Learning Robots

DBD04 | TU/e ID

Robbert van Vliet
070677
Rik Vegt
0752217
Martijn Kors
0751581
Michael Geertshuis
0755051

Lecturers:
dr.ir. E.I. Barakova
dr. J. Hu Pdeng Men

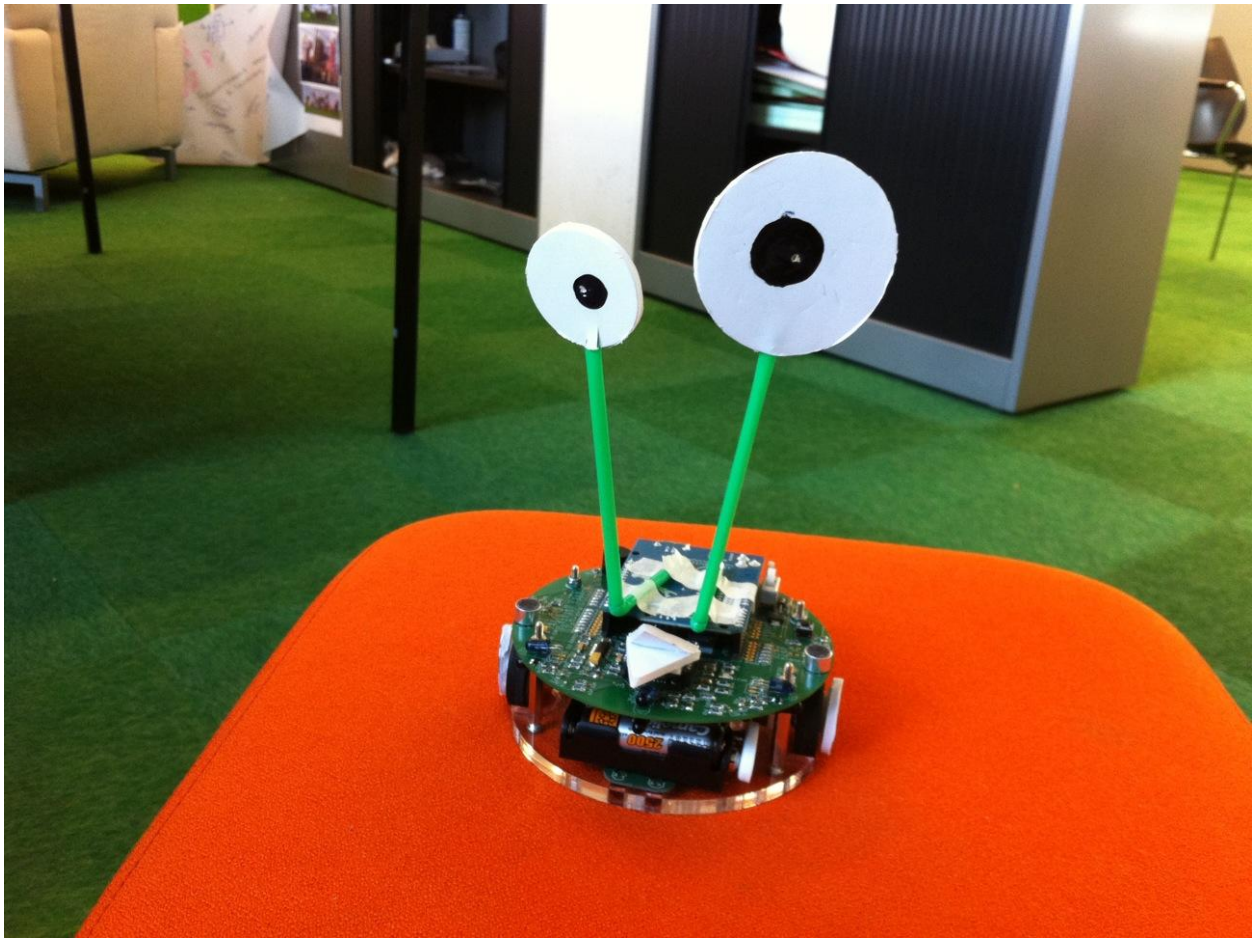
25-3-2011

Index

Introduction

'Learning Robots' is a two week module of which the first week is mainly used for lectures and the assimilation of relevant information on machine learning through various learning algorithms or neural networks. The second half of the assignment is used for the practical implementation of a learning algorithm and creating connections between various software environments needed to accomplish this. The main goal is to find a suitable concept to be able to investigate the workings of various learning algorithms in order to get a hands-on experience with learning robots.

The result is an AdMoVeo based robot that can be remotely controlled by a hand drawn arrow shown to a webcam. The system learned to recognize arrows in different positions that control the movement of the robot. Also a start was made for implementing a Q-learning algorithm to enable the robot to learn to avoid objects when encountered.



Goals

The main goal for this deliverable was to find a suitable concept to be able to investigate the workings of various learning algorithms in order to get a hands-on experience with learning robots. Keeping the concept at a basic level (a toy/game) enables a better focus on applying the algorithms and getting them to work smoothly.

Artificial neural networks

The ability to learn as is implemented during this modules is based on artificial neural networks. These networks are mathematical algorithms that are able to learn mappings between input and output states through supervised learning, or to cluster incoming information in an unsupervised manner [Nehmzov '03].

Concept

The final concept is a robot that can be controlled by showing an image drawn on paper. This robot will try to accomplish the goal using a neural gas learning algorithm for image recognition and with controlling software for executing commands. In this way a user can draw an image on a piece of paper and use this to drive the robot around with simple commands. (see Figure 1)

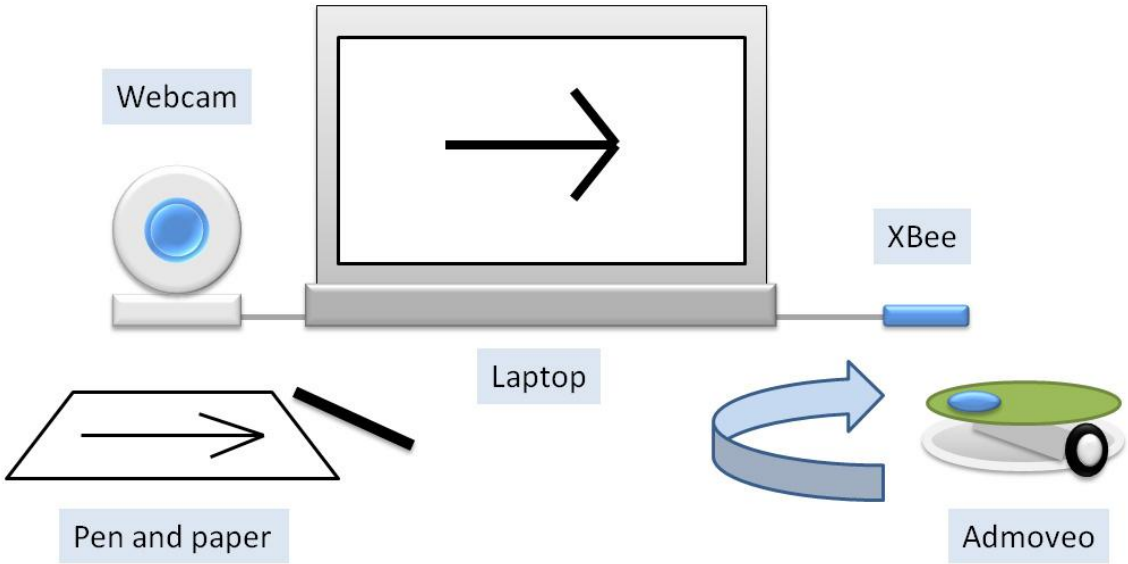


Figure 1. Illustration of the final concept: controlling the robot

In a secondary part a certain level of object avoidance was attempted to reach through the use of a Q-learning algorithm. The goal was to interrupt normal control when an object was encountered. The learning algorithm should find the best combination of actions to drive around the object and continue in the same direction it came from. (see Figure 2)

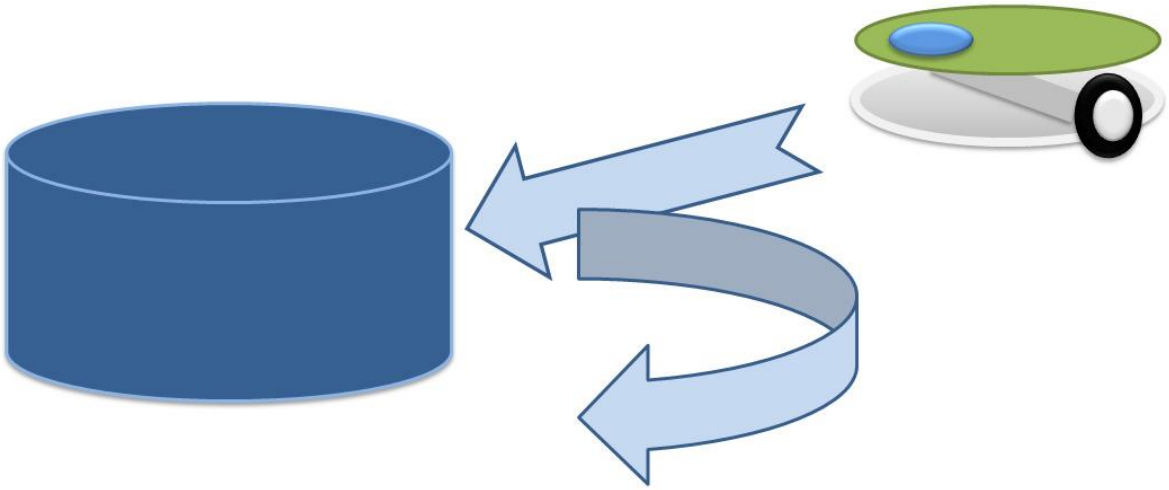


Figure 2. Illustration of the final concept: avoiding collisions

Method

The final results are developed through multiple steps with the general concept in mind. Depending on the success of each step within the available timeframe, goals can be adjusted accordingly. Gaining knowledge on basic learning algorithms through experimentation, discussion and literature is one of the first steps to be taken in the first week of the module. At the end of the week the described concept is defined.

Introduction of platforms

To get the concept working in time, the decision was made to use some existing tools for the different goals. These tools (mostly software code) were based on different programming platforms. The background of these platforms and tools will be described briefly below.

AdMoVeo

The robot platform that is used is called AdMoVeo and it is developed by the TU/e. On board it has a set of simple sensors and actuators that can be interfaced with through Processing. Available sensors are two line sensors, three distance sensors with a range of about 15cm, two light sensors and two sound samplers. The actuators are the two motors, a buzzer and a RGB LED at the bottom of the robot. Also an Xbee controller can be configured for data communication and an Arduino board provides the overall control for the robot.

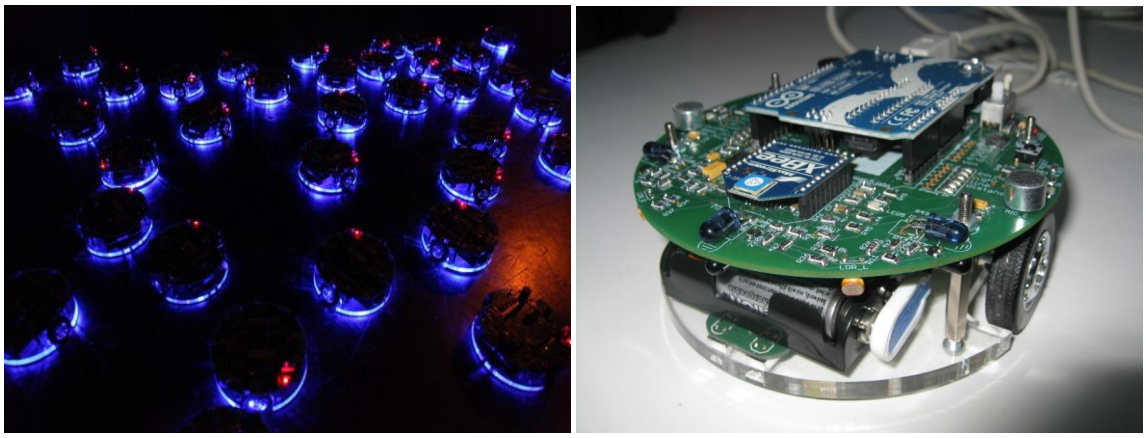


Figure 3. The AdMoVeo in action

Arduino

The Arduino platform is used for directly controlling the robot making use of an extended version of the Firmata protocol called Iduino. Serial data communicated to the AdMoVeo will be interpreted by Iduino and executed on the platform.

Processing

Processing was used for sending data from the laptop to the AdMoveo using the creapro library (Creative programming for Designers) which enables direct communication with Iduino. The program that was written worked as an intermediate solution between the Neural Gas Algorithm running on Matlab and the robot. By configuring a TCP/IP client in processing it was possible to receive data directly from Matlab and use it to control the robot. The complete processing code can be found in 'Appendix C – Supporting code', page 31.

Matlab

In the Matlab programming language, the neural gas learning algorithm was implemented. [Vesanto, '11] This algorithm is first trained to recognize arrows, after which it could be used to perform these recognition on real time webcam snapshots. A more detailed explanation of the different steps in this written piece of code will be explained later in this document.

Webcam

A webcam is used to interface with the robot. In Matlab samples are taken from the video stream at a predetermined rate to be compared to the trained library of the Neural Gas algorithm. When the input from the webcam is recognized, Matlab will send the corresponding action (a number) to Processing, which in turn sends it to the AdMoVeo.

Platform interconnectivity

To give a more clear view on the way the different hard- and software components are connected a diagram was sketched showing the platform interconnectivity. The rectangular elements stand for the various hard- and software components, the circular attachments contain subroutines that enable interconnectivity between the platforms, some are connected directly others through wireless protocols.

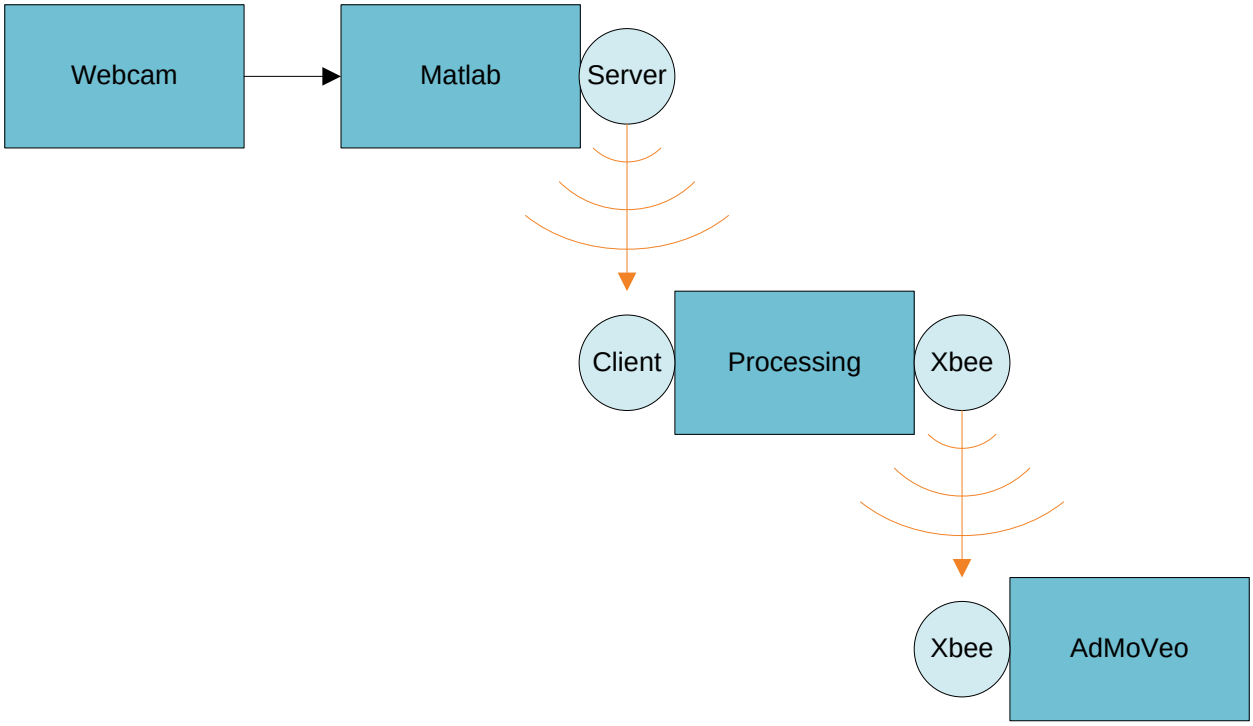


Figure 4. A visualisation of the communication between different platforms

The trained neural gas network receives its input from a webcam connected to the pc that is running the server on Matlab. In Matlab the incoming feed is analyzed and the resulting class is sent to the server. The server is necessary to make a connection possible between Matlab and processing or calculation and execution. The client running in processing receives a stream of data from the server and decides what action to perform. Through the Xbee module the active command is sent to the AdMoveo robot making it react to the hand-drawn arrows as seen by the webcam.

Basic concept

As described above, part of the concept was to achieve a basic level, after which it could be extended if implemented successfully. In this chapter the basic concept is described. The next chapter will explain the extension of the concept.

Image recognition

For recognizing hand-drawn arrows the neural gas algorithm was implemented, which is categorized as an unsupervised learning system. The advantage of this algorithm is that it can be used to detect similarities in input signals and separate those signals into a predefined number of classes. In the case of the arrow recognition system this means that the input signal from the webcam is compared to a neural gas network trained in separating four different pointing directions. The written code consists of three main modules:

1. The image preprocessing code,
2. the learning neural gas algorithm and
3. the image-capturing and comparison with the taught training set.

Preprocessing code

As the neural gas algorithm is capable of 'discovering' common patterns in almost any input, it may not always lead to the desired output. Even though neural gas should be capable of ignoring noise, to make optimal use of its capabilities, it is best and fastest to provide the input in a such way it is already emphasizing on the details it has to recognize. The preprocessing is done in the steps as described in the following graph and are visualized in Figure 5.

First, the colors of the image are converted to grayscale if they aren't already. By comparing all pixels in the image to a pre-set threshold, the image is made binary. This makes it easier for the following steps that checks whether whole horizontal and vertical lines in the image are white or not. If so, they are removed. To be able to compare the different images later on, they will have to be rescaled to the same dimensions, which are in this case 100*100 pixels.

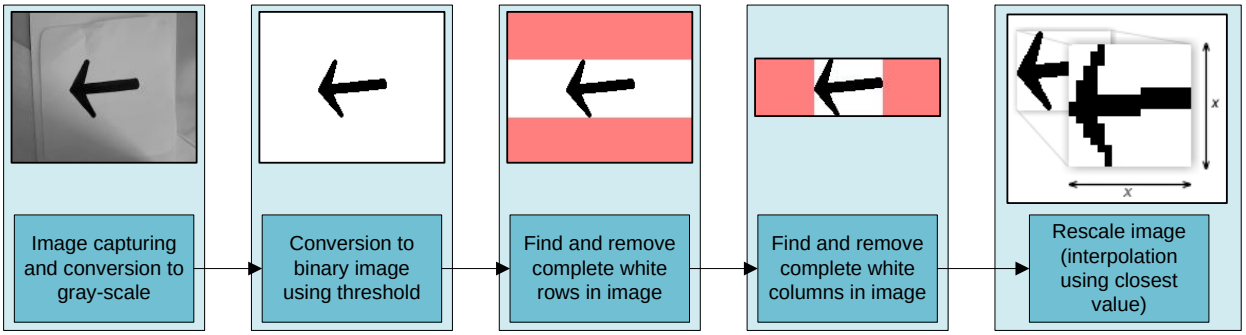


Figure 5. Steps for processing the input images

Second step in the preprocessing involves the calculations to provide the neural gas with only one line of data (see Figure 6). The sum of each vertical line is calculated, as well as each horizontal line. Finally all the calculated values are put in a single dimensioned array of 200 (100+100) values.

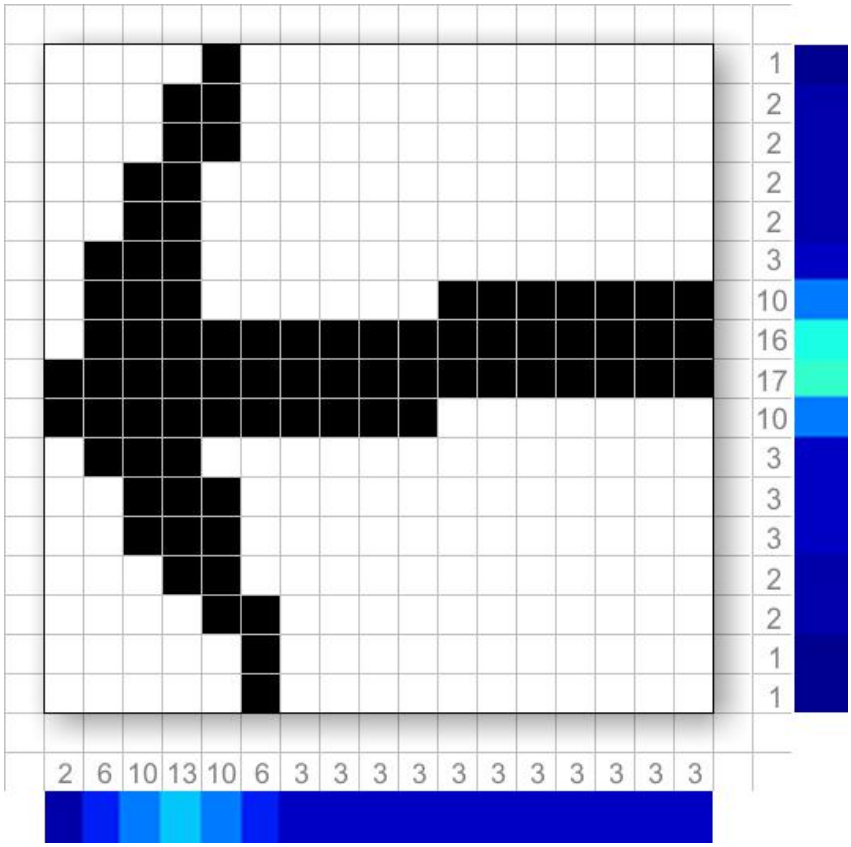


Figure 6. Calculation of values for neural gas

Learning neural gas

As the word 'learning' implies, the neural gas will only be able to perform recognition of images in case the possible images were taught in advance. Therefore a set of 40 different images of arrows was used to train the neural gas (see Figure 7). Each image was first processed by the algorithm described in 'Preprocessing code'. The training involved 60 iterations, an amount which proved to be sufficient for reliable categorization. Five categories were defined, since the algorithm had to learn to distinguish between an arrow pointing up, left, right and down and a blank piece of paper.

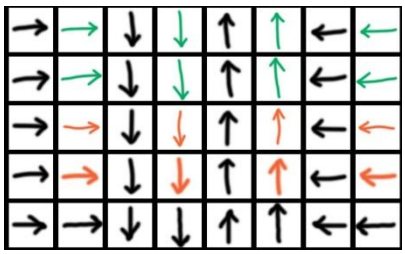


Figure 7. Training set of arrows

After training, the neurons that are used for comparison later on can be visualized as in Figure 8. As can be seen from the figure, five distinct categories were found ('read' from top to bottom). On the horizontal axis, the first 100 values are from the horizontal axis, where the second 100 are for the vertical axis. The upper category (1, red colored) is the blank category, where the others categorized as arrows pointing right, down, left and up respectively.

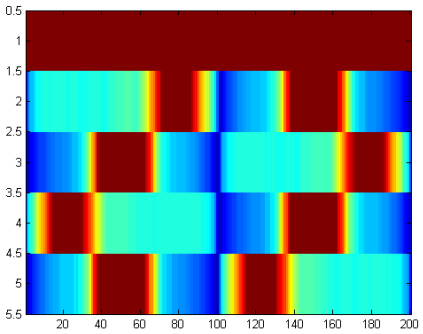


Figure 8. Visualization of the trained neurons

Image-capturing and recognition

The image of the connected camera is captured with a preset interval, which is currently 10 Hz. After a snapshot is taken, the image will be processed as described in 'Preprocessing code' (above) to distill the needed information from it. It will then be compared using the trained neurons from the neural gas algorithm. The algorithm will then find the category that comes closest to the captured image.

Controlling AdMoVeo

As a result from the analysis of the image of the arrow by the Matlab algorithm a number is given. This number corresponds to the category to which the arrow belongs and is depending on the direction of the arrow. The number of the corresponding category will be sent to the previously set-up TCP/IP connection and will be read by a piece of Processing code. The read out value is used to control the AdMoVeo, which is done in the same Processing code (Appendix C – Supporting code, page 31) using the protocol provided. The next list shows what numbers correspond to what actions, the robot will keep performing the action as long as there is an input generated at the webcam:

0 – Stop

1 – Rotate right

2 – Move backward

3 – Rotate left

4 – Move forward

Q-learning extension

A start was made with the implementation of a Q-learning algorithm [Harmon, '96]. The goal for the robot was to be able to learn to avoid objects when encountered and continue on its original course. To reach this goal the first intention was to make use of the distance sensors of the robot, but when the first version of the algorithm was tested it was found that the sensors weren't accurate enough to do the job. Therefore the decision was made to leave out this part in the final presentation of the robot. A description is added to give an impression of the idea.

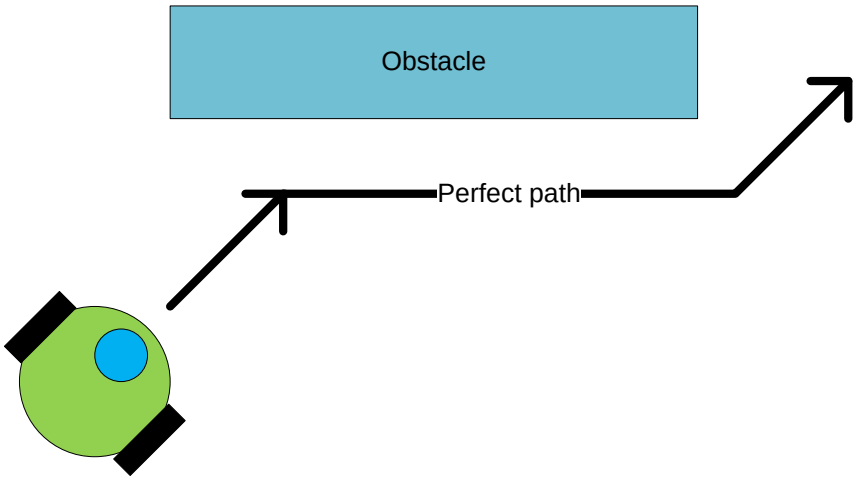


Figure 9. The avoidance of an object

The Q-value grid of the algorithm had three dimensions; the first dimension is a result of the input of three distance sensors on the AdMoVeo. The result is used to find out how far the robot is away from a possible obstruction, if an object is detected the Q-learning will take over from the neural gas algorithm. When this takeover happens the second dimension is set to zero. This dimension changes as the robot decides to turn to the left or the right. This value is important, because the goal of the robot is to get back to its original direction. The reward should be higher when the robot is closer to its original rotation and when the robot doesn't see anything on its distance sensors. The third dimension exists of the different possible actions the robot can perform, so turning left, turning right and moving forward.

Ultimately the robot should learn the best way to drive around an object and move back to its original rotation when the object is cleared.

Equation 1. The Q-learning equation [Harmon, '96]

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{old\ value} + \underbrace{\alpha_t(s_t, a_t)}_{learning\ rate} \times \overbrace{[\underbrace{r_t}_{reward} + \underbrace{\gamma}_{discount\ factor} \underbrace{\max_a Q(s_{t+1}, a)}_{max\ future\ value}}^{expected\ discounted\ reward} - \overbrace{Q(s_t, a_t)}^{old\ value}]$$

Because the algorithm couldn't be implemented accurately it was not possible to find the optimal values for the learning rate, rewards and discount factor. This made the initial model very unpredictable, but looking at the robot it did seem to go through all the calculations in the program. In 'Appendix B - Implemented learning algorithms' (page 21) the processing code can be found. The described algorithm works, but doesn't perform even close to optimal as tested with the AdMoVeo robot.

Conclusion

We successfully applied learning algorithms and investigated different possibilities for using them in interaction designs. Our goal was to use a toy or game like design to provide us with a certain grade of freedom for choosing interactions and learning algorithms to get a broad understanding of their workings.

The neural gas algorithm proved very useful for recognizing hand-drawn shapes, being accurate enough to recognize different rotations and robust enough to identify arrows even when they were drawn partially or in a strange way.

The Q-learning algorithm worked partially at some point of our process, which showed us clearly how actions and sensor readings evolve. A full implementation wasn't possible because it was very hard to debug the program as the sensors of AdMoVeo weren't suitable for our purpose, creating strange behavior.

Overall we were very happy with the process and we think we picked the right algorithms for the intended purposes. By using a hands-on approach to investigate and elaborate we were able to explore a broad scope of learning algorithms and their capabilities and limitations.

Discussion

Upon deciding which concept to implement there was some discussion on the type of signs to use. Any distinctive image could be used for each action the AdMoVeo would perform. Focusing on hand drawn arrows was both a challenge and an opportunity, because the arrows are directional and their position relative to the webcam is variable. The processed images to which the Neural GAS algorithm image recognition is applied need to be as consistent as possible in order to create a robust and reliable method of image recognition. Besides the four directional categories a blank category was added. Unfortunately the blank category did not work that well, since the image never became totally blank, due to the automatic contrast of the camera used. This resulted in dark areas in the images, which slightly hindered image recognition of a situation in which no arrow was shown but still categorized this situation as one of the directional categories.

The concept as it was presented could be a toy or a game; it would have been better if a more relevant goal was addressed, getting a more significant value from using the learning algorithms.

Future development

As discussed before, the Q-learning algorithm for letting the AdMoVeo avoid collisions with objects was not yet fully functional implemented. This could be the first step in developing the concept further. Since a learning algorithm is used for image recognition, it could be also used for a more continuous learning curve. In other words: new and unknown images could be added to the training set.

References

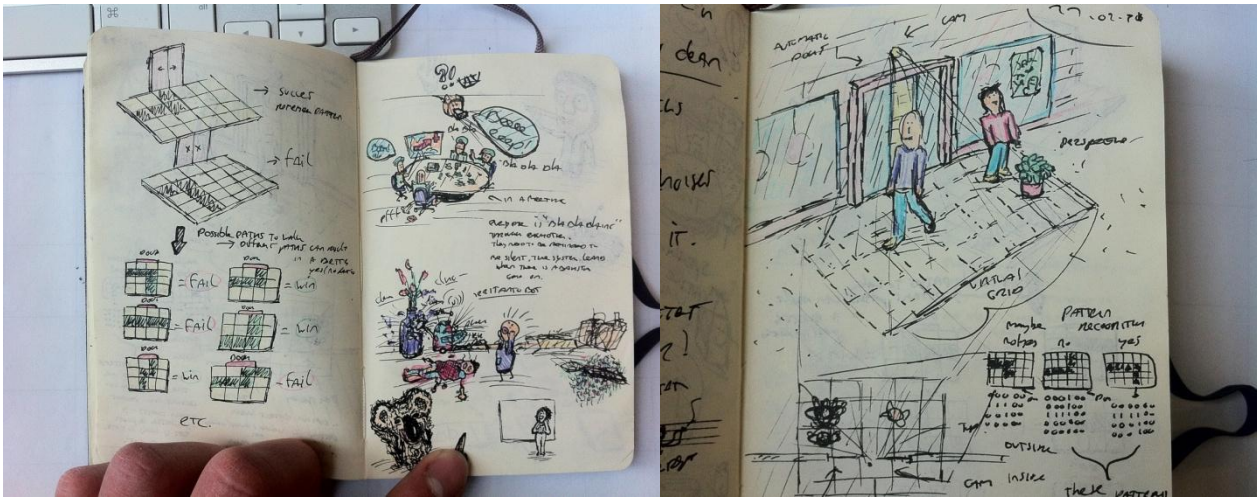
1. U. Nehmzow, Mobile Robotics: A practical introduction -2nd, 2003, Springer-Verlag (London)
2. J Vesanto, SOM Toolbox, Laboratory of Computer and Information Science, http://www.cis.hut.fi/projects/somtoolbox/ (as of March 2011)
3. ME Harmon, Reinforcement Learning: A Tutorial, 1996, Wright State University
4. R Thomson, TCP/IP Java Socket Communication script http://www.mathworks.com/matlabcentral/fileexchange/21131-tcpip-socket-communications-in-matlab (as of March 2011)

Appendix A – Other concepts

Aside from the main concept which we worked out, two other concepts generated throughout our brainstorm sessions had good potential as well. In fact, it moreover was a hard decision picking the right one to hit for the development stage.

Learning doors

We all know the situation where you pass by a store when suddenly automatic sliding doors react on your movement, even though you did not attend to go inside. For these automatic sliding doors it is simply a stage of “being in range” to determine whether to activate the mechanism or not. It is moreover energy wasting, especially during times of winter. Not to think about the fact that it could be a potential stress and wear factor over time. A possible solution could be found in making the automatic sliding doors to learn certain walking patterns and directions to determine the chance of a person’s intention to enter the building. In this case the implementation of a learning algorithm would make a fair addition to the functioning of the automatic sliding doors. As we do not want people to face a closed door we can start out by saying that it would always open up if no clear determination can be made whether someone wants to enter or not. Over time the amount of samples (of people walking in front of the door) with various categorized combinations of walking routes and patterns will increase and contribute to the reliability of the system. To minimize the effect of an error, like when someone faces a closed door, the door will still always open when in really close range detection is sensed. While the person cannot progress directly (as it needs to wait for the doors to open up), he or she can still enter eventually. At the moment such an error occurred the system will be confronted with a heavy change to some of the neural weights. Over time, enough samples and errors will be detected to develop a reliable detection system. BELOW The higher the detection and possibility to recognize body posture and face detection the more reliable the system will become. With regard to the available resources and time it would indicate a similar approach as used in our “arrow detection system”, with the difference that it would generate its patterns based on the walking cycle and path of a person walking in range to detect it’s possibility of entering a store.



Meeting disturbance alert

Another concept was based on the fact that meetings with large amounts of people might unconscious end up in small little groups of attendees chatting to each other and becoming unfocused on the intended topics. In most cases one supervisor is asked to make sure such occurrences are avoided in order to retain focus amongst the members. These situations are often paired with groups of people trying to over sound each other, creating a rather disturbing meeting environment. The solution is a system that monitors and prevents such events. It would be able to detect certain patterns (combination of sound, body gesture, attention, etc.) that have a high potential of turning into a previous stated situations and alerts the attendees prior to the break point.

Appendix B - Implemented learning algorithms

Matlab code: server performing image recognition with Neural Gas

The code below is the main Matlab code to make a webcam snapshot, analyze it and send a result over a TCP/IP connection. It is described in the 'Image-capturing and recognition' chapter.

Img_acq_avg.m

```
% This routine captures frequent images from a connected webcam, after
% which these are analysed by a neural gas algorithm that is trained to
% recognize arrows, pointing at four directions. A TCP/IP server
% continuously broadcasts values that correspond to the direction of the
% recognized arrow.
%
% This routine is written by Michael Geertshuis, Martijn Kors and Rik Vegt
% for the module Learning Robots as part of the Master programme of the
% Department of Industrial Design
% Technical University of Eindhoven,
% The Netherlands, March 2011
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

load TrainingSet;        % load the trainingset
server_on = true;        % boolean to use broadcasting functionality

pause on;                % switch on pause mode for later use
vid = videoinput('winvideo', 1, 'RGB24_176x144'); % initialize external cam
%vid = videoinput('winvideo', 2, 'YUY2_176x144'); % initialize internal cam
src = getselectedsource(vid);

if (server_on)           % initialize TCP/IP server if boolean = true
    [d_output_stream, server_socket, output_socket] = serverinitialize(3000)
end

vidRes = get(vid, 'VideoResolution');  % set vid instance resolution and
nBands = get(vid, 'NumberOfBands');    % colors
vidImage = image( zeros(vidRes(2), vidRes(1), nBands) );
```

```
preview(vid), vidImage);   % draw video in vidImage window: don't remove!
vid.ReturnedColorSpace = 'grayscale';  % set the visualized colorspace

for i=1:600
    for ii=1:5                          % analyse 5 snapshots
        snapshot = getsnapshot(vid);    % save camera snapshot to variable
        img_in = img_proc(snapshot,60);% call img_proc function for
processing
        TestSet_class = neural_gas_class(TrainingSet_Clusters,img_in);
        % Display the frame in a figure window.
        figure(2); imagesc(snapshot);  % draw the UNPROCESSED snapshot
        TestSet_classes(ii) = TestSet_class; % fill list of 5 results
    end

    temp = sort(TestSet_classes);
    TestSet_class = temp(3);            % get median value of 5 results
    switch TestSet_class                % print a direction to screen instead
of nr.
        case 1
            direction = char('nowhere');
        case 2
            direction = char('right');
        case 3
            direction = char('down');
        case 4
            direction = char('left');
        case 5
            direction = char('up');
        otherwise
            direction = char('NOS');    % impossible with only 5 categories
    end
    fprintf(1, 'Pointing %s\n',direction);  % print to screen
    if server_on
        d_output_stream.writeBytes(char(message));  % send values to
outputbuffer
        d_output_stream.flush;                      % cleaning up buffer
    end
    pause(0.1);                                     % lower framerate
end


stoppreview(vid);                                   % stop asking for image
if (server_on) server_socket.close; output_socket.close; end
fprintf('Alles is over\n')
```

Processing code: Q-learning algorithm

Comments are in Blue

Code that is not used for Qlearning is Red

```
import processing.serial.*;
import nl.tue.id.creapro.admoveo.*;
import processing.net.*;

AdMoVeo admoveo;

int MOTOR = 200;
int PORT = 5;

Client c;

int input = 0;
int check = 0;

float LEARNING_RATE = 0.3;
float DISCOUNT = 0.1;

float frontDistanceNew;
float leftDistanceNew;
float rightDistanceNew;
float frontDistanceOld;
float leftDistanceOld;
float rightDistanceOld;

int dRotation = 0;
int oldStateDim1 = 1;
int oldStateDim2 = 0;
int newStateDim1 = 2;
int newStateDim2 = 0;
int oldAction = 1;
int newAction = 0;

float reward = 0;

int nStatesDim1 = 19; // the number of proximity states you have
int nStatesDim2 = 15; // the number of rotation states you have
int nActionsDim3 = 4; // the number of possible actions

float[][][] Q; //Three dimensional array containing the different state/action combinations and their reward values

void setup() {

 frameRate(1);

 admoveo = new AdMoVeo(this,"COM" + PORT);
```

```
 admoveo.getFrontDistanceSensor().enable();
 admoveo.getLeftDistanceSensor().enable();
 admoveo.getRightDistanceSensor().enable();

 admoveo.getLeftMotor().on();
 admoveo.getRightMotor().on();

 c = new Client(this, "127.0.0.1", 3000); // Connect to the server's IP address and port

 Q = new float[nStatesDim1][nStatesDim2][nActionsDim3]; //Initialize all Q values with a random value

  for(int i = 0; i<nStatesDim1; i++){
  for(int j = 0; j<nStatesDim2; j++){
    for(int k = 0; k<nActionsDim3; k++){
      Q[i][j][k] = random(0,100)/1000;
    }
  }
  }
}

 void draw() {
   admoveo.getLeftMotor().setPower(0);
   admoveo.getRightMotor().setPower(0);
   delay(200);

  incoming(); //Get input from Matlab

  if(dRotation == 0 && newStateDim1 >= 16 || newStateDim1 == 0){ //Switch to Neural Gas if there is no input on
  the distance sensors
     if(input != check){
     input = check;
     whattodo(input);
     dRotation = 0; //Set the rotation to zero
    }
  }
  else{ //If there is an input on the distance sensors (object within range) switch to Qlearning
  admoveo.getLeftMotor().setPower(0);
  admoveo.getRightMotor().setPower(0);
  calculateAction(); //Find the best possible action
  doAction();
 }
}

 void incoming(){
   c.clear();
   delay(200);
     if (c.available() > 0) {
     check = c.read();
  }
 }
 //Read input from AdMoVeo sensors and assign the values to the right variables
void inputAvailable(Sensor sensor, int oldValue, int newValue){
  if(sensor == admoveo.getFrontDistanceSensor()){
```

```
    admoveo.execute("getFrontDS",AdMoVeo.NOW);
  }
  else if(sensor == admoveo.getLeftDistanceSensor()){
    admoveo.execute("getLeftDS",AdMoVeo.NOW);
  }
  else if(sensor == admoveo.getRightDistanceSensor()){
    admoveo.execute("getRightDS",AdMoVeo.NOW);
  }
}

void getFrontDS(SensorStatus s){
  frontDistanceNew = s.get(admoveo.getFrontDistanceSensor());
  frontDistanceNew = map(frontDistanceNew, 0, 1023, 0, 6);
}

void getLeftDS(SensorStatus s){
  leftDistanceNew = s.get(admoveo.getLeftDistanceSensor());
  leftDistanceNew = map(leftDistanceNew, 0, 1023, 0, 6);
}

void getRightDS(SensorStatus s){
  rightDistanceNew = s.get(admoveo.getRightDistanceSensor());
  rightDistanceNew = map(rightDistanceNew, 0, 1023, 0, 6);
}

void whattodo(int a){
  switch(a){
    case 0:
    noimage();
    break;
    case 1:
    goforward();
    break;
    case 2:
    gobackward();
    break;
    case 3:
    goleft();
    break;
    case 4:
    goright();
    break;
    case 5:
    uturn();
    break;
  }
}

void getReward(){
    reward = 0.0;

    //goal: Outgoing direction - Incoming direction = 0;
    //      Evade collision (NewDistance >= OldDistance)
```

```
   //     Turning direction * Turning speed * time

   reward += (float(newStateDim1) - float(oldStateDim1)) / 10;  //Reward for the distance to objects
   reward += (1 - float(dRotation)) / 10; //Reward for the rotation from the origin
}

void getNewValues(){ //Convert the sensor values to usable values
  newStateDim1 = floor(frontDistanceNew + leftDistanceNew + rightDistanceNew);
  newStateDim2 = abs(floor(dRotation));
}

void calculateAction(){ //Calculates the best action from a given point in the Q value grid
   getNewValues();
   getReward();
     newAction = getMaxAction(newStateDim1,newStateDim2); // get the action with the highest Q value
     // the formula that does the learning and updates the Q value of the previous state
     Q[oldStateDim1][oldStateDim2][oldAction] += LEARNING_RATE * ((reward + DISCOUNT) *
(Q[newStateDim1][newStateDim2][newAction] - Q[oldStateDim1][oldStateDim2][oldAction]));
   // put the new states in the old states
   oldStateDim1 = newStateDim1;
   oldStateDim2 = newStateDim2;
   oldAction = newAction;

}

int getMaxAction(int s1, int s2){ //Look into the Q value table and find the action with the highest value
   int action = 0;
     for(int x=0; x < nActionsDim3; x++){
      if(Q[s1][s2][x] > Q[s1][s2][action])
        action = x;
       }
      return action;
}


void doAction(){ //Perform the action
  switch(newAction){
       case 0:
       rotateLeft();
   break;
       case 1:
       rotateRight();
   break;
       case 2:
       goForward();
   break;
       case 3:
       goBackward();
   break;
  }

}
//Actions:
```

```
//Turn right
//Turn left
//Drive
//Turn to dDir = 0

//States:
//dDirNew < dDirOld (closer to original)
//dDirNew > dDirOld (further from original)
//dDirNew = dDirOld (going straight)

//newDistance < oldDistance (closer to object)
//newDistance > oldDistance (further from object)
//newDistance = oldDistance (parallel to object)

void goforward(){
  admoveo.getLeftMotor().forward();
  admoveo.getRightMotor().forward();
  admoveo.getLeftMotor().setPower(150);
  admoveo.getRightMotor().setPower(150);
     delay(150);
 }

 void gobackward(){
  admoveo.getLeftMotor().backward();
  admoveo.getRightMotor().backward();
  admoveo.getLeftMotor().setPower(200);
  admoveo.getRightMotor().setPower(200);
  delay(150);
 }

 void goleft(){
  admoveo.getLeftMotor().forward();
  admoveo.getRightMotor().forward();
  admoveo.getLeftMotor().setPower(100);
  admoveo.getRightMotor().setPower(200);
  delay(150);
 }

 void goright(){
  admoveo.getLeftMotor().forward();
  admoveo.getRightMotor().forward();
  admoveo.getLeftMotor().setPower(200);
  admoveo.getRightMotor().setPower(100);
  delay(150);
 }

 void noimage(){
  admoveo.getLeftMotor().forward();
  admoveo.getRightMotor().forward();
  admoveo.getLeftMotor().off();
  admoveo.getRightMotor().off();
  delay(150);
 }
```

```
 void uturn(){
   admoveo.getLeftMotor().forward();
   admoveo.getRightMotor().backward();
   admoveo.getLeftMotor().setPower(250);
   admoveo.getRightMotor().setPower(250);
   delay(150);
}

//Qlearning actions

void rotateLeft(){
   admoveo.getLeftMotor().backward();
   admoveo.getRightMotor().forward();
   admoveo.getLeftMotor().setPower(150);
   admoveo.getRightMotor().setPower(150);
   dRotation -= 1;
   delay(150);
}

void rotateRight(){
   admoveo.getLeftMotor().forward();
   admoveo.getRightMotor().backward();
   admoveo.getLeftMotor().setPower(150);
   admoveo.getRightMotor().setPower(150);
   dRotation += 1;
   delay(150);
}

void goForward(){
   admoveo.getLeftMotor().forward();
   admoveo.getRightMotor().forward();
   admoveo.getLeftMotor().setPower(150);
   admoveo.getRightMotor().setPower(150);
   delay(150);
}

void rotateOrigin(){

}

void goBackward(){
   admoveo.getLeftMotor().backward();
   admoveo.getRightMotor().backward();
   admoveo.getLeftMotor().setPower(150);
   admoveo.getRightMotor().setPower(150);
   delay(150);

}
```

Appendix C – Supporting code

Matlab code: Image Preprocessing code

The code below is the code that is described in the 'Preprocessing code' chapter.

Img_proc.m

```
function [img_out] = img_proc (img_in,threshold)


target_res_x = 100;
target_res_y = 100;

img_bw=img_in;

y = length(img_bw(:,1));
x = length(img_bw(1,:));

for iy=1:y
    for ix=1:x
        if img_bw(iy,ix) <= threshold;
            img_bw_th(iy,ix) = 1;
        else
            img_bw_th(iy,ix) = 0;
        end
    end
end

inew = 0;

for ix=1:x
    if sum(img_bw_th(:,ix)) ~= (y*0)
        inew = inew+1;
        img_bw_th_cut(:,inew) = img_bw_th(:,ix);
    end
end

if ~exist('img_bw_th_cut')
    img_bw_th_cut2 = uint8(zeros(target_res_x,target_res_y));
else
    inew = 0;
    xnew = length(img_bw_th_cut(1,:));
    for iy=1:y
        if sum(img_bw_th_cut(iy,:)) ~= (xnew*0)
            inew = inew+1;
            img_bw_th_cut2(inew,:) = img_bw_th_cut(iy,:);
        end
    end
end

ynew = length(img_bw_th_cut2(:,1));
```

```
%     x_ratio = ynew/target_res_x;
%     y_ratio = xnew/target_res_y;
%
img_bw_th_cut2 =
imresize(img_bw_th_cut2,[target_res_x,target_res_y],'nearest');
xnew = target_res_x;
ynew = target_res_y;

for ix=1:xnew
    img_out(ix) = sum(img_bw_th_cut2(:,ix));
end

for iy=1:ynew
    img_out(ix+iy) = sum(img_bw_th_cut2(iy,:));
end

clear iy ix inew xnew ynew img_bw img_bw_th img_bw_th_cut img_bw_th_cut2
```

Matlab code: server initialization

The code below is used to setup a TCP/IP server.

Serverinitialize.m

```
%MATLAB (server initialization part) TCP/IP server connection setup for
Module: Learning Robots 2011.

% Code is based on Rodney Thomson's TCP/IP Java Socket Communication script
(http://www.mathworks.com/matlabcentral/fileexchange/21131-tcpip-socket-
communications-in-matlab)
% In comparison to the MATLAB provided TCP/IP object this script is able to
allow more than 1 clients at a time per socket as well as that for this
module this implementation had a higher success rate.

% from the main application this function has to be called once in order to
initialize a server connection between MATLAB and Ppocessing
% The data_output_stream is a stream of data that is being send from the
server to Processing (or if requested: the other way around. Note that it can
work in a bidirectional way).
% The server_socket open a server socket and waits untill a request comes in,
once initialized it can accept connections.
% The output_socket replaces the open server_socket as soon as a connection
has been established.
% The serverinitialize(port) is the called function from the main application
and includes the possibility to change the TCP/IP port.
function[data_output_stream, server_socket, output_socket] =
serverinitialize(port)

      %Import the Serversocket object from the Java framework in MATLAB to
      set up a TCP/IP connection.
    import java.net.ServerSocket
    import java.io.*

      %create arrays for the server_socket and the output_socket to store
      data which is buffered
    server_socket  = [];
    data_output_socket  = [];

    while true

        try

            %Here we are setting up the server socket and bind it to the port
            we specified in the function call.
           server_socket = ServerSocket(port);
           server_socket.setSoTimeout(1000);

            %Notifies the user that the server is ready to accept TCP/IP
            connections on local IP as it is the server running this script
           fprintf(1, 'Opening Connection.....accepting clients\n');
```

```
            %While the server socket is now bound to a specific socket, the
            socket does not yet listen to incomming connection calls. We need
            to tell the ServerSocket to accept incomming calls.
           data_output_socket = server_socket.accept;

            %Once an incomming client conenction has been detected we notify
            the user that a connection has been established, the server is
            now ready to send and receive data.
           fprintf(1, 'The server accepted the request to connect to the
client \n');

            %Sets up the steam on which the buffered data can be send.
           output_stream   = data_output_socket.getOutputStream;
           data_output_stream = DataOutputStream(output_stream);

           break;
        end
    end
end
```

Serverlaunch.m

```
%MATLAB (server call part) TCP/IP server connection setup for Module:
Learning Robots 2011.
clc
clear all
%This script can be implemented into the main code where an incoming server
connection is required
% from the main application this function has to be called once in order to
initialize a server connection between MATLAB and Processing
% The data_output_stream is a stream of data that is being send from the
server to Processing (or if requested: the other way around. Note that it can
work in a bidirectional way).
% The server_socket open a server socket and waits until a request comes in,
once initialized it can accept connections.
% The output_socket replaces the open server_socket as soon as a connection
has been established.
% The serverinitialize(port) is the called function from the main application
and includes the possibility to change the TCP/IP port (in this case  port
3000).

%calling the server initializes function in serverinitialize.m
[data_output_stream, server_socket, output_socket] = serverinitialize(3000);

% Lets the user know that DataMessage has been found (what is contains) and
that it can be send.
fprintf(1, 'Writing number %s, how awesome \n', DataMessage)

% Calls the data_output_stream connenction and send "DataMessage" over the
connection stream.
data_output_stream.writeBytes(char(DataMessage));
data_output_stream.flush;
pause(1);

% cleaning up
server_socket.close;
output_socket.close;
```

Processing code: client controlling AdMoVeo

This is the commented code for the client side of the network in processing language. Incoming signal from the analysis of the webcam-feed is translated here to actions on the robot.

```
import processing.serial.*;
import nl.tue.id.creapro.admoveo.*;
import processing.net.*;

Client c;

int input;
char oldInput;

AdMoVeo admoveo;

int MOTOR = 150; //Motor speed setting
int PORT = 5; //AdMoVeo port number

//Initialize the AdMoVeo, client and motors
void setup() {
  frameRate(1);

  admoveo = new AdMoVeo(this,"COM" + PORT); //Initialize AdMoVeo
  c = new Client(this, "131.155.175.128", 3000); //Initialize client on servers IP & port number

  admoveo.getLeftMotor().on();
  admoveo.getRightMotor().on();
}

//Constantly listen for signal from the server and perform an action when the input signal changes
void draw(){
   admoveo.getLeftMotor().setPower(0);
   admoveo.getRightMotor().setPower(0);
   delay(100);
     if (c.available() > 0) {
       oldInput = c.readChar(); //Read from the server
         if(input != oldInput){
           input = oldInput - 48; //Translate incoming char into an int
           doAction(input); //Initialize action
         }
       c.clear(); //clear the buffer
     }
}

//Decides what action to perform depending on the input variable and execute the action
void doAction(int a){
  switch(a){
      case 0:
        noimage();
      break;
      case 1:
        goright();
```

```
      break;
      case 2:
        gobackward();
      break;
      case 3:
        goleft();
      break;
      case 4:
        goforward();
      break;
  }
}

//List of actions that can be called by the doAction() function

void goforward(){
   admoveo.getLeftMotor().forward();
   admoveo.getRightMotor().forward();
   admoveo.getLeftMotor().setPower(MOTOR);
   admoveo.getRightMotor().setPower(MOTOR);
   delay(150);
}

void gobackward(){
   admoveo.getLeftMotor().backward();
   admoveo.getRightMotor().backward();
   admoveo.getLeftMotor().setPower(MOTOR);
   admoveo.getRightMotor().setPower(MOTOR);
   delay(150);
}

void goleft(){
   admoveo.getLeftMotor().forward();
   admoveo.getRightMotor().forward();
   admoveo.getLeftMotor().setPower(MOTOR);
   admoveo.getRightMotor().setPower(MOTOR);
   delay(150);
}

void goright(){
   admoveo.getLeftMotor().forward();
   admoveo.getRightMotor().forward();
   admoveo.getLeftMotor().setPower(MOTOR);
   admoveo.getRightMotor().setPower(MOTOR);
   delay(150);
}

void noimage(){
   admoveo.getLeftMotor().forward();
   admoveo.getRightMotor().forward();
   admoveo.getLeftMotor().setPower(0);
   admoveo.getRightMotor().setPower(0);
   delay(150);
}
```