

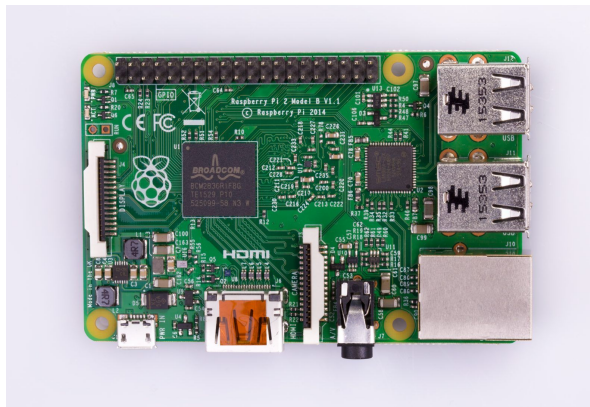
Hide-and-Seek Robot

Ari Gilmore and Christina Last

Introduction

Many similarities of the hide-and-seek game can be found in the interactions between humans, or robots and humans, in urban spaces, similar to looking for a person in a crowded urban environment (Goldhoorn *et al.* 2014). It is suggested that the game of hide-and-seek is an ideal domain for studying cognitive functions in robots and it is a basic mechanism for human robot interaction in mobile robotics, because hide-and-seek requires the robot to navigate, search, interact on, and predict actions of the opponent (Johansson and Balkenius 2005).

In this project we demonstrate an ability to program on Arduino & Raspberry Pi and communicate between the two systems, program in Python, focusing on OpenCV, and implement deep learning models on the robots to identify faces from purely visual inputs.



Concept

The concept is as follows: the project uses machine learning classification techniques to successfully find a hider by identifying the hider's face, and move towards the human (the robot's method of seeking). The visual information captured by the Pi camera is used to train the robot to find a hider. Through the artistic novelty of human-robot playing we explore the real-world implications in supporting human disaster response teams. For example, one of the many practical applications of this experiment is to find persons; as hide-and-seek could be seen as a simplification of search-and-rescue.



Implementation Process

There are two players, the hider (a person) and the seeker (the robot) and they play the game in free space. The game of hide and seek runs for a maximum number of timesteps. Within the number of time steps, the hider will attempt to move past the robot and collect a reward (perhaps enter a certain location in the room, or collect an item the robot is guarding). At each time step, the seeker will scan the room and capture images during the 360 degrees rotation. It will run a machine learning classification algorithm to identify faces from the captured images. Each image is serialised and is described by the angle at which the image was captured. When a face is identified, the robot chassis moves to the correct angle of orientation and forward in attempt to catch the hider. The process is repeated until the hider is successfully caught, or the hider wins.

```

angle_spin_organize | Arduino 1.6.0

angle_spin_organize  TurnSensor.cpp  TurnSensor.h

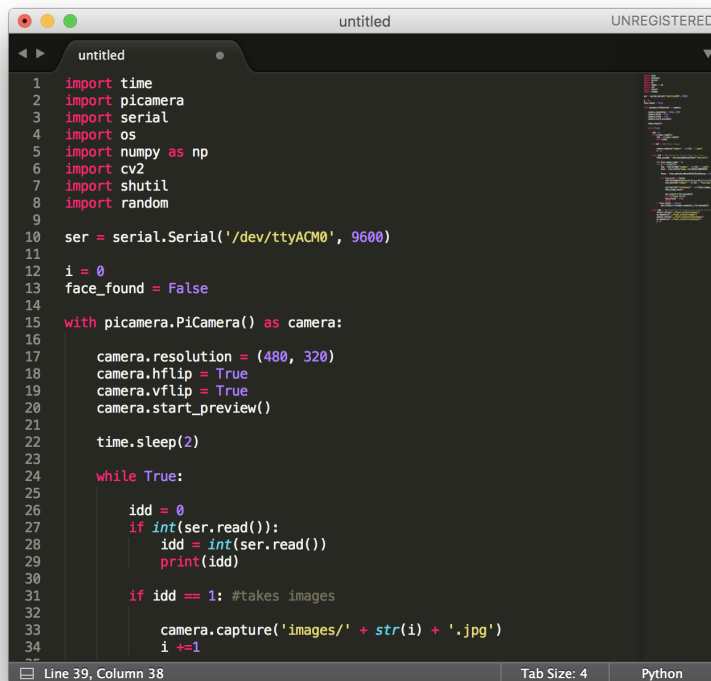
// --- Main program loop
void loop() {
  if(steps == 0){
    turnSensorUpdate();
    int32_t angle = getAngle();
    lcd.setCursor(0,0);
    lcd.print(angle);
    lcd.print(" ");
    if(angle % 30 == 0){
      //lcd.print("here");
      motors.setRightSpeed(0);
      Serial.write('1');
      delay(2000);
      motors.setRightSpeed(75);
      delay(20);
    }
    else{
      motors.setRightSpeed(75);
    }
  }
  if(angle < -1 && angle > -3){
    steps++;
    motors.setRightSpeed(0);
    for(int i=0; i<5; i++){
      Serial.write('2');
    }
    delay(4000);
  }
}

if(steps == 1){
  if(Serial.available() > 0){
    value = int(Serial.read())-48;
    //value2 = int(char(value));
    lcd.setCursor(0, 0);
    lcd.print(value);
    lcd.print(" ");
    steps++;
  }
}

if(steps == 2){
  turnSensorUpdate();
  int32_t angle = getAngle();
  if(value < 7){

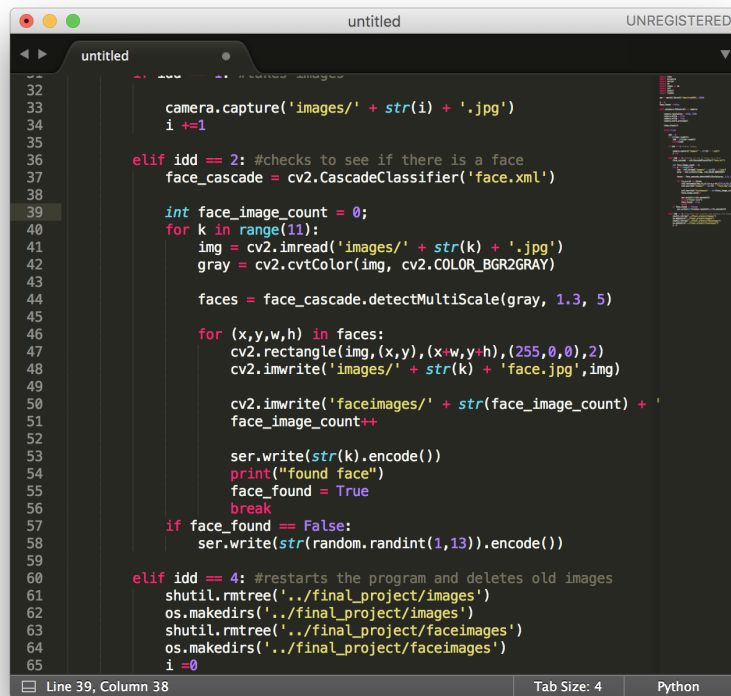
```

Part of Ardunio code where one can see how we implemented the angles that the robot turned at and the signals it sent to the Raspberry Pi.



```
1 import time
2 import picamera
3 import serial
4 import os
5 import numpy as np
6 import cv2
7 import shutil
8 import random
9
10 ser = serial.Serial('/dev/ttyACM0', 9600)
11
12 i = 0
13 face_found = False
14
15 with picamera.PiCamera() as camera:
16
17     camera.resolution = (480, 320)
18     camera.hflip = True
19     camera.vflip = True
20     camera.start_preview()
21
22     time.sleep(2)
23
24     while True:
25
26         idd = 0
27         if int(ser.read()):
28             idd = int(ser.read())
29             print(idd)
30
31         if idd == 1: #takes images
32
33             camera.capture('images/' + str(i) + '.jpg')
34             i +=1
```

Part 1 of Python Code:
Reading in the serial number from the Arduino, so the Raspberry Pi knows when to take an image.

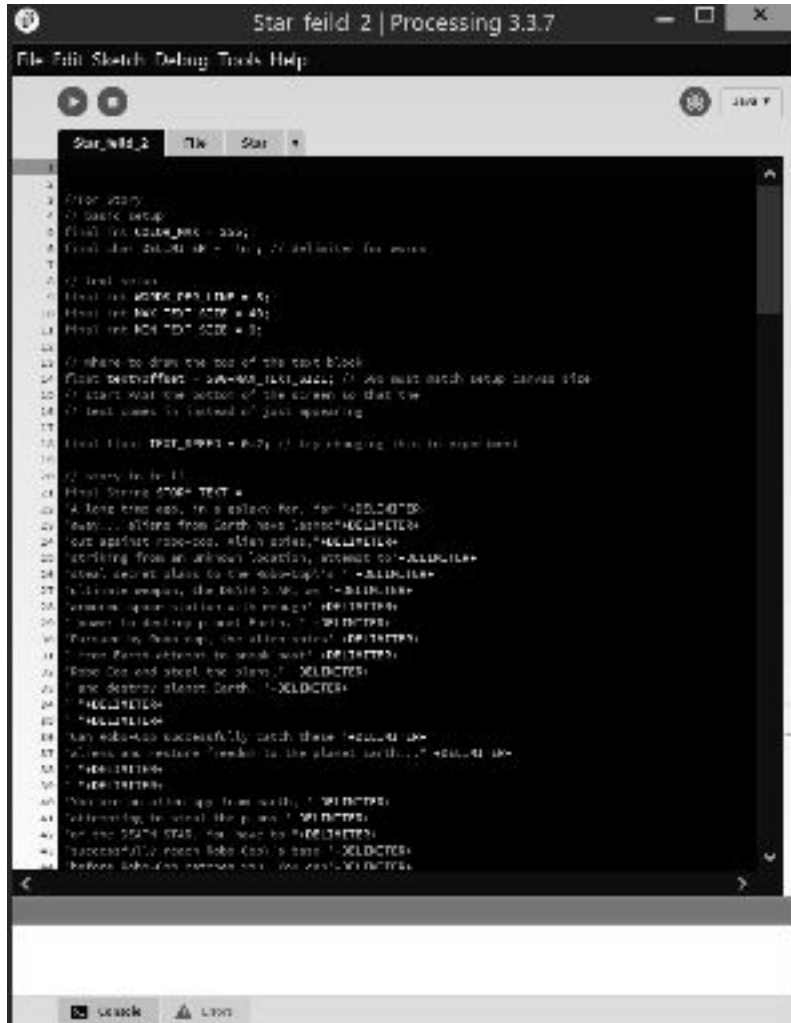


```
32 camera.capture('images/' + str(i) + '.jpg')
33 i +=1
34
35 elif idd == 2: #checks to see if there is a face
36     face_cascade = cv2.CascadeClassifier('face.xml')
37
38     int face_image_count = 0;
39     for k in range(11):
40         img = cv2.imread('images/' + str(k) + '.jpg')
41         gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
42
43         faces = face_cascade.detectMultiScale(gray, 1.3, 5)
44
45         for (x,y,w,h) in faces:
46             cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
47             cv2.imwrite('images/' + str(k) + 'face.jpg',img)
48
49             cv2.imwrite('faceimages/' + str(face_image_count) + 'face.jpg',img)
50             face_image_count+=1
51
52             ser.write(str(k).encode())
53             print("found face")
54             face_found = True
55             break
56
57     if face_found == False:
58         ser.write(str(random.randint(1,13)).encode())
59
60 elif idd == 4: #restarts the program and deletes old images
61     shutil.rmtree('../final_project/images')
62     os.makedirs('../final_project/images')
63     shutil.rmtree('../final_project/faceimages')
64     os.makedirs('../final_project/faceimages')
65     i =0
```

Python 2 Code Part:
Checks each image saved in the images folder to see if there is a face. If there is a face send that number to the Arduino, if not send a random number. Wait for Arduino to send next serial number to delete the old images and restart the process.

Steps of implementation:

1. Capture an image at each angle during a single rotation
 - a. Determine how many photos are needed to capture all detail in a 360 degree rotation.
 - b. Using Arduino, accelerate chassis at correct speed for correct time to pause at each 30 degree angle in 360 rotation.
 - c. Using Arduino, send serial number to an 'Image capture' python program to take an image at each 30 degree angle.
 - d. Use facial recognition machine learning program to attempt to detect a face in each image.
 - e. Name image files 'num.jpeg' and send all image files to a folder called 'Images'.
2. Detect the face of the hider
 - a. Use the machine learning face classification programme to identify potential faces in each image.
 - b. Name each image containing a face 'numFace.jpeg' and send all face images to a folder called 'FaceImages'.
3. Move robot towards hider
 - a. Use LCD screen on chassis to determine angle that image is captured from any given start angle (ranges from 0-(+180), 0-(-180)).
 - b. Use this angle as signal to send serial number from python, (identifying the angle of the selected image for the arduino).
 - c. Function in arduino to move forward a fixed amount (one step) at this given angle when serial is read.
4. Develop programme to display images to public
 - a. Create title and instruction slide in 'star wars screen title theme'.
 - b. Draw live images into processing from 'Images' and 'FaceImages' folder and display them when specific keys are pressed.
 - c. Reset programme after one round of 'seeking' is complete.



Processing Code: The gave the viewer the opportunity to learn about the project, see the images the robot was taking, and show the faces it found.



Screenshot of how the processing code was presented on the screen.

Final Result

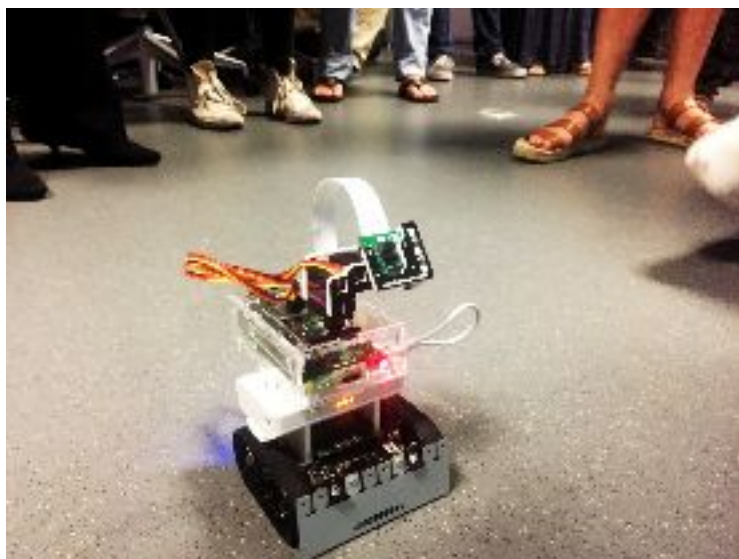
We anticipated that this experiment would give us important insights into the functionality of automates seeker methods used by a robot in real-world playing (interacting and predicting) against humans. We were able to successfully implement machine learning techniques by classifying faces from purely visual inputs. However, we were unable to implement model-based reinforcement learning to enable the robot to make a strategic movement. Instead, we extracted the angle of image, gave it a serial number, and sent this to the arduino to move to the specific angle, then move forward for a given distance.

In the future, we would like to find a better way of implementing the Processing visual with the automated robotic 'seeking'. Because the images took up large amount of working memory on the Pi, the 360 scan and the face identification processing time increased significantly after the first round of seeking. We were also planning to 'stitch' the panorama of images the robot took, and visually display this to the audience. However, the multiple image stitching programmes we found were difficult to implement with the Pi and did not work well with over 4 images in the dataset. In addition, we would like to produce our own training data which identifies a 'hider' and a 'base'. Using this training data, we can reward the 'seeker' when it completes a move that:

- Minimises the distance of the seeker from the hider (d_{sh})
- Maximises the hider from the base (d_{hb})
- The distance of the seeker from the base is less than the distance of the hider from the base ($d_{sb} < d_{hb}$).

Because of the time limitation on the project we were unable to generate enough training data and implement the reward function we created.

Overall, the success of this project can be assessed from the reactions at the MAT EoYS. We believe the audience felt engaged, inspired and even amused at the outcome of our project and the novelty of real-world play with a robot. Our project involved our audience visually and physically, and engaged them with questions such as "what situations can these techniques be adopted?", and "what situations can these techniques be abused?". Although the novelty of human-robot playing may evoke purely artistic expression, this experiment successfully explored the first steps toward real world search-and-rescue simulation.





Distribution of Labor

The distribution of labour is as follows:

1. Proposal Writing
 - a. Content - Christina
 - b. Layout - Christina
2. Capture an image at each angle during a single rotation
 - a. Arduino code - Ari and Christina
 - b. Python code - Ari and Christina
3. Detect the face of the hider
 - a. Python code - Ari and Christina
4. Move robot towards hider
 - a. Arduino code - Ari
 - b. Python code - Ari
5. Develop programme to display images to public
 - a. Processing code - Christina
6. End of Year Show
 - a. Delivery - Ari and Christina
7. Final Documentation
 - a. Content - Christina
 - b. Layout - Ari
 - c. Film - Ari and Christina
8. Presentation
 - a. Delivery - Ari

References

M.A. Armada et al. (2013) ROBOT2013: First Iberian Robotics Conference, 505 Advances in Intelligent Systems and Computing, 253.

Johansson, E., Balkenius, C. (2005) It's a child's game: Investigating cognitive development with playing robots. In: International Conference on Development and Learning, vol. 164.

Luo, R. (2017) Collecting boxes in the Unreal. (available at: http://www.rodgerluo.com/projects/rl_drones.html).

Professor Craven. (2016) Example code for Polulu's Arduino-based Zumo 32U4 robot, (available at https://github.com/pvcraven/zumo_32u4_examples).

RodgerLuo (2018) UCSB Intelligent Machine Vision Course, (available at <https://github.com/RodgerLuo/robotic-vision#ucsb-intelligent-machine-vision-course>).