Sorting robot

Group 6, Maria-Simona Chirita (3098508), Alexandru-Andrei Craciun (3103579), Katarzyna Łuszczewska (3097447), Peter Titev (2478404)

I. Introduction

There are numerous sorting robots in various factories and companies which perform different types of sorting such as rubbish, clothes or even packages sorting. That is why, the idea of the sorting robot as a final project appeared which is able to perform two types of sorting.

II. Purpose of the project

The aim of the project was to create a code, which will enable the robot to sort wooden cubes by colors and numbers. The robot, having chosen a certain element, will place it to the correct box. Moreover, the project enables to to check which method is faster - Machine Learning or only Python code with the use of OpenCV library. In the future in can be developed as an advanced robot detection complicated number arrays or, as presented in the introduction, sort various types of objects.

III. Design decisions and implementation

The sorting can be performed using Arduino Tinkerkit Braccio robotic arm and a camera, which can detect the cubes. The camera is placed on a wooden construction, 10 centimeters above robot's arm.

The project is using IP Webcam (on Android) to send the output of the phones camera to the laptop. The app is creating a small server on the phone, then using HTTP protocol, the python program can connect to the server and receive the phones camera output.

```
URL = "http://130.89.88.113:8080/shot.jpg"
def getImageInput():
    img_resp = requests.get(URL)
    img_arr = np.array(bytearray(img_resp.content), dtype=np.uint8)
    img = cv2.imdecode(img_arr, -1)
    img = cv2.resize(img, (SCREEN_X, SCREEN_Y), interpolation = cv2.INTER_AREA)
    return img
```

Fig. 1. Code for receiving the IP Webcam output

To make robot works as intended, a code for digit and color detection was needed, as well as the code to control the robot. What is more, the simple app was made, so the user can choose which type of sorting should the robot perform. However, the simulation had to be created so that it was possible to compare results with actual performance of the robot and verify, if making such robot is doable.

A. Simulation in UPPAAL

As mentioned above, the project started with making the simulation in UPPAAL. At the beginning, global variables were declared such as initial motor position which were taken from Inverse Kinematics library for Arduino or the current one. Next, two specs were created. The first one for the position of the brick, second one for updating the time.

Fig. 2. Spec for position

Fig. 3. Spec for time update

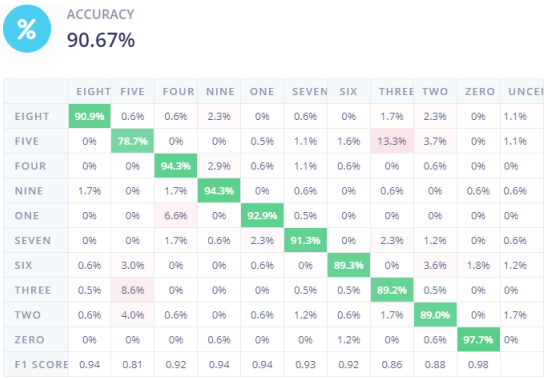
The first spec only force each part of the arm claw to move when it is needed. The second checks the position of each motor and if the certain motor has moved. If so, both time and position are updated for a certain motor. The whole process ends when the arm goes back to its initial position after performing sorting and total time is updated. The sufficient query for that was written. It checks, if it is possible for the robot to sort pieces within 588 seconds with the speed of 1 second (this one was declared in global variables part). However, the robot moves with speed of 30 degrees per second. Therefore, the time is 19.6 seconds plus 10 seconds for the delays which results in around 30 seconds for the whole performance. This query was satisfied so it was highly probable, that the robot could also perform the task properly.

B. Digit detection

For the digit detection, 20% of the MNIST (Modified National Institute of Standards and Technology database) handwritten digit database was used. The Edge Impulse software was used to create the neural network. For extracting features, an image processing block was used to normalize image data. For training the model, a transfer learning block was used, because of its accuracy for image classification. A dataset of

around 10.000 samples was used, alongside a training dataset of around 1.700 samples that was used in order to check the performance of the neural network.

Even though the model testing had an accuracy of over 90% and F1 scores between 0.81 – 0.98, the digit detection did not go as planned because of the fact that the model was trained to only detect a digit in a plain background and the phone camera was capturing a lot more. That is why 2 other approaches were tried.



	EIGHT	FIVE	FOUR	NINE	ONE	SEVEN	SIX	THREE	TWO	ZERO	UNCEI
EIGHT	90.9%	0.6%	0.6%	2.3%	0%	0.6%	0%	1.7%	2.3%	0%	1.1%
FIVE	0%	78.7%	0%	0%	0.5%	1.1%	1.6%	13.3%	3.7%	0%	1.1%
FOUR	0%	0%	94.3%	2.9%	0.6%	1.1%	0.6%	0%	0.6%	0%	0%
NINE	1.7%	0%	1.7%	94.3%	0%	0.6%	0%	0.6%	0%	0.6%	0.6%
ONE	0%	0%	6.6%	0%	92.9%	0.5%	0%	0%	0%	0%	0%
SEVEN	0%	0%	1.7%	0.6%	2.3%	91.3%	0%	2.3%	1.2%	0%	0.6%
SIX	0.6%	3.0%	0%	0%	0.6%	0%	89.3%	0%	3.6%	1.8%	1.2%
THREE	0.5%	8.6%	0%	0%	0%	0.5%	0.5%	89.2%	0.5%	0%	0%
TWO	0.6%	4.0%	0.6%	0%	0.6%	1.2%	0.6%	1.7%	89.0%	0%	1.7%
ZERO	0%	0%	0%	0.6%	0%	0%	1.2%	0%	0.6%	97.7%	0%
F1 SCORE	0.94	0.81	0.92	0.94	0.94	0.93	0.92	0.86	0.88	0.98	

Fig. 4. Accuracy of the model

The first one was to use Object Detection for training instead of Transfer Learning for images and use FOMO (Faster Objects, More Objects) in the neural network architecture. That would allow detection of digits in any place as long as it was in the camera's sight. However, sampling would have taken a lot of time as using the MNIST database would not have been possible. The second approach was to firstly detect the piece of paper and then use the initial algorithm to detect the digit.

C. Color detection

For the color detection, it has been used OpenCV. First, using the image received from the IP Webcam, the python script creates a mask for every type of color(blue, purple, yellow and green).

```python
def getProperMask(img):
    listOfMasks = []
    hsv = cv2.cvtColor(img , cv2.COLOR_BGR2HSV)
    #blue
    lower_bound_1 = np.array([105, 96, 121])
    upper_bound_1 = np.array([153, 255, 255])
    listOfMasks.append(cv2.inRange(hsv, lower_bound_1, upper_bound_1))

    #yellow
    lower_bound_3 = np.array([23, 95, 92])
    upper_bound_3 = np.array([44, 154, 247])
    listOfMasks.append(cv2.inRange(hsv, lower_bound_3, upper_bound_3))

    #green
    lower_bound_4 = np.array([44, 101, 68])
    upper_bound_4 = np.array([79, 255, 255])
    listOfMasks.append(cv2.inRange(hsv, lower_bound_4, upper_bound_4))

    #purple
    lower_bound_5 = np.array([125, 111, 94])
    upper_bound_5 = np.array([147, 193, 156])
    listOfMasks.append(cv2.inRange(hsv, lower_bound_5, upper_bound_5))

    return listOfMasks
```

Fig. 5. Creating the list of masks

Now, inside every mask, every pixel that matches the color has been represented with the white color and the background has been represented with the black color. After this, the program iterates through all the masks and tries to detect biggest contour.

```python
def findBricks(img):
    allPos = dict.fromkeys(POSSIBLE_COLOR)
    listOfMasks = getProperMask(img)
    index = 0
    rect = None
    for mask in listOfMasks:
        #cv2.imshow(POSSIBLE_COLOR[index], mask)
        contours, h = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        biggest_contours = (-1, -1, -1, -1)
        maxArea = 1000
        for c in contours:

            x, y, w, h = cv2.boundingRect(c)
            if x > 200 or y < 200:
                continue
            if cv2.contourArea(c) >= maxArea:
                biggest_contours = cv2.boundingRect(c)
                maxArea = cv2.contourArea(c)
                rect = cv2.minAreaRect(c)
        x, y, w, h = biggest_contours
        if x != -1 and y != -1:
            (_, _), (_, _), angle = rect
            allPos[POSSIBLE_COLOR[index]] = (round(x + w/2), round(y + h/2), angle)
        index += 1
    return img, allPos
```

Fig. 6. Detecting the contour of the bricks

D. Arm control

Using Inverse Kinematics library*, the program can control the robot arm based on cartesian coordinates instead of servo motor angles. The project has implemented a serial communication between the Arduino that is controlling the robot and the python script that is finding and calculating the positions of the bricks. First the python code calculates the position of the detected bricks ,then it send the initial position and the final position of the brick trough the serial communication to the Arduino.

* https://github.com/cgxeiji/CGx-InverseK

```python
def sendPosToArduino(x_start, y_start, angle, x_finish, y_finish, arduino):
    output = str(x_start) + " " + str(y_start) + " " + str(round(90 - angle)) +  " " +  str(x_finish) + " " + str(y_finish) + "\n"
    print("Writting: " + output)
    arduino.write(output.encode('utf-8'))
    while True:
        response = arduino.readline().decode('utf-8')
        if 'DONE\n' in response:
            print("The code is done")
            break
        elif 'ERROR_DOWN\n' in response:
            print("There is no solution to this coord on drop")
            break
        elif 'ERROR_UP\n' in response:
            print("There is no solution to this coord on pick up")
            break
        #print("response: " + response)
        time.sleep(0.5)
```

Fig. 7. Sending the position from Python to Arduino

```
void loop() {
  int posx = -1, posy = -1;
  char input[INPUT_SIZE + 1];
  bool read = false;
  while(Serial.available() > 0){
    byte size = Serial.readBytes(input, INPUT_SIZE);
    input[INPUT_SIZE] = 0;
    read = true;
  }
  if(read){
    //int posxstart = map(atoi(strtok(input, " \n")), 0, 600, -300, 300);
    //int posystart = map(atoi(strtok(NULL, " \n")), 0, 600, -300, 300);
    int posxstart = atoi(strtok(input, " \n"));
    int posystart = atoi(strtok(NULL, " \n"));
    int angle = atoi(strtok(NULL, " \n"));
    //int posxstart = map(atoi(strtok(input, " \n")), -500, 500, -300, 300);
    //int posystart = map(atoi(strtok(NULL, " \n")), -500, 500, -300, 300);
    int posxfinish = atoi(strtok(NULL, " \n"));
    int posyfinish = atoi(strtok(NULL, " \n"));

    solveAndMoveRobot(posxstart, posystart, angle, posxfinish, posyfinish);
    read = false;
  }
}
```

Fig. 8. Receiving the data on the Arduino

After that, using the inverse kinematics, the Arduino calculates each step motors angle and finally we move the robot based on those angles. After the robot finishes the moves, it sends back a feedback based on the status of the solution.

```
void solveAndMoveRobot(int xStart, int yStart, int angle, int xFinish, int yFinish){
  float a0UP, a1UP, a2UP, a3UP;
  float a0DOWN, a1DOWN, a2DOWN, a3DOWN;

  bool goUP = false, goDOWN = false;

  goUP = InverseK.solve(xStart, yStart, 20, a0UP, a1UP, a2UP, a3UP);
  goDOWN = InverseK.solve(xFinish, yFinish, 50, a0DOWN, a1DOWN, a2DOWN, a3DOWN);

  if(goUP == false){
    Serial.print("ERROR_UP\n");
    return;
  }
  if(goDOWN == false){
    Serial.print("ERROR_DOWN\n");
    return;
  }
  moveRobotToPickUp(a0UP, a1UP, a2UP, a3UP, 20);
  moveRobotToDrop(a0DOWN, a1DOWN, a2DOWN, a3DOWN, angle);
  Serial.println("DONE\n");
}
```

Fig. 9. Solving the Inverse Kinematics Problem

E. User Interface

For the user interface, an Android application was created in Android Studio in order to choose the desired type of sorting that has to be performed. A server was created to make the communication between the app and the python script possible. The server is running on the laptop and is waiting for a post request from the app. That post request contains the mode of sorting that was chosen by the user. All the comunication is done over WI-FI.

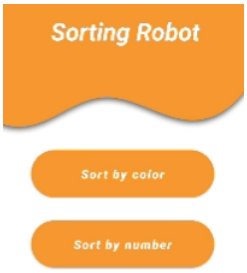


Fig. 10. Android Application

IV. Sorting comparison

The two types of sorting were performed and verified with a model made in UPPAAL. It can be seen, that an actual sorting time for color detection is longer than in a simulation. The whole operation of colors sorting consists of placing 16 colorful bricks into right boxes. Time which is needed to perform it in an UPPAAL model is 30 seconds, whereas during actual sorting, robot need 320 seconds to place all elements in the right box. For the second type of sorting 4 bricks were used, each pair has a number 0 or 4 on them. In this case also the cubes were placed in correct boxes. The whole performance took the robot 112 seconds.

TABLE I
Time robot requires to sort one brick

Type of sorting	Time[s]
by numbers	20
by digits	28

As it is presented in the table above, color sorting is undeniably faster. This is caused by using Machine Learning for digit sorting, as camera needed more time to detect the number. Moreover, results for color sorting vary a lot which is mainly because of the robot delays such as arm movement and time it needs to detect the color.

V. Conclusion

To conclude, we believe that the basic functionality, that we had in mind as we took on this assignment, has been achieved although, as always, there is room for future improvements as some of the features did not work as well as we wanted them to. One of these features is the number detection, that worked as a neural network but lead to problems when implemented alongside the robot arm. Another future improvement would be the pick up action of the arm. As of now the robot can only pick up pieces placed in a particular way on a particular spot. For future development we would like to have the robot detect the orientation and location of the bricks and rotate its claw in such a way that it can be able to grip the pieces from the sides. Unfortunately implementing this was deemed too time consuming and complex for the scope of this project but could be a useful feature that can be added in the future. On the other hand, color detection worked exactly as we initially planned, although not as fast as the UPPAAL model, and we even added some additional features like the phone application to control the robot arm and the custom bricks and boxes we used for the color sorting feature. To wrap things up, it is safe to say that the team is satisfied with the final result of the robot and the things we accomplished throughout the project.