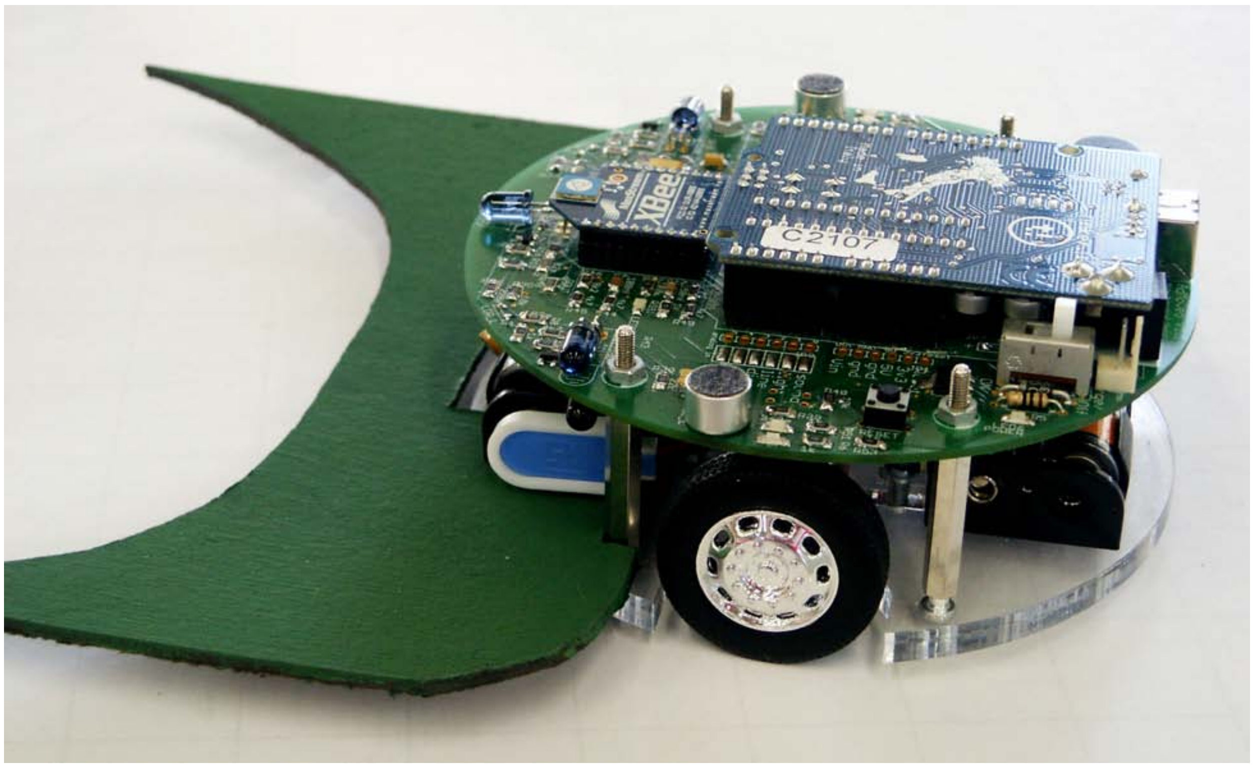
DBD04 Learning Robots



Jelle Dekker s070085
Laura van Geel s070089
Jasper de Kruiff s070094
Vincent Klerkx s071143

Department of Industrial Design
TU/e, March 2011

Table of Contents

Chapter 1

1.1 Introduction

The aim of this module was to develop a system that resulted in autonomous and intelligent behaviour. To develop such a system learning algorithms were introduced. Three different types of learning algorithms were explained; Unsupervised learning, reinforcement learning and supervised learning.
In the case of unsupervised learning a system is able to make different clusters by it's self, there is no correcting factor that influences the system.
Reinforcement learning is when you guide that system, the system performs an action and a reward is given for correct behaviour, and no reward for incorrect behaviour.
Supervised learning is when you give direct feedback to all the actions that the system performs.

In this report we will show how we implemented unsupervised learning algorithm through theory and practice. We do this by means of developed concept, in our case the 'Garbage remove' concept. Input from the systems environment will be used to make a closeting between different types of objects that can be found. As a result of the clustering a trained algorithm is able to identify different objects in the given environment. Further in this report we explain the concept, the platform which is used and the chosen learning algorithm and how it was implemented.

1.2 Initial concept

In the beginning of this module we were handed the AdMoVeo robot (see 1.3 The Platform) as a platform to interface. The initial concept was mainly based on the specifications and the expected performance and opportunities of the AdMoVeo. The goal was to apply reinforcement learning on individual agents to enable object detection using its distance sensor to sense the length of objects and use this to identify different types of trash by unsupervised learning to the agents. On top of that, reinforcement learning is used to learn the agents to help each other.

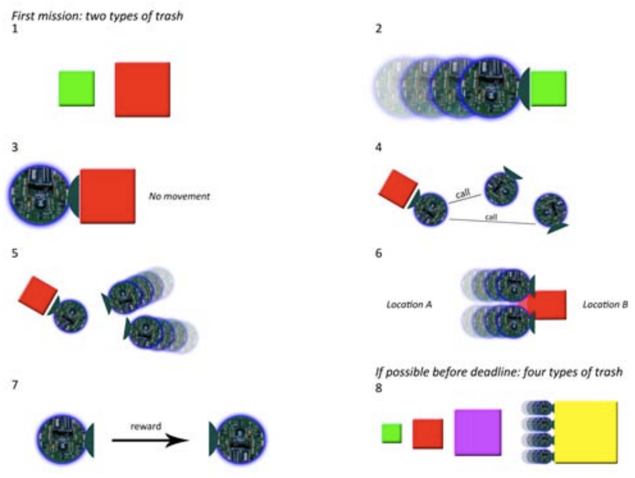

First mission: two types of trash
1
2
3
No movement
4
call
call
5
6
Location A
Location B
If possible before deadline: four types of trash
7
reward
8

Figure 1 Concept overview

The concept:

1 Two different types of trash are represented by foam blocks with two different colors: red (big) blocks and green (small) blocks

2 One robot can push away green blocks

3 Red blocks are *too heavy* for one robot to push

4 When a robot encounters a red block, it calls the other robots

5 When other robots receive the call, they move towards it

6 Multiple robots push away red blocks

7 The robot which called for help sends a *thanks*-command to the helping robot, which stimulates learning to help other agents

8 Multiple types of trash are represented by foam blocks with multiple colours (*multiple weights*)

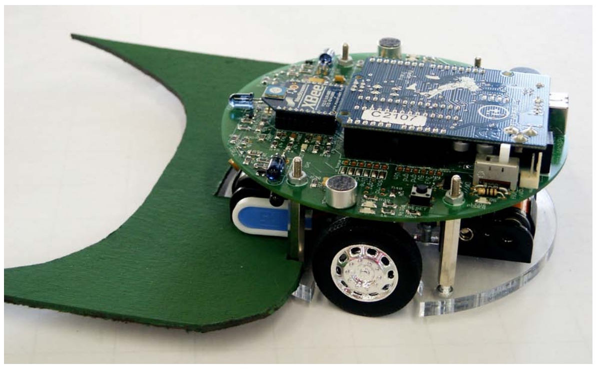
1.3 Final concept 'Clean-O-Bots'

The concept was to make the robots remove garbage from a given space. A space was defined in which the robots were expected to remove garbage and thereby leave other robots where they are.

The expected learning behaviour was that the system would be able to identify garbage from itself; we used three different types of garbage paper clots, soda cans and cups. By means of unsupervised learning the system is able to cluster the different types of objects, after it has been trained. In the training stage the system makes different clusters for each type of object that the system can distinguish. After this learning process the system is able to recognise the garbage and make a distinction between garbage and robots. The robot will now move towards the garbage, collect it and move it towards its corresponding garbage bin.

1.3 The Platform

The platform used in this module was a combination of a robot and an overhanging webcam that work together as an integrated system.



FigFigure 2 AdMoVeo

1.3.1 AdMoVeo robot

The AdMoVeo robot is an educational robot developed within the faculty of Industrial Design at the Eindhoven University of Technology to teach first year students the basics of programming. In total it contains 8 sensors and 4 actuators:
- three infrared distance sensors
- two light sensors
- two sound sensors
- a line follower
- two geared DC motors (for two independent wheels)
- a buzzer
- an RGB Led

The robot is controlled by an Arduino board that communicates to a regular computer using either USB or a wireless Xbee communication module. It is powered by four high capacity rechargeable AA batteries.

We made a custom addition to the chassis of the robot: a front facing cardboard scoop that allows the robot to collect and transport small pieces of trash.

1.3.2 Webcam

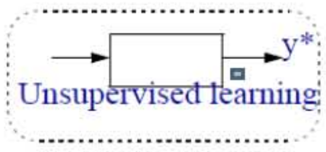
As the sensors on the robot itself were not sufficient for our needs, an overhanging webcam was used to act as the 'eyes' of the entire system. We choose a Microsoft LifeCam VX-800 for its picture quality and low costs. It operates at 15 frames per second at a maximum resolution of 640 x 480 pixels (0,3 megapixel).

The webcam was used to gain an overview of the environment, to detect and identify different types of trash. It also enabled us to track and guide the AdMoVeo robot to its targets.

Chapter 2

2.1 Learning Algorithm

Unsupervised learning



An unsupervised learning algorithm is able to make a clustering between the provided inputs without receiving rewards. Such a system can be followed by coded action that have to be performed when the input is related to one of the clusters or a supervised or reinforced learning algorithm can be used.

In this learning algorithm the system will make an internal image of clusters of the different types of garbage. For our system we used the neural gas algorithm.

The following formula is that basis of this algorithm:

$$w_{i_k}^{t+1} = w_{i_k}^{t} + \epsilon \cdot e^{-k/\lambda} \cdot (x - w_{i_k}^{t})$$

Given a probability distribution P(x) the data of vectors x. Per each time step t a data vector chosen from P is presented. Then the distance to the data vector x is determent, i0 gives the index of how close the vector. [1]

In the formula ε is the adaptation step size and λ is the so-called neighbourhood range. ε and λ are reduced with increasing t. after many adaptations the vectors cover the data space with a low error rate. [1]
"The adaptation step of the neural gas can be interpreted as gradient descent on a cost function. By adapting not only the closest feature vector but all of them with a step size decreasing with increasing distance order, compared to k-means clustering a much more robust convergence of the algorithm can be achieved. The neural gas model does not delete a node and also does not create new nodes."[1]

The advantage of the learning algorithm is that it is possible to place more garbage of the same type in the environment and the robot would automatically go to right location. Also it would be easier to learn the system to recognize other types of garbage by using the learning
algorithm, and remove this garbage from the scene.

[1] *http://en.wikipedia.org/wiki/Neural_gas, retrieved 2011, March11*

Figure 3 Webcam

2.2 Implementation

2.2.1 Set-up

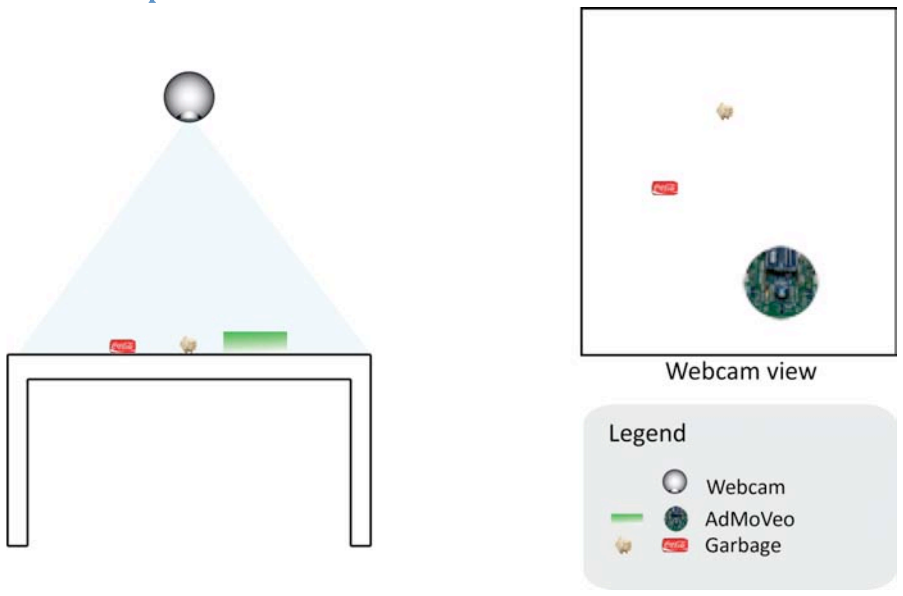


Figure 4 Set-up Overview

In the image above an illustration is given of how the set-up looked like for our system. On the table a white paper was placed to determent the area in which the robot could navigate and to have to highest contrast between the background and garbage in the images. Whenever a robot moves out of the range of the camera the robot is stopped preventing it from driving of the table.

2.2.2 Overview of the implementation

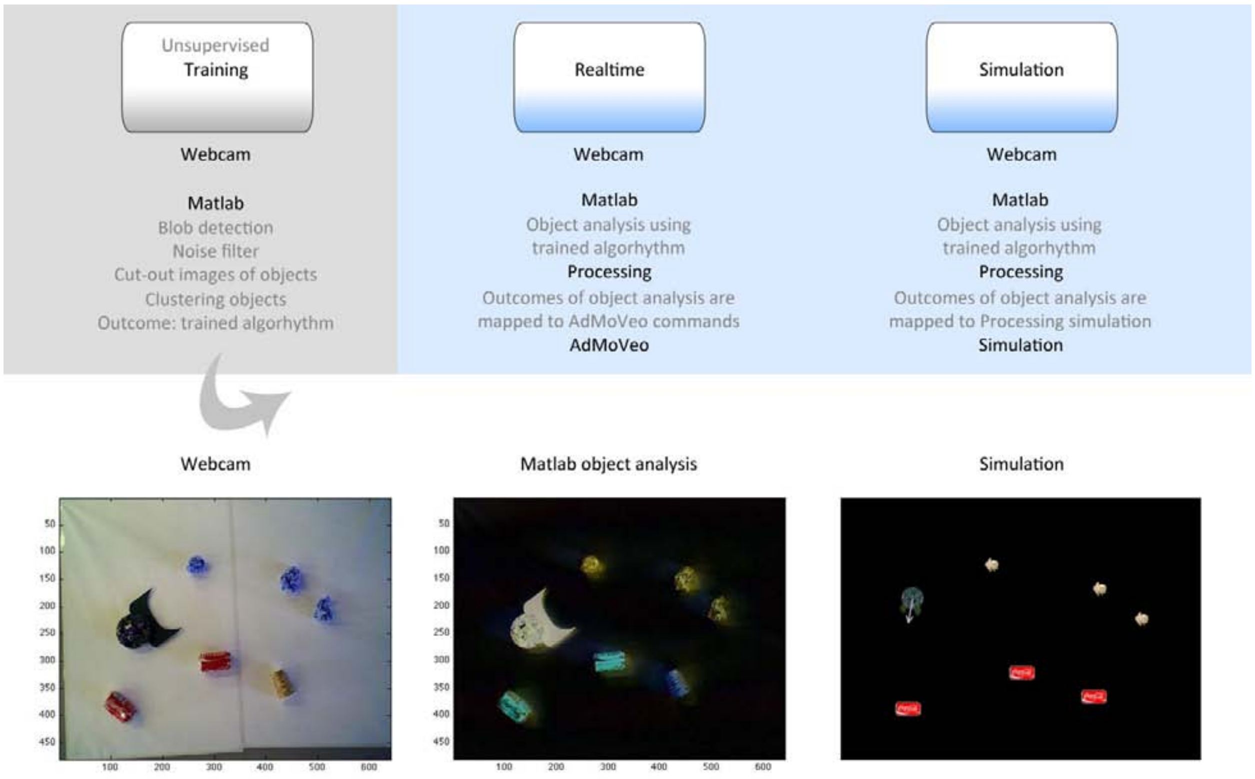


Figure 5 Overview implementation

Webcam
The input that the system gets is captured by means of the webcam. Be fore starting the system has to recalibrate the background a background image needs to be captured. Then the garbage can be placed in the environment and a real-time image can be captured.

Matlab
Matlab is used to capture the images of the webcam, and to process the images into data that can be read by the neural gas algorithm. To do so first the difference between the background image and the new scene is calculated, leaving only the garbage and robot on the screen. Blob detection is used to find the different artefacts in the scene, these artefacts are cropped and resized into separate images for further processing. Matlab takes the median HSV for each pixel of these cropped samples. These variables could be identified by means of the learning algorithm. The blob detection also was able to subtract the coordinates of the object in the real image.

Processing
Matlab would communicate the coordinates of the garbage to processing to control the AdMoVeo and to place the garbage in out simulation. We could move the robot towards the garbage and make it clean the garbage in the right location.

Chapter 3

3.1 Expectation

We expect that the system is able to make a difference between the robots and the garbage. After identifying the garbage the system should be able to move the robot towards the garbage and place the garbage in the garbage bin (in the corner of the area).

When the system is able to identify paper clots the robot will move towards a paper clot and place the paper clot at a predefined location in the environment.

3.2 Execution

Before we could make the system a smart learning system a training set was developed which could be used for training the algorithm. In the training set there were four objects that could be distinguished, the AdMoVeo, soda cans, paper cups and paper clots. Figure 6 shows a larger version of the images that were used as training set for the algorithm.

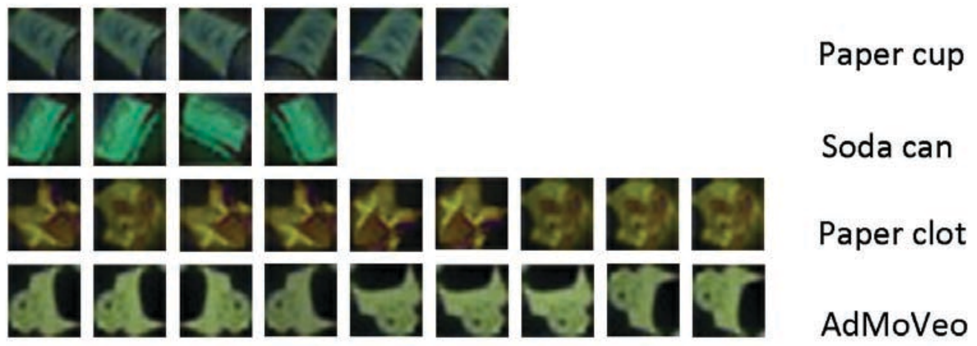


Figure 6 Training set

When the system is activated the system will first ask you to take an image of the background (see figure 7 & 8). Then the garbage and robot can be placed in the scene, a new image has to be taken. From these two images we take the absolute difference for further processing. This new image is used for blob detection, all objects seen in the image are separated and all objects below a certain size are ignored to remove noise from the data. These images are then cropped to contain only the object and resized to 20*20pixels.

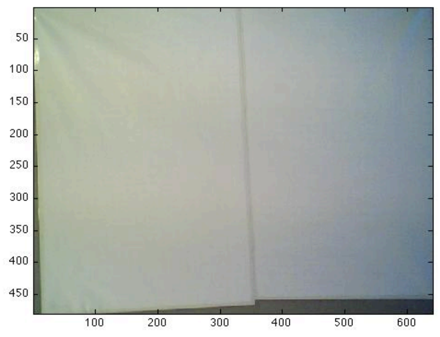


Figure 7

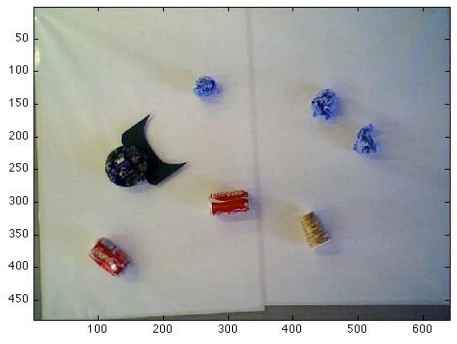


Figure 8

3.2.1 How to analyse the data for the neural gas algorithm?

Several methods have been tried in combination with the Neural Gas Algorithm to make a distinction between different types of objects (cups, cans, clots and robots). First simply all pixels were read in Matlab, this made all images so different that the Neural Gas algorithm couldn't make a useful distinction. A second try was to use all pixels but only black/white data, so each pixel was set to black or white, depending on the threshold set for this conversion. This resulted in too similar images for the different objects. A third method used was the contour of the object, for this a function was used in Matlab that made all pixels black with the exception of the contour of the blob it had detected. This made the images too similar even for human eyes because the noise from shadows made all contours very much alike.

One method that was presented during the lectures was to count b/w pixels in horizontal and vertical directions, which is a way of looking at the image in an entirely different view than when looking at all pixels. This works well to determine different orientations of the same object, if you already know the object. Unfortunately for us we don't know the orientation of the image before analyzing it so this method doesn't work to identify our different kinds of objects in the scene that can be oriented in many ways.

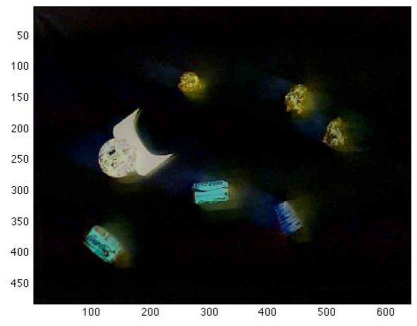


Figure 9

One thing that stood out when viewing the resulting image from the background and scene is the colour of the objects. There is a difference between the range of colours within the same class of object and between the different classes. Using the image on itself in the Neural gas algorithm didn't work, there was too much variation in how the individual pixels made the objects. However instead of looking at the separate Red, Green and Blue (RGB) values for each pixels, we can also convert the image to HSV, which will give the Hue (tint), Saturation and Value (brightness) of each pixel. A slight difference of light can change the red, green and blue-value of a pixel but may only change the brightness (V) of the HSV-colourmap of the same pixel. This can be seen when we look at the variables made in Matlab of the images, the saturation and especially the hue are more consisted in the cropped images than the lightness. The "noise" that the light conditions and camera recording cause in the RFB-colourmap is reduced because it has been isolated from the other colour information in the image.

Now using 20x20 pixels and recording 3 different values on each pixel would give us an array of 1200 numbers for each image to work on in the Neural Gas algorithm, we want to categorize 4 different, quite simple, objects. Especially with the picture quality and resizing after taking the smallest possible crop that contains the whole blob, these objects consist mainly of a single hue/colour/plane, with some noise from the background that is still part of the crop. The goal is to remove the noise as much as possible, and pass on less data to the Neural Gas algorithm. To do this we take the median of the HSV-map of the 20*20px image, this returns the median value for each column, resulting in 20 numbers describing the hue of the image, 20 numbers for the saturation and 20 numbers for the brightness. Possible the 'mean' may also have been used, but 'median' appears to be more consistent probably because with the 'mean' the background present in the image has a bigger influence on the resulting number for the column.

3.2.2 How to train, test and use the network?

To train the network an image was taken using Matlab as described earlier, however the resulting image with the absolute difference of the scene and the background is not ready to use as a training set for the network. But this image was used as a base for the training set, editing the crops of the objects in many ways, playing with the rotation, lightness and contrast of the images to make a more complete training set. In the end this resulted in 28 images for the 4 different categories, on average 7 per category. To test the network the data from the training set was used as a test set to see if the objects, because of the limited amount of samples for the training set, the training of the network had to be tweaked to take more steps (500 in the end) before it managed to separate all categories in the right way.

After the training and test the trained network could be used in the 'real' situation. For this the exact same steps are taken as before regarding the visual input. By loading the trained network it's known which categories in the Neural Gas correspond with which objects. Based in this 4 text files are made where the coordinates of the centroids of the blobs are stored, each category (cans, cups, clots and robots) is stored in it's own text file, ready to be read by Processing for the next step.

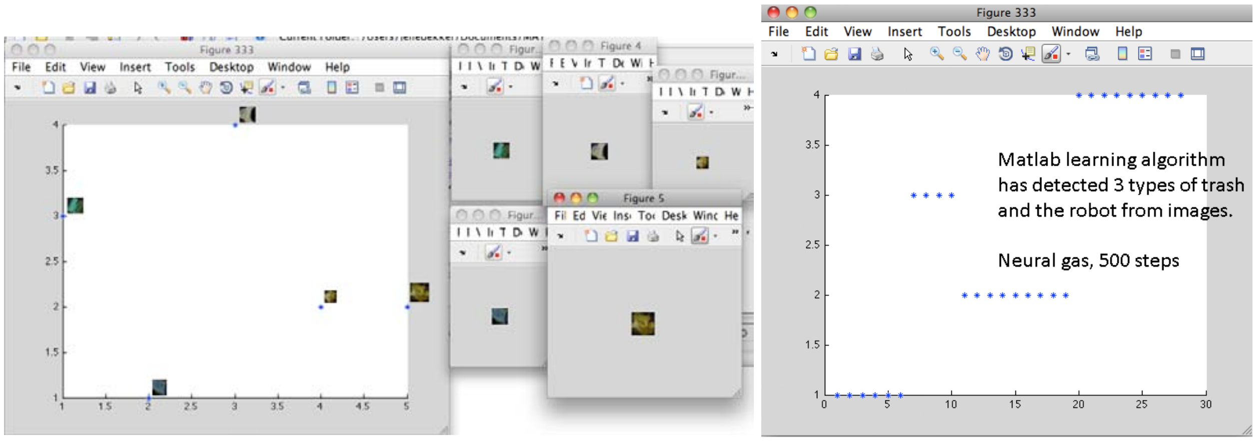


Figure 333
File Edit View Insert Tools Desktop Window Help
Figure 4
Figure 5
Matlab learning algorithm has detected 3 types of trash and the robot from images.
Neural gas, 500 steps

Figure 10

3.2.3 Using the data from the network to control robot/simulation

The files with coordinates that came from Matlab are loaded into processing, to keep the implementation more manageable here only two objects are used instead of four. Both a simulation and a control system for the robot were developed, the simulation was made to be able to clarify the concept without the difficulties added by controlling the robot using the webcam as input.

Controlling the AdMoVeo
The second version in processing uses the webcam, OpenCV and the actual AdMoVeo to move the trash to predetermined locations. The system in Processing knows where the trash is based on the coordinates from the analysis done in Matlab (imported using the text-files). It tracks the robots position using the webcam, it can do this without learning by simply ignoring all smaller objects. The biggest difficulty encountered is the movement of the AdMoVeo, it doesn't know its direction or speed. This is solved by making the AdMoVeo move, then calculating the difference in position, trying to get closer in both the x and y-direction. In the execution this wasn't done using a learning algorithm, but Q-learning would probably have been a suiting implementation.

Simulation
In the simulation the trash is placed on a field according to the coordinates from the Matlab text-files. After that agents (robots) are loaded that will move the trash to the right locations. The learning algorithm is used only to identify the trash in the scene here.

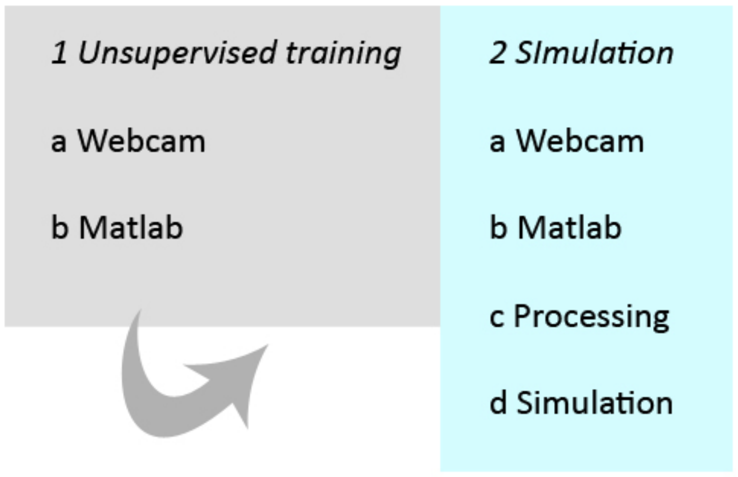
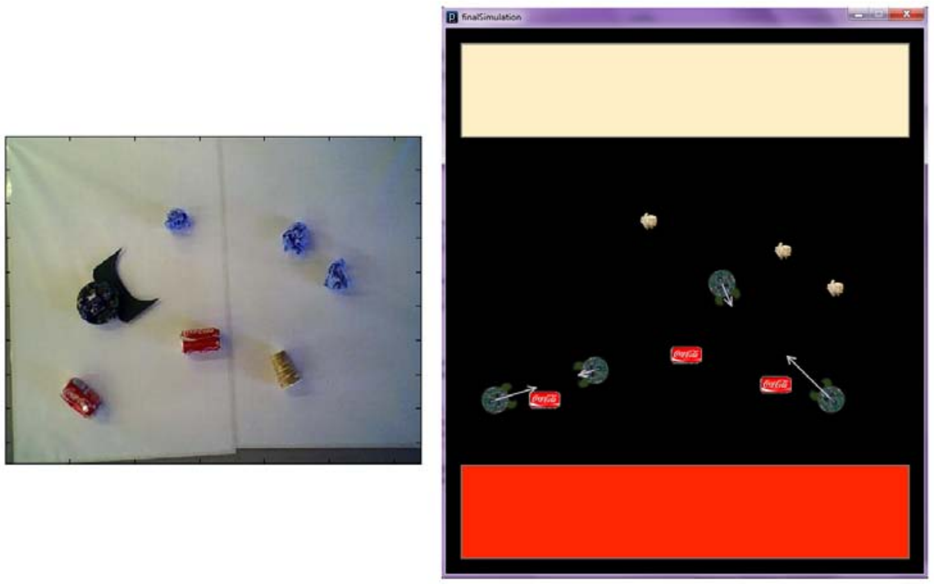


Figure 11



Chapter 4

4.1 Improvements of the developed system

The way the Clean-o-bot system was developed could be improved in the future. Now it was hard to direct the AdMoVeo in the direction of the waist as well as it was hard to make it move the garbage to the right location. If we had implemented the Q-learning system it would have been possible for the system to learn what the shortest route was all by itself. Currently the system did this by trying what was possible and if it was wrong try another option Q-learning would have done the same but the system would be able to learn when it was wrong and not try that again.

4.2 Learning algorithms and a design project

The learning algorithms could be of great advantage to the development of intelligent products for society. It opens up possibilities for system that can adapt to changes without having to reprogram the system again. For a design project it means that you can make a system that is able to adapt by itself which will give interesting outcomes, like for instance you could create emergent behaviour.

In case of Q-learning you could also develop a system that is able to adapt to changes over time. For instance if you have a system that can recognise different people who are in a room and adapt the atmosphere to that person such a system would be able to identify and learn if there is a new person in that room. A learning system gives room for all sorts of personalised systems and advice giving services.

It can be said that learning systems have a close relation to Ambient Intelligence and could be developed into interesting products that are of use for society. For instance our concept could be scaled to a bigger scale and used in the harbour for locating containers and trucks.

Appendix A Code

A.1 Matlab code

Train Network

```
%Made for the DBD04 Learning Robots Module
%with help from the "Neural Gas"-algorithm, provided by lecturer Emilia I. Barakova
%Jelle Dekker, Laura van Geel, Jasper de Kruiff, Vincent Klerkx
%Feb-Mar 2011, Industrial Design, TU/e

%Function to train the network, it takes images as input, calculates the
%median Hue-Saturation and feeds that data to the neural network for
%unsupervised categorization.

%%Open file to store img data
fid=fopen('train_data.txt','w');

%%Folder where the images are stored
folder = 'ImgFolder/3';

%%declare amount of images
n = 28;
for i = 1 : n
    % Get the base file name.
    baseFileName = sprintf('%d.jpg', i);
    % Combine it with the folder to get the full filename.
    fullFileName = fullfile(folder, baseFileName);

    %%Get the median Hue-Saturation (hsv) from the image
    img_temp = imread(fullFileName);
    hsv=rgb2hsv(img_temp);
    img=median(hsv);

    %%Convert data to a string and remove unnecessary spaces
    a = num2str(img);
    for k=1:px
         a = strrep(a, '  ', ' ');
    end

    %%start a new line and write img data
    fprintf(fid,'\n');
    fprintf(fid,a);
end

fclose=(fid);
clear;

%%load the data on imgs, use as input for neural gas
load train_data.txt;
TrainingSet_Clusters = neural_gas_cluster (train_data, 4, 500);
```

Analyze Scene

```
%Made for the DBD04 Learning Robots Module
%with help from the "BlobsDemo":
%(http://www.mathworks.com/matlabcentral/fileexchange/25157-blobsdemo)
%Jelle Dekker, Laura van Geel, Jasper de Kruiff, Vincent Klerkx
%Feb-Mar 2011, Industrial Design, TU/e


%Live/online version of the img analysis. This takes two screenshots from
%a video input (background and scene), compares these, finds the objects
%(blobs), extracts these from the images and then uses these crops for the
%classification input. The catagories are then written to 4 files, each
%containing the coordinates of a category of objects found.

disp('Starting intelligent scene analyzer...');

vid = videoinput('macvideo',1,'YUY2_640x480');
set(vid,'ReturnedColorSpace','rgb');
set(vid,'TriggerRepeat',Inf);
%start(vid);
preview(vid);

%Ask for first screenshot
reply = questdlg('Make background img?','System ready...', 'Yes', 'No', 'Yes');
loop=true;
while loop==true
    if strcmpi(reply, 'Yes')
        bg = getsnapshot(vid);  %take screenshot
        disp('Background image captured...');
        loop=false; %exit loop
    end
    if strcmpi(reply, 'No')
        reply = questdlg('Make background img?','System ready...', 'Yes', 'No', 'Yes');
    end
end

%Ask for second screenshot
reply2 = questdlg('Is the scene ready?','BG made...', 'Yes', 'No', 'Yes');
loop=true;
while loop==true
    if strcmpi(reply2, 'Yes')
        b = getsnapshot(vid); %take screenshot
        disp('Scene captured, starting analysis...');
        stop(vid);
        loop=false; %exit loop
    end
    if strcmpi(reply2, 'No')
        reply2 = questdlg('Is the scene ready?','BG made...', 'Yes', 'No', 'Yes');
    end
end

%Calculate difference between images and make a binary version
diff_im = imabsdiff(b,bg);
diff_bw = im2bw(diff_im,0.2);
binaryImage = imfill(diff_bw, 'holes');

%make a grayscale version, and find connectivity in the binary version
originalImage = rgb2gray(diff_im);
labeledImage = bwlabel(binaryImage, 8);

%use the two version to find blobs in the image, and count them
blobMeasurements = regionprops(labeledImage, originalImage, 'all');
```

```
numberOfBlobs = size(blobMeasurements, 1);

%open txt files to write the coordinates into
robots=fopen('robots.txt','w');
clots=fopen('clots.txt','w');
cans=fopen('cans.txt','w');
cups=fopen('cups.txt','w');

blobECD = zeros(1, numberOfBlobs);
% Print header line in the command window.
fprintf(1,'Blob #      Mean Intensity  Area   Perimeter    Centroid       Diameter\n');
% Loop over all blobs printing their measurements to the command window
for k = 1 : numberOfBlobs           % Loop through all blobs.
    % Find the mean of each blob.  (R2008a has a better way where you can pass the
original image
    % directly into regionprops.  The way below works for all versions including earlier
versions.)
    thisBlobsPixels = blobMeasurements(k).PixelIdxList;  % Get list of pixels in current
blob.
    meanGL = mean(originalImage(thisBlobsPixels)); % Find mean intensity (in original
image!)
    meanGL2008a = blobMeasurements(k).MeanIntensity; % Mean again, but only for version >=
R2008a

    blobArea = blobMeasurements(k).Area;        % Get area.
    blobPerimeter = blobMeasurements(k).Perimeter;      % Get perimeter.
    blobCentroid = blobMeasurements(k).Centroid;        % Get centroid.
    blobECD(k) = sqrt(4 * blobArea / pi);                   % Compute ECD - Equivalent
Circular Diameter.

    %filter out noise, only blobs over a certain size are considered
    if (blobPerimeter > 80)
        fprintf(1,'#%2d %17.1f %11.1f %8.1f %8.1f %8.1f % 8.1f\n', k, meanGL, blobArea,
blobPerimeter, blobCentroid, blobECD(k));

        %Get the current blobs box and crop it
        thisBlobsBoundingBox = blobMeasurements(k).BoundingBox;
        subImage = imcrop(diff_im, thisBlobsBoundingBox);
        subImage = imresize(subImage, [20 20]);
        figure, imshow(subImage);

        %Get the median Hue-Saturation (hsv) from the image
        hsv=rgb2hsv(subImage);
        img=median(hsv);

        %Turn data into string and remove unnecessary spaces
        a = num2str(img);
        for k=1:20
            a = strrep(a, '  ', ' ');
        end

        %Put data in a 'txt' file
        fid=fopen('test_data.txt','w');
        fprintf(fid,a);
        fclose=(fid);

        %Read this txt file so raw text is made into 1x60 double
        load test_data.txt;
        for Scan = 1:1:1
            test_data_class = neural_gas_class(TrainingSet_Clusters,test_data(Scan,:));
%Classifies the test vectors to the known classes

            %determine the class (based on categorization after training)
            %and put the coordinates of the objects in the right files.
            if test_data_class==1
                fprintf(cups,'%2.1f\t%2.1f',blobCentroid);
                fprintf(cups,'\n');
            end

            if test_data_class==2
                fprintf(clots,'%2.1f\t%2.1f',blobCentroid);
                fprintf(clots,'\n');
            end

            if test_data_class==3
                fprintf(cans,'%2.1f\t%2.1f',blobCentroid);
                fprintf(cans,'\n');
            end

            if test_data_class==4
                fprintf(robots,'%2.1f\t%2.1f',blobCentroid);
                fprintf(robots,'\n');
            end
        end
    end
end

%close the text files
fclose=(robots);
fclose=(clots);
fclose=(cans);
fclose=(cups);
```

A.2 Processing Code

Robot Control

```
//Made for the DBD04 Learning Robots Module
//Based on the OpenCV example 'blobs', http://ubaa.net/shared/processing/opencv/
//Jelle Dekker, Laura van Geel, Jasper de Kruiff, Vincent Klerkx
//Feb-Mar 2011, Industrial Design, TU/e

//Program to control the movements of an AdMoVeo robot to pick up different types
//of trash and transport them to corresponding dumpsites. Coordinates from trash
//are read from text files (generated with MathLab) and robot control uses blob
//detection from the OpenCV library.



import processing.serial.*;
import nl.tue.id.creapro.admoveo.*;
import hypermedia.video.*;
import java.awt.*;

AdMoVeo admoveo;

float Ycur = 0;                         //current Y-coordinate of robot
float Yold = 0;                         //old Y-coordinate of robot
float Xcur = 0;                         //current X-coordinate of robot
float Xold = 0;                         //old X-coordinate of robot
float Xtar = 0;                         //X-coordinate of current target
float Ytar = 0;                         //Y-coordinate of current target

int target = 0;                         //current target index (0 = dumpsite A, 1 =
dumpsite B, 2 = gargabe 1, 3 = garbage 2, 4 = garbage 3, etc)
int targetOld = 0;                      //previous target
float X = 0;                            //temporary storage of x coordinate robot
float Y = 0;                            //temporary storage of y coordinate robot
String[] linesA,linesB;                 //used for processing of text files
float[][] coordinates;                  //coordinates of trash and dumpsites
boolean hasfound=false;                 //used for robot detection

OpenCV opencv;

int w = 640;
int h = 480;
int threshold = 80;

void setup() {
  //load text file with coordinates of clots
  linesA = loadStrings("clots.txt");
  //load text file with coordinates of cans
  linesB = loadStrings("cans.txt");
  int coorLength=linesA.length+linesB.length+2;
  coordinates = new float[coorLength][2];
  //define dumpsite A coordinates (for clots)
  coordinates[0][0] = 560;
  coordinates[0][1] = 200;
  //define dumpsite B coordinates (for cans)
  coordinates[1][0] = 60;
  coordinates[1][1] = 420;
  //load clots coordinates
  for ( int i=0; i<linesA.length; i++ ) {
    String[] pieces = split(linesA[i], '\t');
    coordinates[i+2][0] = float(pieces[0]);
    coordinates[i+2][1] = float(pieces[1]);
  }
  //load cans coordinates
  for ( int a=0; a<linesB.length; a++ ) {
    String[] pieces2 = split(linesB[a], '\t');
    coordinates[a+2+(linesA.length)][0] = float(pieces2[0]);
    coordinates[a+2+(linesA.length)][1] = float(pieces2[1]);
  }
  target = 2;

  //initialize OpenCV
  println("Initialising OpenCV..");
  size( 2*w+40, h+20 );

  opencv = new OpenCV( this );
  opencv.capture(w,h);
  delay(15000);

  //initialize AdMoVeo robot
  println("Initialising Robot..");
  admoveo = new AdMoVeo(this, "/dev/tty.usbserial-A6005u0b");   //select correct COM port
  admoveo.getLeftMotor().forward();
  admoveo.getRightMotor().forward();
  admoveo.getLeftMotor().on();
  admoveo.getRightMotor().on();
}

void draw() {
  getPosition();

  //if robot is found on camera, continue with cleanup
  if (hasfound==true) {
    moveRobot();
  }
  //else try various manouvres to get back on the screen
  else {
    delay(1000);
    getPosition();
    while(hasfound==false) {
      admoveo.getLeftMotor().backward();
      admoveo.getRightMotor().backward();
      admoveo.getLeftMotor().setPower(150);
      admoveo.getRightMotor().setPower(150);
      delay(250);
      getPosition();
      admoveo.getLeftMotor().forward();
      admoveo.getRightMotor().backward();
      admoveo.getLeftMotor().setPower(150);
      admoveo.getRightMotor().setPower(150);
      delay(150);
      getPosition();
      admoveo.getLeftMotor().forward();
      admoveo.getRightMotor().forward();
      admoveo.getLeftMotor().setPower(150);
      admoveo.getRightMotor().setPower(150);
      delay(150);
      stopRobot();
      delay(250);
    }
  }
}

//Gets current coordinates of robot using blob detection from OpenCV
void getPosition() {
  background(0);
  opencv.read();
  //opencv.flip( OpenCV.FLIP_HORIZONTAL );

  image( opencv.image(), 10, 10 );                // RGB image
  image( opencv.image(OpenCV.GRAY), 20+w, 10 );   // GRAY image
  image( opencv.image(OpenCV.MEMORY), 10, 20+h ); // image in memory

  opencv.absDiff();
  opencv.threshold(threshold);

  // working with blobs
  Blob[] blobs = opencv.blobs( 2600, w*h/3, 20, true);  //minimal blobsize is 2600, so
only robot will be detected as blob

  noFill();

  pushMatrix();
  translate(10,10);
  for( int i=0; i<blobs.length; i++ ) {
    Rectangle bounding_rect = blobs[i].rectangle;
    float area = blobs[i].area;
    float circumference = blobs[i].length;
    Point centroid = blobs[i].centroid;
    Point[] points = blobs[i].points;

    //put current x-coordinate of robot in X
    X = centroid.x;
    //put current y-coordinate of robot in Y
    Y = centroid.y;

     //Determine the position of the robot when the system is initiated
    if (hasfound==false) {
      Xcur = X;
      Ycur = Y;
    }

    // rectangle
    noFill();
    stroke( blobs[i].isHole ? 128 : 64 );
    rect( bounding_rect.x, bounding_rect.y, bounding_rect.width, bounding_rect.height );

    noFill();
    stroke(0,0,255);
    if ( points.length>0 ) {
      beginShape();
      for( int j=0; j<points.length; j++ ) {
        vertex( points[j].x, points[j].y );
      }
      endShape(CLOSE);
    }
    noStroke();
  }
  popMatrix();

  //if robot is detected on camera, hasfound=true
  if (blobs.length>0) {
    hasfound=true;
  }
  //else hasfound=false
  else {
    hasfound=false;
  }
}

//moves the robot to the current target when detected on camera
void moveRobot() {
  //stop robot and get current position
  stopRobot();
  delay(50);
  getPosition();

  //save old coordinates and load the current ones
  Yold = Ycur;
  Xold = Xcur;
  Xcur = X;
  Ycur = Y;

  //load the target coordinates
  Xtar = coordinates[target][0];
  Ytar = coordinates[target][1];

  //if current location is quite close to target location, target has been reached
  if (abs(Xcur - Xtar) < 60.0 && abs(Ycur - Ytar) < 60.0) {
    //if target was a piece of garbage, turn on blue LED
    if (target>targetOld) {
      admoveo.getBlueLed().setPower(255);
      admoveo.getRedLed().setPower(0);
      targetOld=target;
      getPosition();
    }
    //if target was a dumpsite, turn on red LED and turn around
    else if(target<2) {
      admoveo.getRedLed().setPower(255);
      admoveo.getBlueLed().setPower(0);
      admoveo.getLeftMotor().backward();
      admoveo.getRightMotor().backward();
      admoveo.getLeftMotor().setPower(200);
      admoveo.getRightMotor().setPower(200);
      delay(1000);
      getPosition();
      admoveo.getLeftMotor().forward();
      admoveo.getRightMotor().backward();
      admoveo.getLeftMotor().setPower(250);
      admoveo.getRightMotor().setPower(250);
      delay(150);
      getPosition();
    }
    stopRobot();
    println("Destination Reached!");
    //determine the new target (if any)
    getTarget();
  }
  //if the target has not yet been reached move closer to the target
  else {
    //move slightly forward
    admoveo.getLeftMotor().forward();
    admoveo.getRightMotor().forward();
    admoveo.getLeftMotor().setPower(150);
    admoveo.getRightMotor().setPower(150);
    delay(300);
    getPosition();

//the clever part that controls the actual movements of the robot
    //if current x-coordinate of the robot is closer to the target than last check,
    //but y-coordinate is further away, action is required
    if ((abs(Xtar-Xold)<abs(Xtar-Xcur))&&(abs(Ytar-Yold)>abs(Ytar-Ycur))) {
      //through logic we determine whether the robot should move left or right in this
situation
      if (Xcur>Xold && Ycur<Yold) {
        goLeft();
      }
      else if(Xcur>Xold && Ycur>Yold) {
        goRight();
      }
      else if(Xcur<Xold && Ycur<Yold) {
        goRight();
      }
      else if(Xcur<Xold && Ycur>Yold) {
        goLeft();
      }
    }
    //if current y-coordinate of the robot is closer to the target than last check,
    //but x-coordinate is further away, action is required
    else if((abs(Xtar-Xold)>abs(Xtar-Xcur))&&(abs(Ytar-Yold)<abs(Ytar-Ycur))) {
      //through logic we determine whether the robot should move left or right in this
situation
      if (Xcur>Xold && Ycur>Yold) {
        goLeft();
      }
      else if(Xcur>Xold && Ycur<Yold) {
        goRight();
      }
      else if(Xcur<Xold && Ycur>Yold) {
        goRight();
      }
      else if(Xcur<Xold && Ycur<Yold) {
        goLeft();
      }
    }
    //if both x and y coordinates are further away from the target than last check, make a
u-turn
    else if((abs(Xtar-Xold)<abs(Xtar-Xcur))&&(abs(Ytar-Yold)<abs(Ytar-Ycur))) {
      uTurn();
    }
  }
}

//makes the robot go left
void goLeft() {
  getPosition();
  admoveo.getLeftMotor().backward();
  admoveo.getRightMotor().forward();
  admoveo.getLeftMotor().setPower(150);
  admoveo.getRightMotor().setPower(150);
  delay(50);
  getPosition();
}

//makes the robot go right
void goRight() {
  getPosition();
  admoveo.getLeftMotor().forward();
  admoveo.getRightMotor().backward();
  admoveo.getLeftMotor().setPower(150);
  admoveo.getRightMotor().setPower(150);
  delay(50);
  getPosition();
```

```
}

//makes the robot turn 180 degrees
void uTurn() {
  getPosition();
  admoveo.getLeftMotor().forward();
  admoveo.getRightMotor().backward();
  admoveo.getLeftMotor().setPower(150);
  admoveo.getRightMotor().setPower(150);
  delay(1000);
  getPosition();
  admoveo.getLeftMotor().forward();
  admoveo.getRightMotor().forward();
  admoveo.getLeftMotor().setPower(150);
  admoveo.getRightMotor().setPower(150);
  delay(100);
  getPosition();
}

//makes the robot stop
void stopRobot() {
  admoveo.getLeftMotor().setPower(0);
  admoveo.getRightMotor().setPower(0);
}

//determines the current target of the robot
void getTarget() {
  //if previous target was trash
  if (target>1) {
    //if not all clots are cleaned up
    if (target<(linesA.length+2)) {
      //make current target dumpsite A (for clots)
      target=0;
    }
    //else, if all clots are cleaned up
    else {
      //make current target dumpsite B (for cans)
      target=1;
    }
  }
  else {
    //if previous target was a dumpsite
    //make current target the next piece of trash
    target=targetOld+1;
  }
}

//required for OpenCV
void keyPressed() {
  if ( key==' ' ) opencv.remember();
}

//required for OpenCV
public void stop() {
  opencv.stop();
  super.stop();
}
```

A.3 Simulation Code

```
//Made for the DBD04 Learning Robots Module
//Jelle Dekker, Laura van Geel, Jasper de Kruiff, Vincent Klerkx
//Feb-Mar 2011, Industrial Design, TU/e

//Program to control the movements of virtual agents to pick up different types
//of trash and transport them to corresponding dumpsites. Coordinates of trash
//are read from text files (generated with MathLab)

//declaration of variables; variables are explained throughout the code
import seltar.motion.*;
TrashA[] trashA;
Agent[] teamAgents = new Agent[4];
int trashbinSize = 125;
int selectedTrash = 0;
int selectedAgent = 0;
int currentTrashA = 0;
String[] linesA;
String[] linesB;
float[][] coordinates;
PImage imgA;
PImage imgB;
PImage imgC;

//initialization of values for variables
void setup()
{
  //size of screen is set to 640◊730
  size(640,730);
  smooth();
  frameRate(90);
  //linesA and linesB are arrays which hold the coordinates for the instances of the two
types of trash
  //the content of these arrays is variable and defined by two external .txt files
  //here the content of the two .txt files is loaded into arrays linesA and linesB
  linesA = loadStrings("coordinatesA.txt");
  linesB = loadStrings("coordinatesB.txt");
  //imgA and imgB hold the images for the two types of trash, and imgC holds the image for
the agents
  imgA = loadImage("blikje.png");
  imgB = loadImage("propje.png");
  imgC = loadImage("AdmoveoTopview.png");
  //a float variable is created to hold the coordinates of each instance when called
  coordinates = new float[linesA.length+linesB.length][2];
  //x trash instances are created; x is defined by the total amount of instances of the
two types of trash
  trashA = new TrashA[linesA.length+linesB.length];

  //trash instances of type A are initialized; amount of instances is variable and defined
by the content of the
  //.txt file which is loaded into the array linesA (coordinatesA.txt)
  //the .txt file holds the coordinates of the instances of 'linesA'; inside this .txt
file the x-value and y-value
  //of each instance is separated by a TAB ('\t'), and the index# of each instance is
separated by a RETURN
  for ( int i=0; i<linesA.length; i++ ) {
    //an array of strings called 'pieces' is created to hold the x- and y-coordinates of
each instance i separately
    String[] pieces = split(linesA[i], '\t');
    //each instance i of trashA has an x-value, defined by the first float in the two-
dimensional array 'coordinates'
    coordinates[i][0] = float(pieces[0]);
    //each instance i of trashA has an y-value, defined by the second float in the two-
dimensional array 'coordinates'
    coordinates[i][1] = float(pieces[1]);
    //the x- and y-coordinates of each instance i are printed, to enable error tracing
    println(coordinates[i][0]);
    println(coordinates[i][1]);
  }
  //trash instances of type B are initialized; amount of instances is variable and defined
by the difference between the
  //total length of both .txt files (linesA+linesB), and the length of the first .txt file
(loaded into linesA),
  //which results in the length of the second .txt file (which was loaded into linesB)
  for ( int i=linesA.length; i<linesA.length+linesB.length; i++ ) {
    //an array of strings called 'pieces' is created to hold the x- and y-coordinates of
each instance i separately
    String[] pieces = split(linesB[i-linesA.length], '\t');
    //each instance i of trashA has an x-value, defined by the first float in the two-
dimensional array 'coordinates'
    coordinates[i][0] = float(pieces[0]);
    //each instance i of trashA has an y-value, defined by the second float in the two-
dimensional array 'coordinates'
    coordinates[i][1] = float(pieces[1]);
    //the x- and y-coordinates of each instance i are printed, to enable error tracing
    println(coordinates[i][0]);
    println(coordinates[i][1]);
  }

  //as long as the variable selectedTrash is smaller than the total amount of trash
instances (the length of the array coordinates),
  //trashA instances and Agent instances are created (this means it is impossible to have
more Agent instances than trash instances,
  //which is a point of weakness in this simulation)
  for (int selectedTrash = 0; selectedTrash < coordinates.length; selectedTrash ++ ) {
    trashA[selectedTrash] = new TrashA(coordinates[selectedTrash][0], coordinates
[selectedTrash][1]+125);
    //the Agents are inserted randomly into the space in between the two 'trashbins'
    teamAgents[selectedAgent] = new Agent(random(width),random(trashbinSize, height-
trashbinSize), selectedAgent);
    //if selectedAgent is 1, this means the first Agent instance is used and assigned to
the trash instance which is created at
    //the moment; if selectedAgent is 2, this means the second Agent instance is used and
assigned to that trash instance
    //when the variable int selectedAgent is greater than the total amount of Agent
instances, it is reset to prevent that no Agent instance
    //can be found to assign the trash instance to; by resetting selectedAgent we make
sure the upcoming trash instance is assigned to the
    //first Agent instance and the counting process will start again, as long as the
variable selectedTrash is smaller than the total
    //amount of trash instances
    selectedAgent=(selectedAgent+1)%teamAgents.length;
  }
  //the variable currentTrashA is used to make the system aware of which trash instance
should be removed by which Agent instance
  //each time a trash instance is created, currentTrashA is set to the index# of the trash
instance which is created, divided
  //(using the modulus operator '%') by the total amount of Agent instances
  //this means every time a trash instance is created, immediately one of the Agent
instances is assigned to it
  currentTrashA = selectedAgent;
  //the variable int currentTrashA is printed to enable error tracing
  println(currentTrashA);
}
```

```
void draw()
{
  //background color is set to black
  background(0);

  //the function displayTrashbins() is called to display the trashbins
  displayTrashbins();

  //if variable int currentTrashA is smaller than the amount of instances of trash of type
A, trash is displayed
  if (currentTrashA<trashA.length) {
    //as long as variable int selectedtrash is smaller than the total amount of instances
of trash of type A, trash is displayed as type A
    for (int selectedTrash = 0; selectedTrash < linesA.length; selectedTrash ++ ) {
      trashA[selectedTrash].displayA();
    }
    //as long as variable int selectedTrashB is bigger than the total amount of instances
of trash of type A and smaller than the total amount of trash instances (linesA + linesB),
trash is displayed as type B
    for (int selectedTrashB = linesA.length; selectedTrashB < linesA.length+linesB.length;
selectedTrashB ++ ) {
      trashA[selectedTrashB].displayB();
    }
  }

//as long as variable j is smaller than the total amount of Agent instances (the length of
the array 'teamAgents'),
//Agent instances are displayed
  for (int j = 0; j < teamAgents.length; j ++ ) {
    teamAgents[j].display();
  }

//if variable int currentTrashA is bigger than the total amount of trash instances (if all
trash instances are removed),
//the displayA and displayB instances are called to prevent nullpointer errors and to make
sure the program will keep
//looping instead of crash
  if (currentTrashA>=trashA.length) {
    for (int i = 0; i < linesA.length; i ++ ) {
      trashA[i].displayA();
    }
    for (int s = linesA.length; s < linesA.length+linesB.length; s ++ ) {
      trashA[s].displayB();
    }
 }
}


//class for constructing the Agent instances
class Agent
{
  int grabState = 0;
  Motion m;
  int a;

//when Agent instances are created, they are assigned an x-value, a y-value and a variable
to determine which trash instances to remove
  Agent(float X, float Y, int assignedTrash)
  {
    //the movement of each Agent is stored inside variable m of class Motion
(seltar.motion is an external class downloaded from the Processing website)
    m = new Motion(X,Y);
    //the variable to determine which trash instance to remove is called 'a'
    a = assignedTrash;
  }

  void followTrashA() {
    //if a is smaller than the total amount of trash instances, motion m of the current
Agent instance will follow the currently selected trash instance
    if (a < trashA.length) {
      m.followTo(trashA[a].x,trashA[a].y);
      m.wrap(0,0,width,height);
      m.move();
    }
    //if a is bigger than the total amount of trash instances (if all trash is removed),
the current Agent instance will retrieve grabState 2 (move to the middle)
    else
      if (a > (trashA.length)-1) {
        grabState = 2;
      }
  }

//the grabState of the current Agent instance is defined
  void checkBaggage() {
    //if a is smaller than the total amount of trash instances, the function
defineGrabState is called
    if (a < trashA.length) {
      defineGrabState();
    }
    //if a is bigger than the total amount of trash instances, the variable int grabState
is set to 2
    else
      if (a > (trashA.length)-1) {
        grabState = 2;
      }
  }

//the grabState of the current Agent instance is further defined
  void defineGrabState() {
    //if the current x-state and the current y-state of the current Agent instance are
within a circle of 15 pixels around the trash instance
    //it should remove ('if the current Agent has grabbed the right trash instance'),
grabState is set to 1 ('move to trash')
    if(m.getX() > ((trashA[a].x)-15) && m.getX() < ((trashA[a].x)+15) && m.getY() >
((trashA[a].y)-15) && m.getY() < ((trashA[a].y)+15)) {
      grabState = 1;
    }
    //if the y-state of the current trash instance (the trash instance which the Agent
instance is currently removing) is in between 600
    //and 800, this means it is inside the trashbin and grabState is set to 2 ('move back
to middle of stage')
    if(trashA[a].y > 600 && trashA[a].y < 800) {
      grabState = 2;
    }
    //if the y-state of the current trash instance (the trash instance which the Agent
instance is currently removing) is in between 0
    //and 100, this means it is inside the trashbin and grabState is set to 2 ('move back
to middle of stage')
    if(trashA[a].y < 100 && trashA[a].y > 0) {
      grabState = 2;
    }
    //if an Agent instance has moved a trash instance into a trashbin (if grabState is 2)
and if it is back in the middle of the stage,
    //grabState is set to 0 ('look for trash instances')
    if(m.getX() > (320-15) && m.getX() < (320+15) && m.getY() > (365-15) && m.getY() <
(365+15) && grabState == 2) {
      grabState = 0;
      //a new trash instance is assigned to be removed
      currentTrashA = currentTrashA + 1;
      a=currentTrashA;
    }
  }

//actions are assigned to the possible states the variable int grabState can take
  void enableMove() {
    //the state of grabState is checked and modified if needed
    checkBaggage();
    //if grabState is 0, this means the current Agent instance is 'looking for the current
trash instance' but has not found it yet
    if (grabState==0) {
      //as long as grabState is 0, the current Agent instance will move towards the
current trash instance
      followTrashA();
    }
    //if grabState is 1, this means the current Agent has 'grabbed' the current trash
object
    if (grabState==1) {
      //if a is smaller than the amount of trash instances of type A, the Agent instance
will move the trash instance towards trashbin A
      if (a < linesA.length) {
        m.followTo(400,650);
        m.wrap(0,0,width,height);
        //the trash instance of type A is 'sticked to' the Agent instance as long as the
Agent instance is moving towards trashbin A
        trashA[a].x = m.getX();
        trashA[a].y = m.getY();
        m.move();
      }
      //if a is bigger than the amount of trash instances of type A, this means the
selected trash instance is of type B, and so the Agent instance
      //will move the trash instance towards trashbin B
      if (a >= linesA.length) {
        println(currentTrashA);
        m.followTo(400,0);
        m.wrap(0,0,width,height);
        //the trash instance of type B is 'sticked to' the Agent instance as long as the
Agent instance is moving towards trashbin B
        trashA[a].x = m.getX();
        trashA[a].y = m.getY();
        m.move();
      }
    }
    //if grabState is 2, this means the trash instance has successfully been dropped into
the right trashbin, and the current Agent instance will
    //move back to the middle of the stage
    if (grabState==2) { //
      m.followTo(320,365);
      m.wrap(0,0,width,height);
      m.move();
    }
  }

//this is the main function in which all the sub-functions of the current Agent instance
are called
  void display()
  {
    drawVector(20);
    enableMove();
  }
```

```
//this method actually draws the Agent instances; without this function the trash
instances will be removed but the
//Agent instances will be ëinvisibleí
  void drawVector(float scayl) {
    stroke(200);
    float arrowsize = 10;
    //the Agent instance is pointed towards the current trash instance
    pushMatrix();
    translate(m.getX(),m.getY());
    rotate(m.v.getDirection());
    float len = m.v.getVelocity()*scayl;

    // Draw three lines to make an arrow (draw pointing up since we've rotate to the
proper direction)
    image(imgC,0-20,0-18);
    line(0,0,len,0);
    line(len,0,len-arrowsize,+arrowsize/2);
    line(len,0,len-arrowsize,-arrowsize/2);
    popMatrix();
  }
}


//class for constructing the trash (TrashA) instances; the name TrashA is universal and is
not related to the type of trash
class TrashA {

// TrashA's variables
  float x,y;

// TrashA constructor; an x-value and a y-value is assigned to each trash instance
  TrashA(float tempX, float tempY) {
    x = tempX;
    y = tempY;
  }

//when the function displayA is called, the instance will be displayed as imgA (which
holds the file 'blikje.png')
  void displayA() {
     image(imgA,x,y);
  }

//when the function displayB is called, the instance will be displayed as imgB (which
holds the file 'propje.png')
   void displayB() {
     image(imgB,x,y);
  }
}

//function for displaying both trashbins
void displayTrashbins()  {
    stroke(128);
    fill(#FFEFC6);
    strokeWeight(2);
    //variable trashbinSize is used to easily change the size of both trashbins without
having to change the entire code
    //the first trashbin is constructed as a rectangle with color (#FFEFC6)
    rect(0+20,0+20,width-40,trashbinSize);
    fill(#FF2600);
    //the second trashbin is constructed as a rectangle with color (#FF2600)
    rect(20,height-trashbinSize-20,width-40,trashbinSize);
}
```