzer            = 10; // Buzzer to arduino                     pin 10

#define NOTE_B4  494
#define NOTE_E7  2637
#define melodyPin 10

// Mario coin sound melody
int melody[] = {
  NOTE_B4, NOTE_E7
};

int tempo[] = {
  25, 4
};

void setup() {
  pinMode(11, OUTPUT);
  pinMode(12, OUTPUT);
  pinMode(pingPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(touch, INPUT);
  pinMode(buzzer, OUTPUT);
  Serial.begin(9600); // Starting Serial Terminal (communication with the computer)
}

long microsecondsToCentimeters(long microseconds) { // Function to translate duration (ms) to distance (cm)
   return microseconds / 29 / 2;
}

void turn() {
  digitalWrite(11, LOW);
  digitalWrite(12, HIGH);
  delay(random(600, 1101));
}
```

```

void loop() {
                                                          //SONAR
  long duration, cm;
  digitalWrite(pingPin, LOW);
  delayMicroseconds(2);
  digitalWrite(pingPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(pingPin, LOW);
  duration = pulseIn(echoPin, HIGH);
  cm = microsecondsToCentimeters(duration);
  Serial.print("Distance in cm: ");
  Serial.println(cm);


                                                          // DRIVING
  digitalWrite(12, HIGH);
  digitalWrite(11, HIGH);
                                                          //TURN IF OBJECT CLOSEBY
  if (cm < int(threshold)){
    turn();
    digitalWrite(12, HIGH);
    digitalWrite(11, HIGH);
    sing();

}


                                                          // Touch Sensor & Game Over Sound
  int touchState = digitalRead(touch); // read new state

  if (touchState == HIGH) {
    Serial.println("Touched!");
    digitalWrite(11, LOW);
    digitalWrite(12, HIGH);
    for (int y = 0; y < 4; y++) {
      for (int x = 0; x < 180; x++) {
        //convert angle of sinusoidal to radian measure
        float sinVal = (sin(x * (3.1412 / 180)));
```

```
        //generate sound of different frequencies by sinusoidal value
        int toneVal = 2000 + (int(sinVal * 1000));
        //Set a frequency for Pin-out 8
        tone(buzzer, toneVal);

        if (y == 3) {
          for (int z = toneVal; z > 400; z = z - 300) {
            tone(buzzer, z);
          }
        }
        delay(2);
      }
    }
    noTone(buzzer);
    digitalWrite(12, LOW);
    digitalWrite(11, LOW);
    exit(0);
  }

  else {
    Serial.println("Not touched.");
    }

delay(100);
}

void sing() {
  Serial.println("Coin Sound");
  int size = sizeof(melody) / sizeof(int);
  for (int thisNote = 0; thisNote < size; thisNote++) {

      // to calculate the note duration, take one second
      // divided by the note type.
      //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
  int noteDuration = 1000 / tempo[thisNote];

  buzz(melodyPin, melody[thisNote], noteDuration);

      // to distinguish the notes, set a minimum time between them.
      // the note's duration + 30% seems to work well:
  int pauseBetweenNotes = noteDuration * 1.30;
  delay(pauseBetweenNotes);

      // stop the tone playing:
   buzz(melodyPin, 0, noteDuration);
  }
}

void buzz(int targetPin, long frequency, long length) {
  digitalWrite(13, HIGH);
  long delayValue = 1000000 / frequency / 2; // calculate the delay value between transitions
  //// 1 second's worth of microseconds, divided by the frequency, then split in half since
  //// there are two phases to each cycle
  long numCycles = frequency * length / 1000; // calculate the number of cycles for proper timing
  //// multiply frequency, which is really cycles per second, by the number of seconds to
  //// get the total number of cycles to produce
  for (long i = 0; i < numCycles; i++) { // for the calculated length of time...
    digitalWrite(targetPin, HIGH); // write the buzzer pin high to push out the diaphram
    delayMicroseconds(delayValue); // wait for the calculated delay value
    digitalWrite(targetPin, LOW); // write the buzzer pin low to pull back the diaphram
    delayMicroseconds(delayValue); // wait again or the calculated delay value
  }
  digitalWrite(13, LOW);

}
```

Appendix C

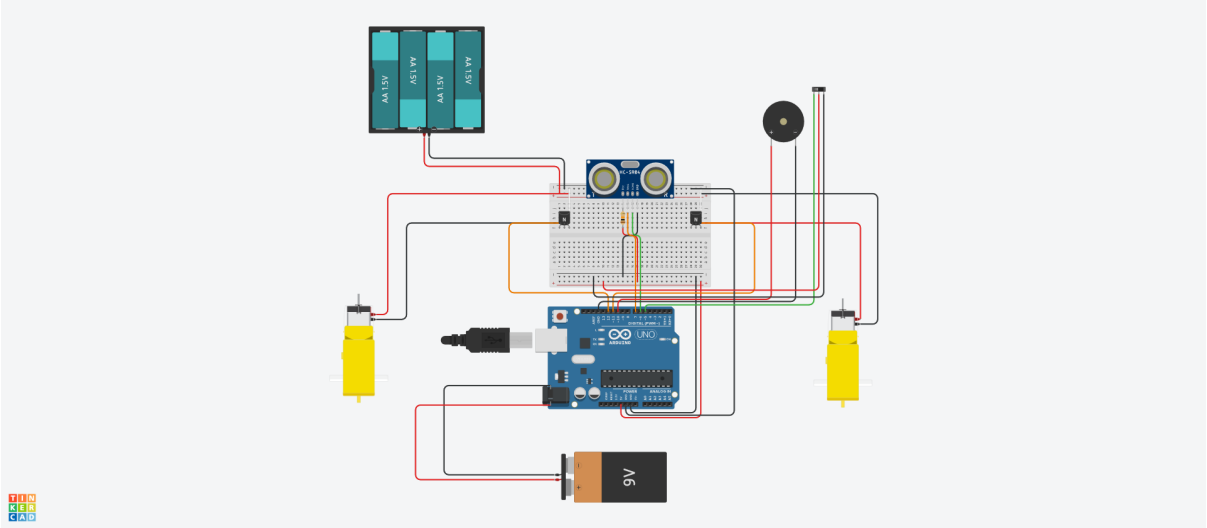


Figure 1.

Notes on figure 1:

It was not possible to connect the 9V battery to the Arduino directly in the software we used to make the schematic. So we placed a small diode behind the Arduino board to be able to connect the battery wires.

There was no touch sensor available so we used a switch (top right) to replace it in the schematic.

Both of these adaptations are only to make the schematic complete.