MEC6509 Analyse des systèmes de production flexible

Simulation of task assignment to Automated Guided Vehicles with a distributed intelligence approach

Class project

By
Stefan Bracher

Presented to
El-Kébir Boukas

17. April 2007

École Polytechnique de Montréal

Index

1. Introduction

In many factories Automated Guided Vehicles (AGV) are used to transport goods between different stations. There are different ways to solve the task assignment problem [2], [1]. One possibility is to use a distributed intelligence approach, where an intelligent behaviour emerges from the interaction of the stations and AGV through the interaction by virtual pheromones.

In this work, a simulator is developed in order to prove that the distributed intelligence approach is functional and to study the influence of different factors, such as number of machines, number of AGVs and the shop layout on the system performance.

2. Swarm behaviour and stigmergy

The distributed intelligence approach is based on two fundamental concepts, Swarm behaviour and Stigmergy, which are biology inspired. The first can be defined as “the cooperative behaviour of a group of many individuals”, while Stigmergy is "the influence on behaviour of the persisting environmental effects of previous behaviour" [7]. Both can be seen in ant colonies, where the collaboration of relatively simple individuals produces amazing results. For example, the ants build “trails” which connect food sources to the nest by emitting a pheromone trace.

Figure 1: An ant transporting a leave to the nest on an ant trail. Foto: Stefan Bracher

The concepts Swarming and Stigmergy have been successfully implemented to robotic systems by various researchers [5] [6] [4][...] and can be reduced to the formula:”Simple Agents following simple rules produce a complex behaviour”.

3. Implementation of swarm behaviour and stigmergy in the manufacturing environment

In the manufacturing environment, the swarm concept can be implemented easily. The “swarm” consists of the various machines, loading/unloading stations as well as the AGVs, and is thus a heterogeneous swarm, as the swarm members are not all identical.

More difficult is the adaptation of the stigmergy concept, as the different swarm agents have to communicate trough the environment. The use of chemical pheromones is not practical, as the, compared to ant colonies, big distances would require quite high concentrations that could harm the human workers. Using light or sound as transmitter, has the disadvantage that the signal can not bend around edges and obstacles like the chemical pheromone. And finally, a radio signal has the opposite problem, as it penetrates through walls.

The solution to this problem is the virtual pheromone: The behaviour of the chemical pheromone is simulated. One possibility to do this is to use a central unit computing a simulation on a map of the shop layout: The various agents send their position and status to the unit and receive the simulated pheromone concentration. Another possible way is to place relay stations throughout the shop, that sum the incoming signals and resend the sum in all directions. In this way also light and sound transmissions can “bend” around edges. Shen et al. [3] realized already an approach quite similar to that one by using the swarm robots themselves as relays stations.

The main advantage of the central unit solution is, that it is comparably cheap and able to simulate the real pheromone very accurate, while using relays is more reliable as changes on the layout, example workers walking around or other moving obstacles, are included in the simulation and neither a shop map nor any position information is needed. In fact, using relays is probably closer to the swarm concept as the central computation solution, as the relays themselves can be seen as just some more agents within the swarm.

In both ways, the virtual pheromone has to have the following features: Dispersion throughout the shop building a gradient, that leads two its source and dissipation over time so that the recently emitted signals are more important than the older.

4. The simulation program

4.1 The programming language

The software is programmed in MATLAB® . This, because MATLAB® offers many ready to use functions and an easy way to realize a graphical user interface while still executing at acceptable speed.

4.2 Simplifications and limitations

The program is made for the concept with central computation only.

For the simulation of the system, several simplifications have been made:

- The shop floor is discretized in fields of 1 square meter.
- Just one type of machines is used.
- A machine occupies 9 square meter and has a loading/unloading bay of one square meter.
- The AGVs are one square meter in size and can move in 4 directions with constant speed only.
- Loading bays produce the jobs to be done and occupy 9 square meters with a bay of one square meter.
- Unloading bays unload the jobs from the system and occupy 9 square meters with a bay of one square meter.
- Loading and Unloading of goods to and from AGVs occurs instantly.
- All elements are 100% reliable and tasks always take the same time to be executed.
- Just one job type is used.

Further on, the simulation software is programmed for research purposes only. So it is neither optimized for calculation speed nor is it failsafe against unforeseen user inputs. Also, in order to reduce development time, some minor bugs, that do not affect the overall goal are not corrected (For example: The “Stop” button has to be pressed several times in order to interrupt a ongoing simulation).

4.3 Software components

In the following, a short explanation about the major components of the software is given, for more details on each component, see the source code in the Annex.

4.3.1 Floor map

As already said, the shop is discretized in fields of one square meter. The floor map is represented by a two dimensional matrix where the element of the i-th column and j-th row is one, if there is an obstacle on the corresponding field and zero if there is no obstacle.

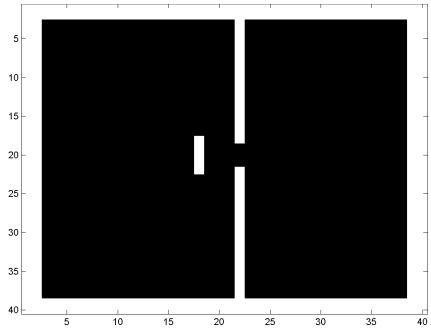

Figure 2: A graphical representation of a floor map, with obstacles being white and free space being black

4.3.2 Virtual pheromones and pheromone dissipation/dispersion

Three virtual pheromones are introduced: One pheromone that is emitted when there is a transportation job to be done, one pheromone that is emitted by idle machines and one that is emitted by idle unloading bays. For each pheromone a discretized map of the shop corresponding to the floor map is made, represented by a 2D matrix where the matrix elements correspond to the virtual pheromone concentration in the fields.

For the pheromone dissipation and dispersion the following algorithm is used in order to fulfil the requirements specified under point 3:

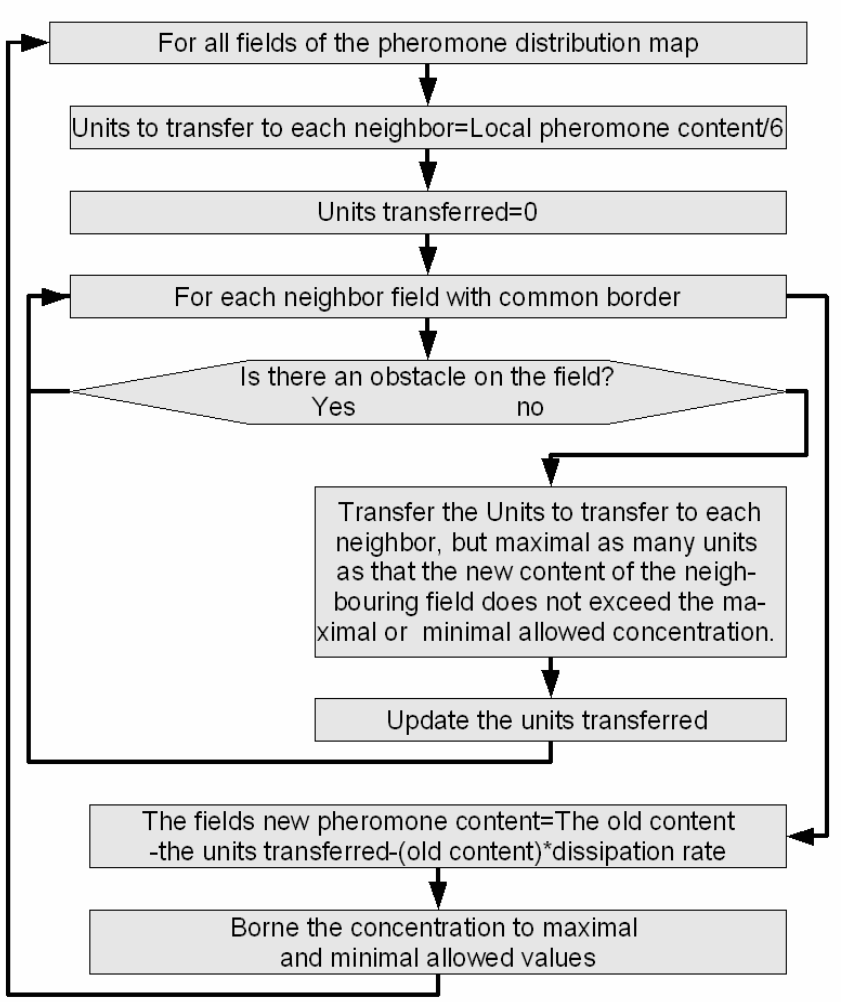

Figure 3: Pheromone dispersion and dissipation algorithm

If a pheromone source is placed at the centre of the floor map of figure 2. The following pheromone distributions are obtained by the algorithm using a low dispersion rate.

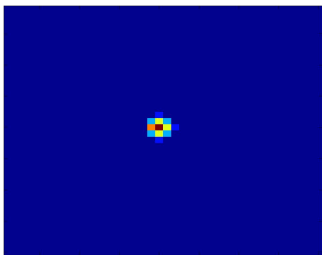

Figure 4: Pheromone distribution t=2s

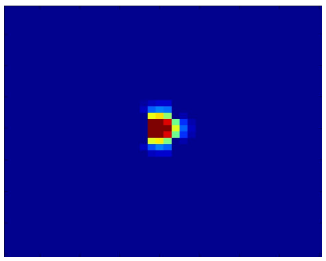

Figure 5: Pheromone distribution t=16s

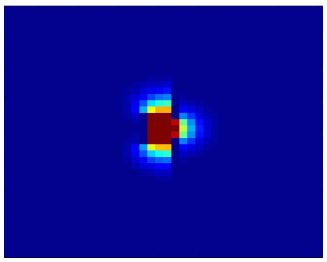

Figure 6: Pheromone distribution t=50s

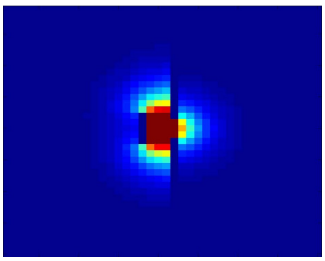

Figure 7:Pheromone distribution t=1000s

It can be seen how the pheromone dispersion is blocked by the walls and how the pheromone "bends" around the edges.

4.3.3 Loading bays

Loading bays occupy 9 square meters and have a bay of one square meter. It takes the bay a certain time to load a new job. Once a job is loaded, the pheromone indicating a transportation job to be done is increased in the loading bay. When an idle AGV is in the bay, the job is transmitted to the AGV and the loading bay starts to load a new job.

Loading bay rules:

- When idle, start loading a new job
- When job is prepared, start emitting the pheromone “transportation job to be done”
- When an empty AGV is in the bay and a job is prepared, transfer the job to the AGV and start loading a new job.

4.3.4 Unloading bays

The unloading bays are similar to the loading bays, just that they perform the opposite task: When idle, they emit a pheromone telling the AGVs that they are idle. Once an AGV with an completed job is in the bay, the job is transferred from the AGV to the unloading bay. The unloading bay now needs a certain time to unload the completed job. When finished it starts again to emit its idle pheromone.

Unloading bay rules:

- When idle, emit the pheromone “Unloading bay idle”.
- When an AGV with a completed job is in the bay, transfer the job and start unloading it.

4.3.5 Machines

Machines as well occupy an area of nine square meters. When idle, they emit the “machine idle” pheromone in their loading/unloading bay. Once a AGV with a new job is in the bay, the job is transferred to the machine. After the machine has completed the job, the pheromone “transport job to be done” in the bay is increased. This time, the machine waits for an idle AGV, to which, as soon as it is in the bay, the completed job is transferred. Now the machine is idle again.

Machine rules:

- When idle, emit the pheromone “Machine idle”.
- When an AGV with a new job is in the bay, transfer the job and start working on it.
- When the job is completed, emit the pheromone “Transportation job to be done”.
- When an empty AGV is in the bay and a completed job is waiting, transfer the job to the AGV.

4.3.6 AGVs

AGVs occupy an area of one square meter. They can move forward, backward and sideways, rotation is neglected. The AGV has three possible status: idle, loaded with a new job and loaded with a completed job. Depending on the internal status, the AGV follows the gradient of the corresponding pheromone, while avoiding any collision, to find to its destination. Further on the AGV is reducing the pheromone which it is currently following the gradient, in order to avoid getting too close to AGVs with the same status.

AGV rules:

- Avoid any obstacle
- Follow the gradient of the pheromone corresponding to the current status, if there is no gradient, do random walk
- Reduce the pheromone your following around you

4.4 The graphical user interface

A graphical user interface has been programmed in order to control the simulation. It's layout is the following:

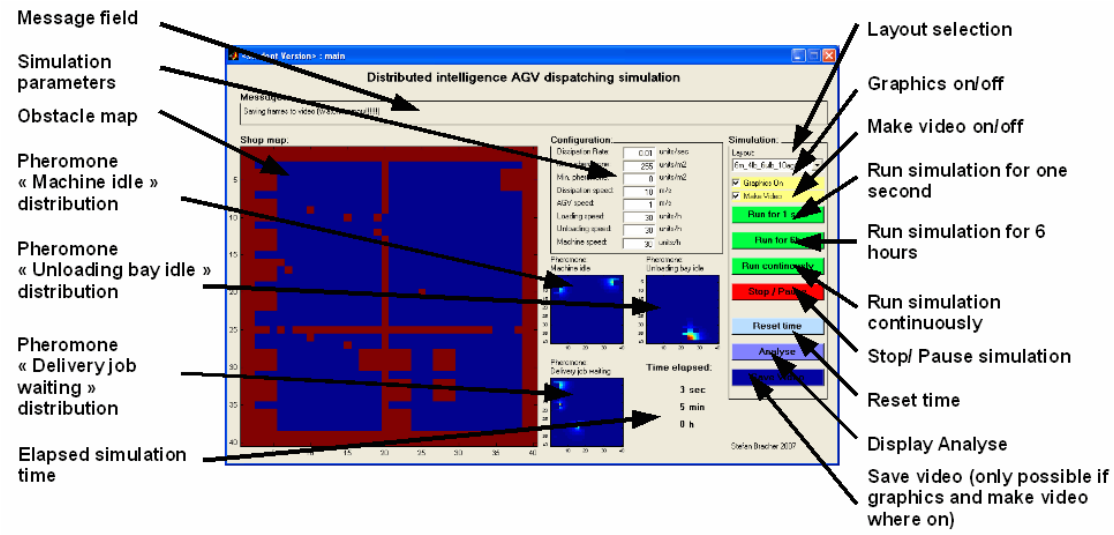

Figure 8: The main graphical user interface

Once the analyze button is pressed, another window displaying the analysis of the simulation run opens:

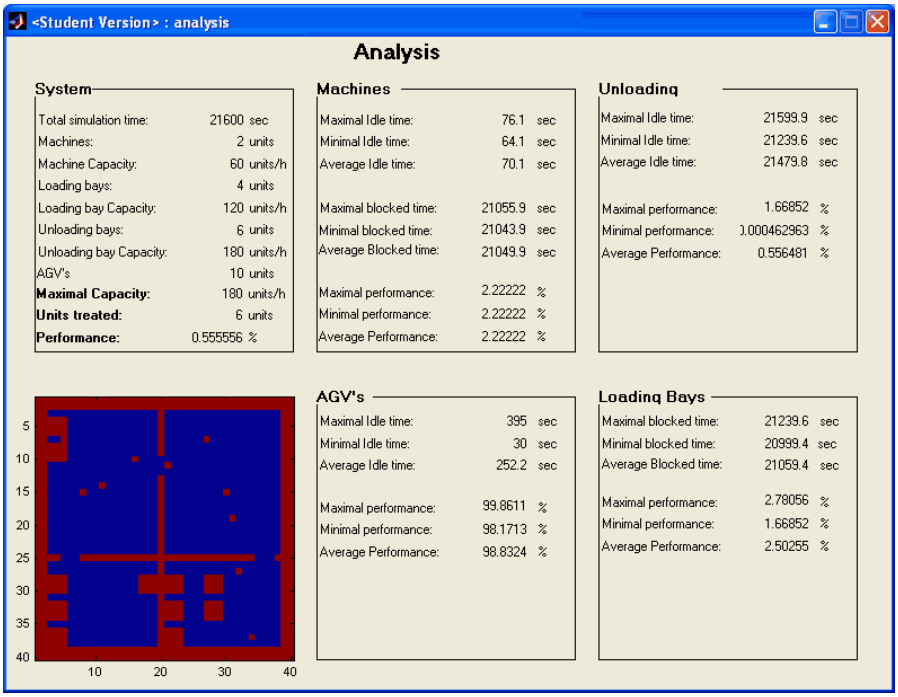

Figure 9: Analysis window

5. Experiments

5.1 Proof of concept

In a first test run, a simple floor map dividing the shop floor in four areas is loaded. One loading bay and one AGV are placed in the room on the bottom left, one machine each in the two top rooms, and two unloading bays in the room on the bottom right.

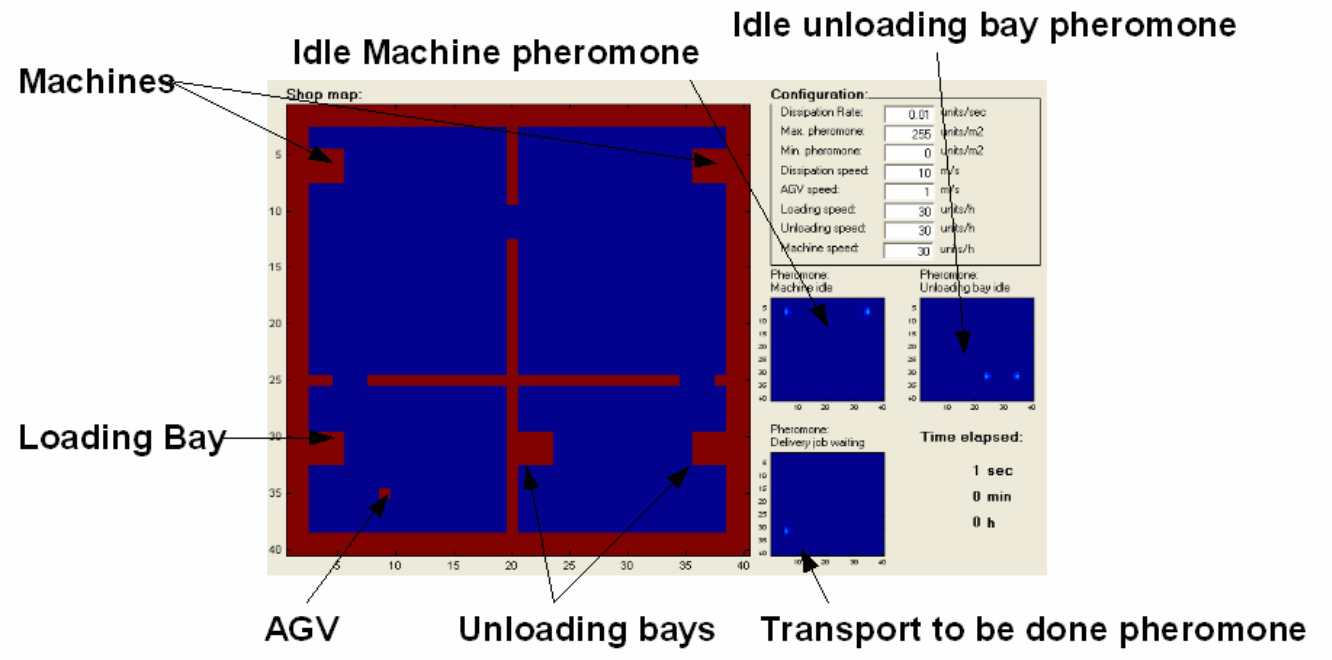

Figure 10: The first layout to be tested

The AGV starts directly to move toward the loading bay, which has a new job to deliver to the machines. It reaches the machines loading bay after 8s.

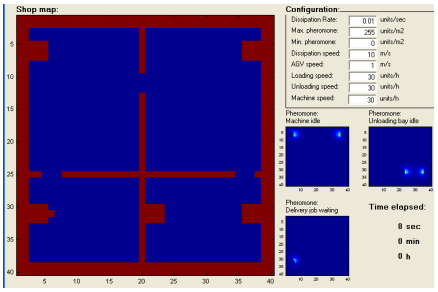

Figure 11: The simulation after 8s

After 31 seconds the AGV almost reached a machine. The distributed intelligence approach has automatically chosen the machine that was closer.

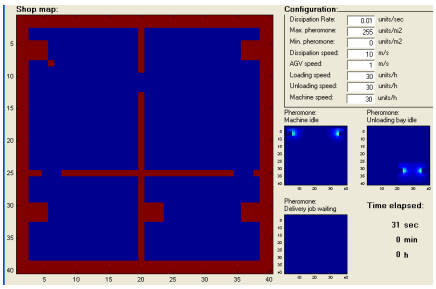

Figure 12: The simulation after 31s

And just seconds later, another intelligent behaviour can be observed: Although the loading bay has not yet loaded another job, the AGV is going close to it, as there is still some pheromone remaining from the last time the bay had a job to deliver. Unlike the first intelligent behaviour, this one was not foreseen.

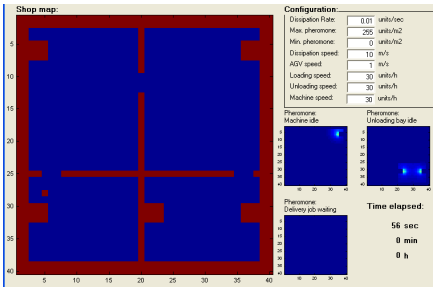

Figure 13: The simulation after 56s

As the closest machine is currently occupied, the AGV brings the second job to the machine that is further away.

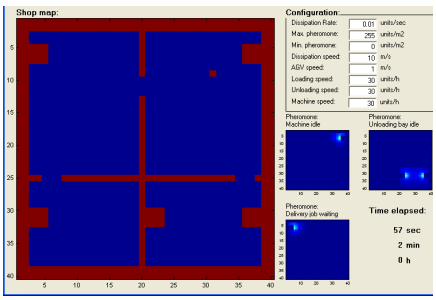

Figure 14: The simulation after 2 min 57s

In the meanwhile, the first machine had finished its job and the AGV is transporting the completed job now to the closest unloading bay.

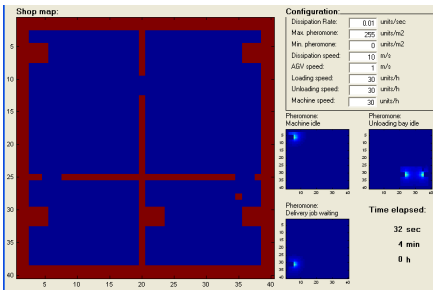

Figure 15: The simulation after 4min 32s

The system seems to work and has proved the formula ”Simple Agents following simple rules produce a complex behaviour”, even with behaviour that was not foreseen.

Now the simulator can be used to experimentally determine the influence of different parameters on the overall system performance.

5.2 Dependence on machine number

To determine the dependence on the machine number, the number of loading bays, unloading bays, AGVs as well as the floor layout are kept equal. The loading bays are in the room on the bottom left, the unloading bays in the bottom right and the machines in the two top rooms.

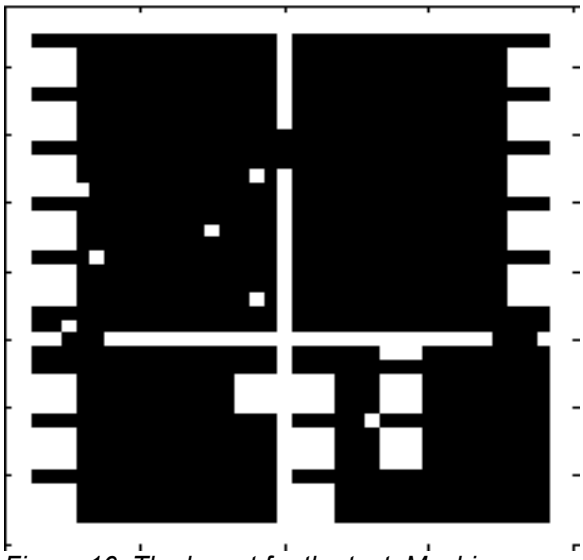

Figure 16: The layout for the test: Machines are placed in the upper rooms, Loading bays at the bottom left and Unloading bays at the bottom right.

The system parameters are:

	Number	Total capacity
Loading bays	4	120 units/h
Machines	1...10	30-300 units/h
Unloading bays	6	180 units
AGVs	10	-

The result:

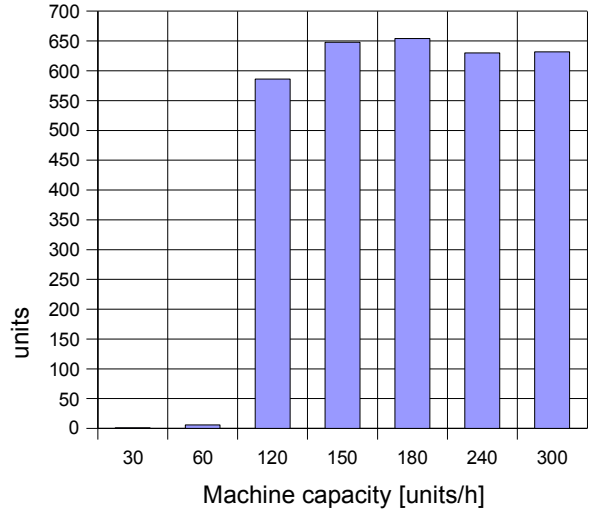

Figure 17: The production

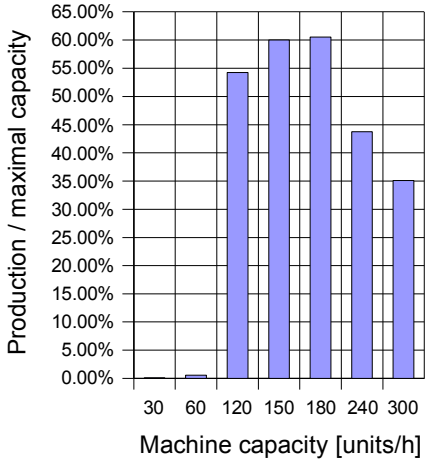

Figure 18: The system performance. Calculated as the production divided by the capacity of the system component with the highest capacity

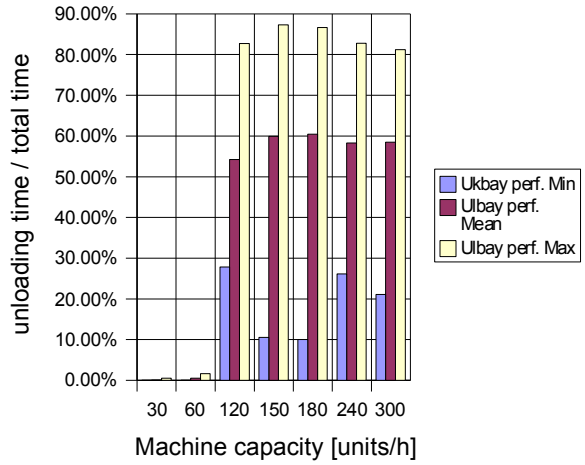

Figure 19: The unloading bay performance.

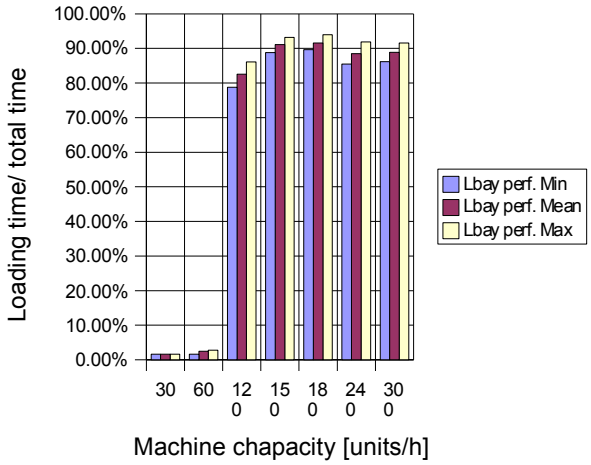

Figure 20: The loading bay performance

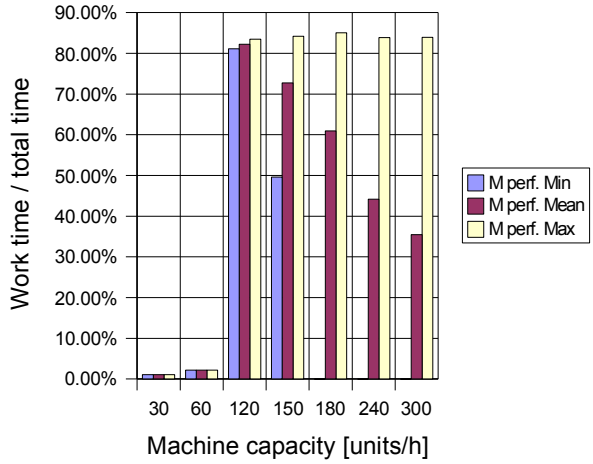

Figure 21: The machine performance

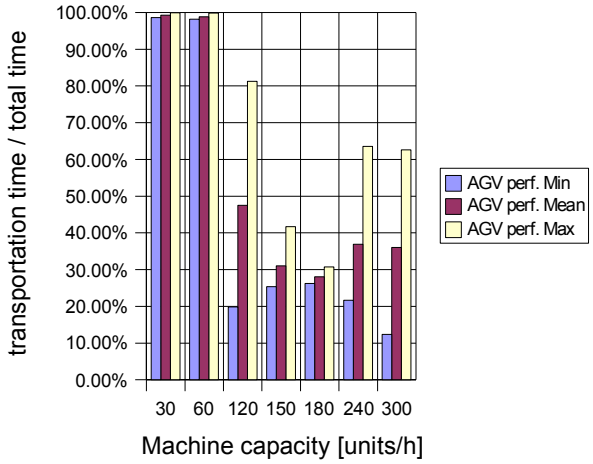

Figure 22: The AGV performance

For a total machine capacity smaller than the loading bay capacity, the system production and performance is very poor. What happens is that the loading bays produce the jobs faster than the machine can proceed them. After a while all the AGVs have loaded a new job that they try to deliver to the machines. But with no idle AGV left, a completed job can not be removed from the machines, thus the system becomes blocked.

It is known that for a system to be stable, the capacity must be higher than the input. But the charts show that already for a machine capacity equal the loading bay capacity, the system is stable. This is probably due to the layout, as any AGV coming back from an unloading bay must pass the machines and thus the jobs created by the machines have priority. In this way the system blockage as described before does not occur. In fact, the loading bays are not able to load jobs at full capacity as they have to wait for AGVs. Calculating the actual input

flow as the loading capacity times the average loading bay performance gives a lower loading rate and thus the system stability criteria is fulfilled.

Another observation is that the maximal machine performance is reached for the total machine capacity equal the total loading bay capacity. Adding more machines still increases the overall production but an upper limit is reached very fast. Once this upper limit is reached (180 units/ h and more), additional machines are idle all the time because the job flow is now limited by other parameters such as the loading bay and the AGV capacity.

5.3 Dependence on loading bay number

Using the same layout, this time the number of loading bays is changed.

The system parameters are:

	Number	Total capacity
Loading bays	1..12	30-360 units/h
Machines	5	150 units/h
Unloading bays	6	180 units
AGVs	10	-

The result:

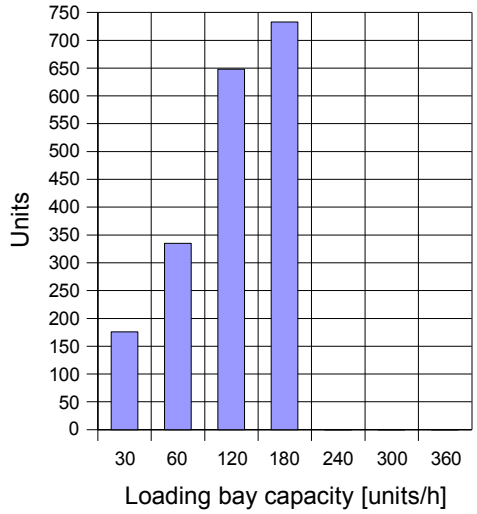

Figure 23: The production.

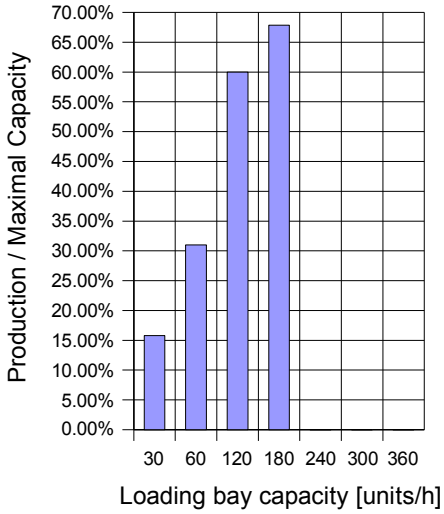

Figure 24: The system performance.

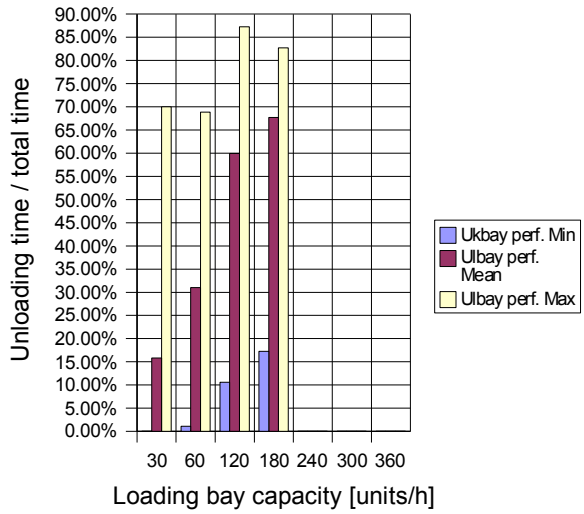

Figure 25: The unloading bay performance.

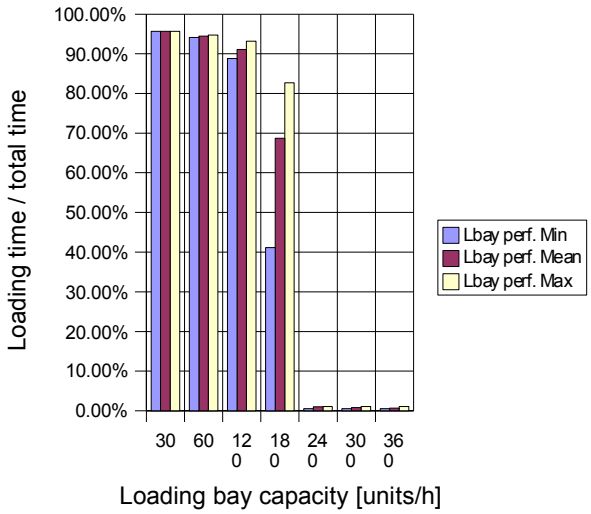

Figure 26: The loading bay performance

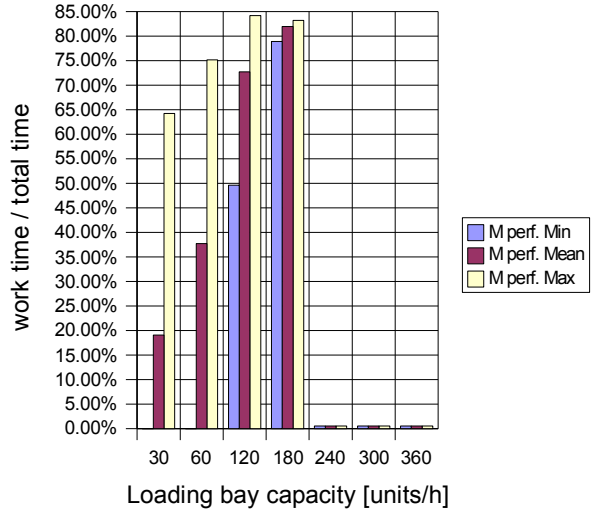

Figure 27: The machine performance

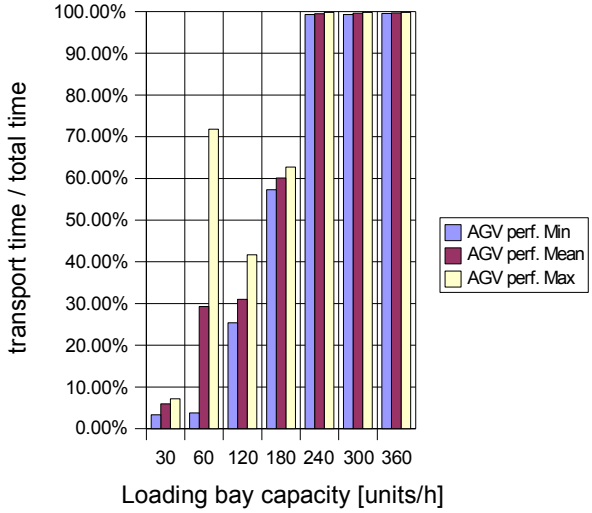

Figure 28: The AGV performance

The production of 6h running time is increasing with the number of loading bays until the total input rate (the loading bay performance*total loading capacity) is higher than the total machine capacity, when again all vehicles are loaded with new jobs and the system breaks down.

5.4 Dependence on unloading bay number

Using the same layout, the number of unloading bays is changed.

The system parameters are:

	Number	Total capacity
Loading bay	4	120 units/h
Machines	5	150 units/h
Unloading bays	1..12	30-360 units/h
AGVs	10	-

The result:

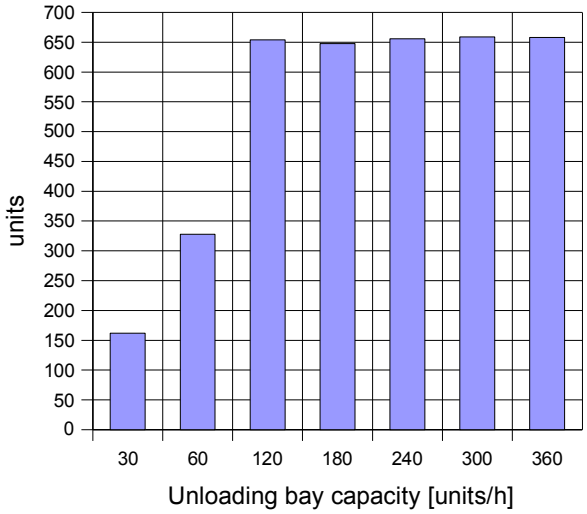

Figure 29: The production

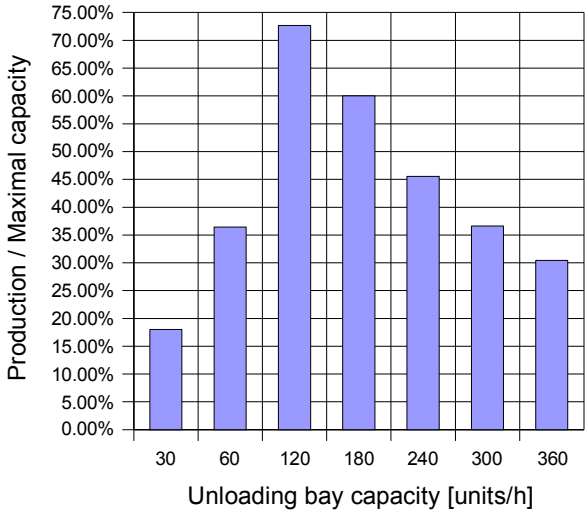

Figure 30: The system performance

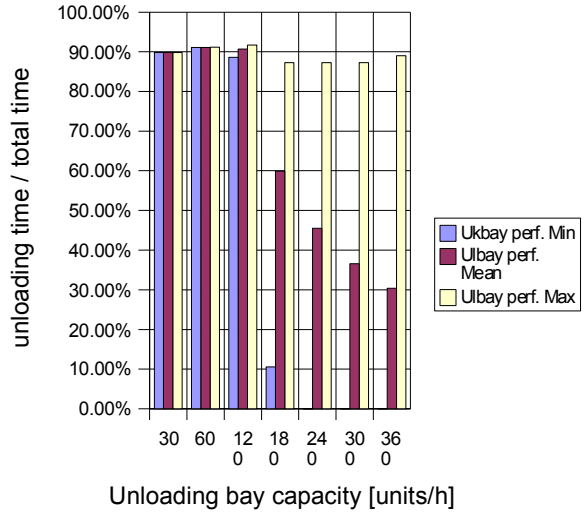

Figure 31: The unloading bay performance.

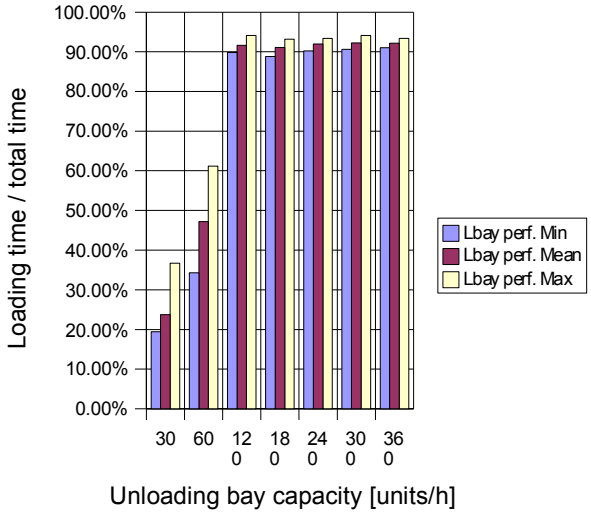

Figure 32: The loading bay performance

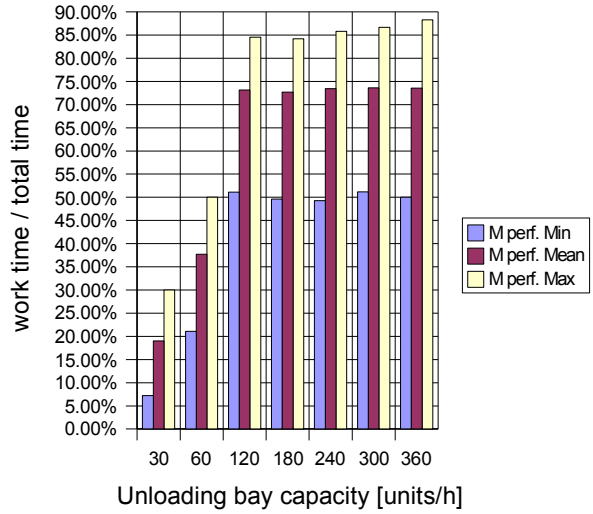

Figure 33: The machine performance

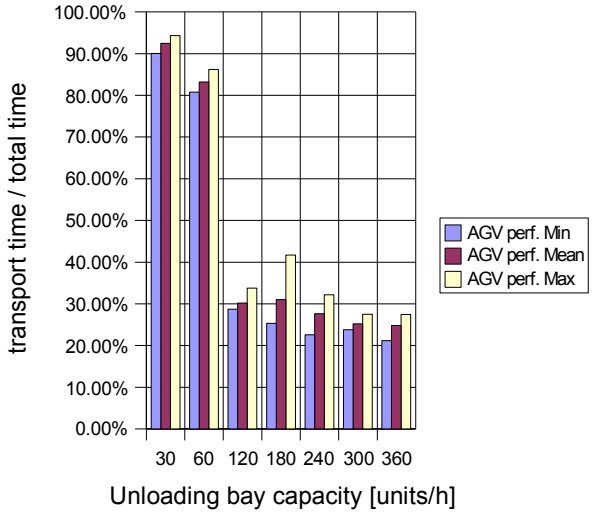

Figure 34: The AGV performance

Unlike before, a total unloading bay capacity lower than the total machine capacity and the loading bay capacity does not cause the system to fail. The AGVs are in this cases almost all the time loaded, but as the unloading stations can never get blocked, instead of system instability a self regulation is emerging. The machine performance is adjusted in a way that the machine performance times the machine capacity is lower as the unloading bay capacity and thus the stability criteria is met.

As soon as the total unloading bay capacity is higher than the loading bay capacity, the addition of further unloading bays is absolutely useless, as being at the end of the chain, it can not increase the overall flux.

5.5 Dependence on AGV number

Using the same layout at a stable configuration, the number of AGVs is changed

The system parameters are:

	Number	Total capacity
Loading bays	4	120 units/h
Machines	5	150 units/h
Unloading bays	6	180 units
AGVs	1..20	-

The result:

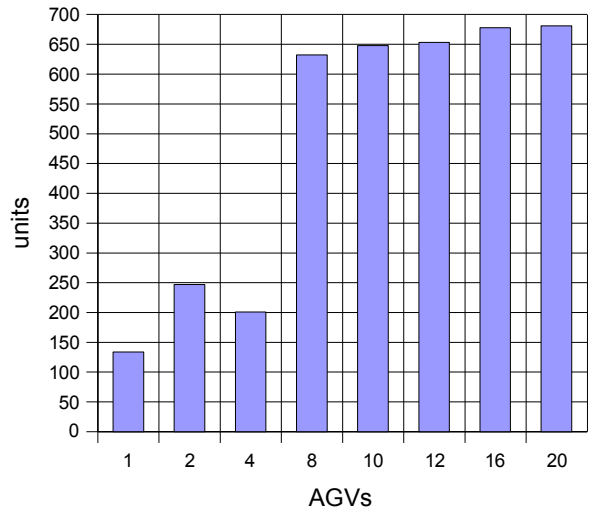

Figure 35: The production

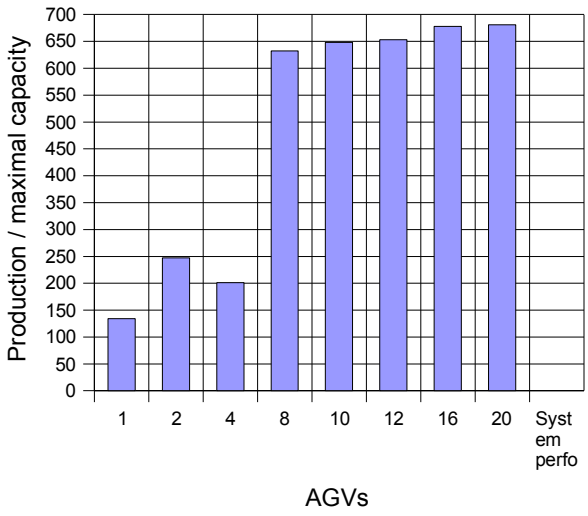

Figure 36: The system performance

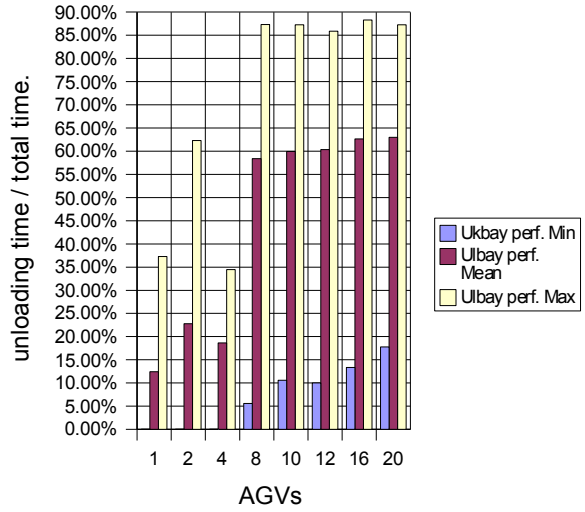

Figure 37: The unloading bay performance.

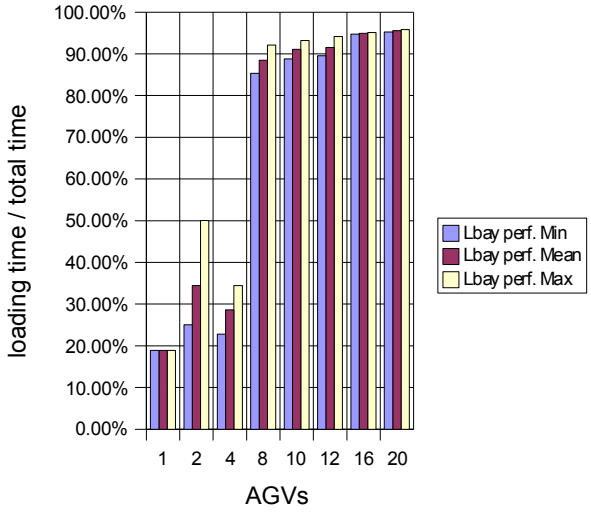

Figure 38: The loading bay performance

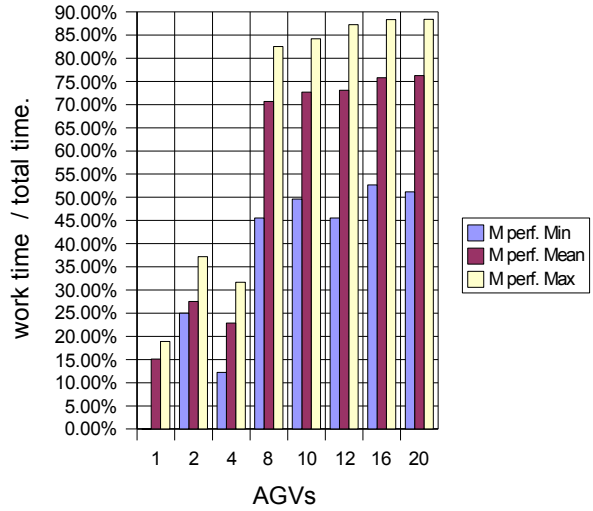

Figure 39: The machine performance

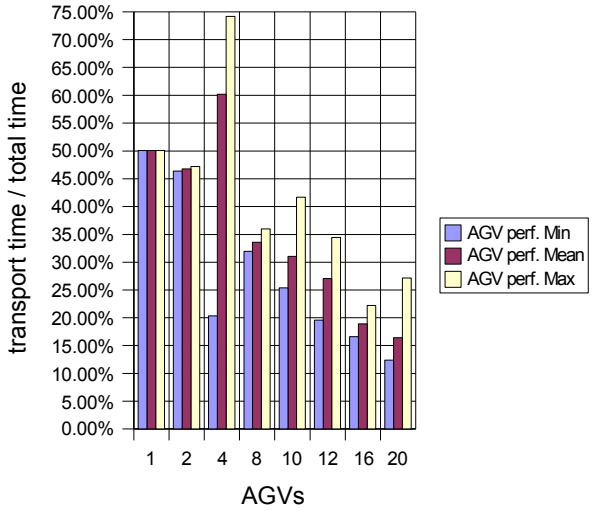

Figure 40: The AGV performance

The more AGVs are in a system, the higher is the production, although at a hight level, the added production is marginal. Probably there exists also a number of AGVs at which the system collapses due to traffic jam, but with 20 AGVs, this situation is not reached yet in the studied system.

5.6 Dependence on layout

Finally, keeping the other factors constant, the shop layout is changed.

The system parameters are:

	Number	Total capacity
Loading bays	4	120 units/h
Machines	10	30-300 units/h
Unloading bays	6	180 units
AGVs	10	-

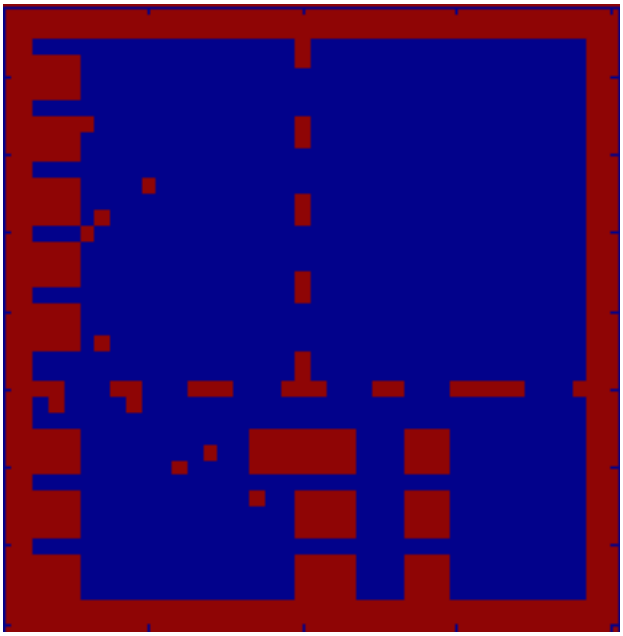

Figure 41: The "Many doors" layout. As the title says, many doors have been introduced

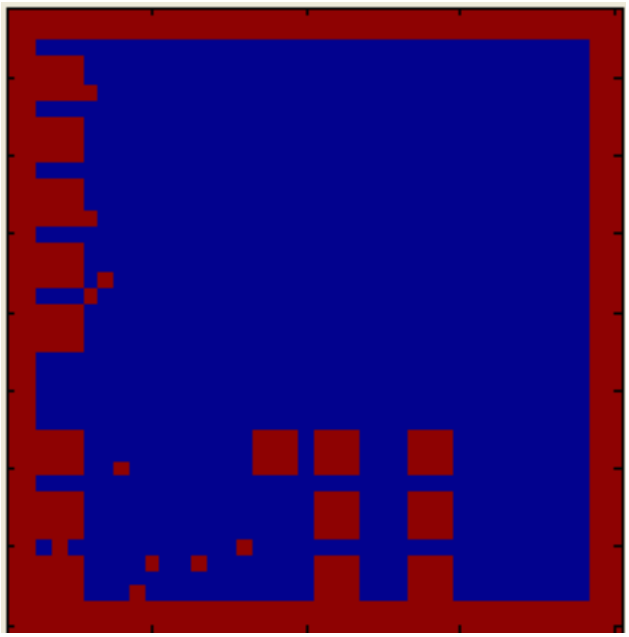

Figure 42: The "No walls" layout

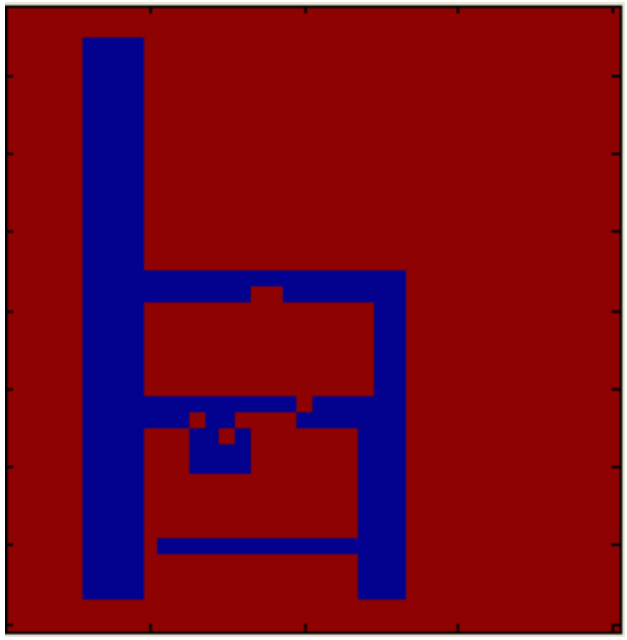

Figure 43: The "paths" layout. (The machines, loading and unloading bays are at the same position as before)

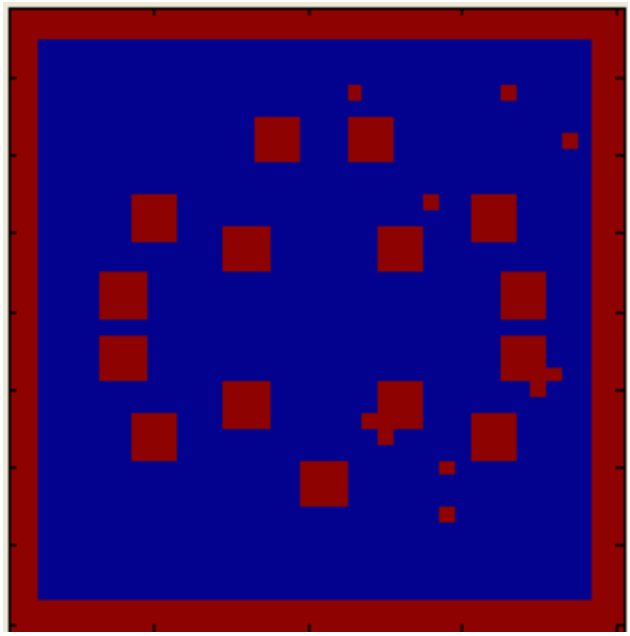

Figure 44: The "circular arrangement" layout. The loading bays are at the center, the machines in the medium circle and the unloading bays at the peripheries

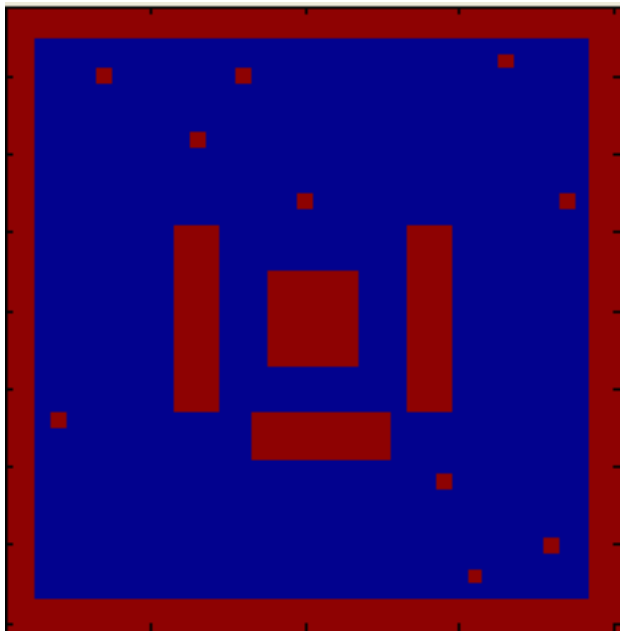

Figure 45: The "close circular arrangement" layout. The same as before but with reduced radius

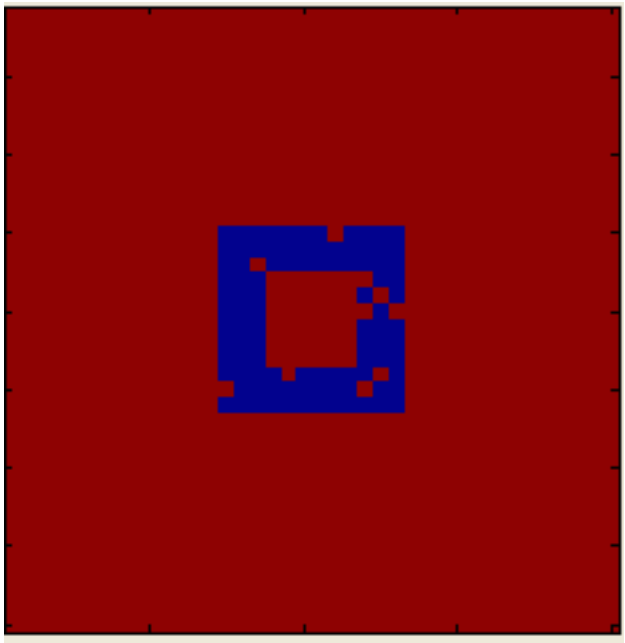

Figure 46: The "circular path" layout. (The machines, loading and unloading bays are at the same position)

The result:

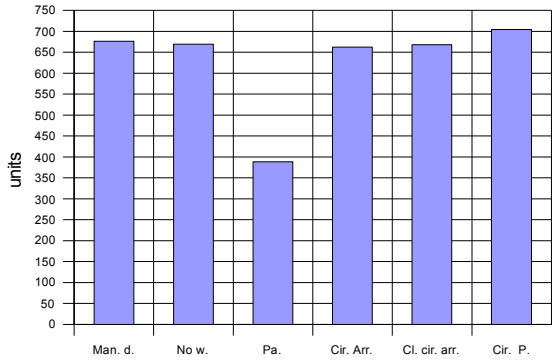

Figure 47: The production

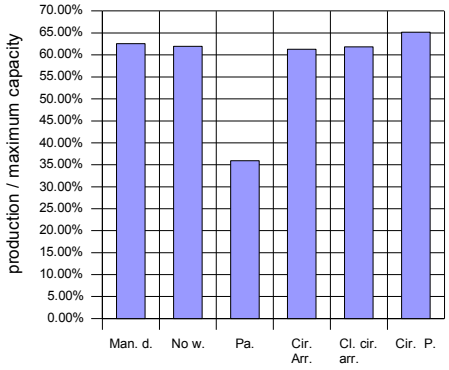

Figure 48: The system performance

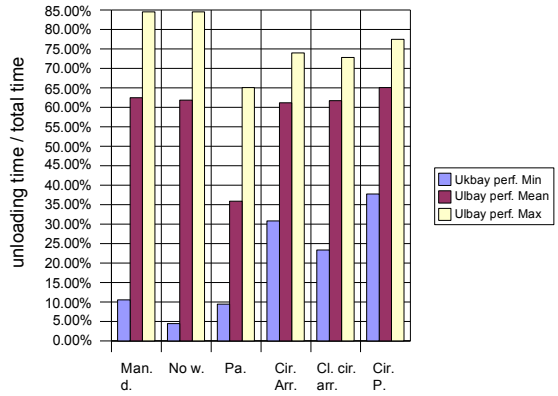

Figure 49: The unloading bay performance.

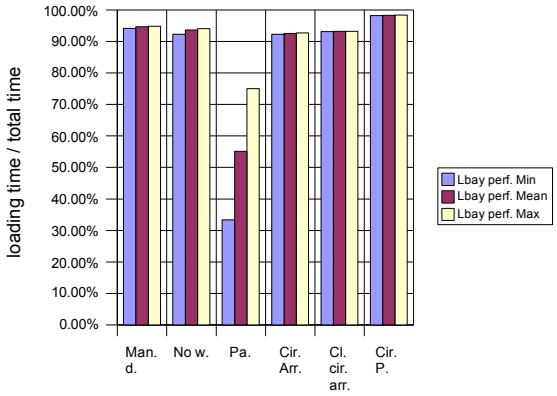

Figure 50: The loading bay performance

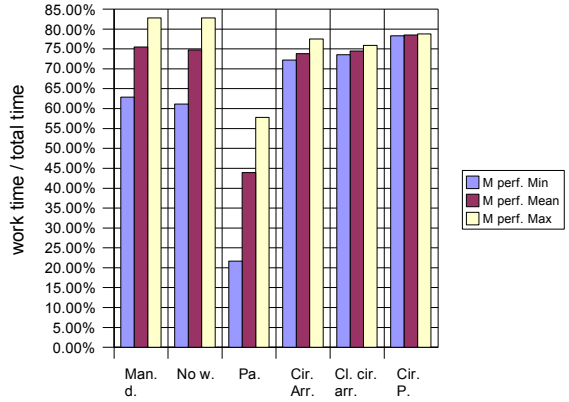

Figure 51: The machine performance

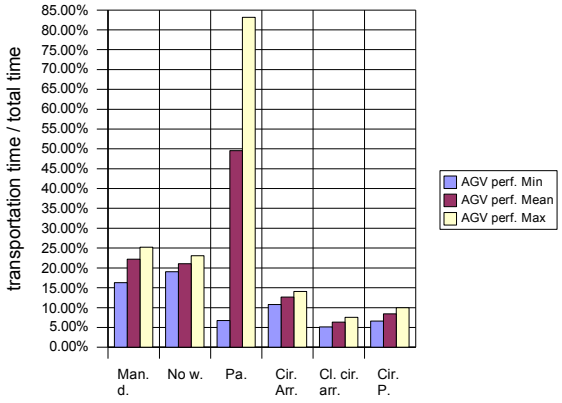

Figure 52: The AGV performance

The results of the different layouts are similar, with the exception of the “Paths” layout. It is not quite clear why the Path layout has a lower performance, especially as the “Circular Path” layout produces the best result. Some AGVs of the “Paths” layout show a quite high transportation time over total time ratio, so they might have got stuck in some corner or a traffic jam occurred at a point the path was too thin.

5.7 Not studied dependencies

There are several parameters that can be changed in the simulation program of which the influence of the performance has not been studied yet:

- Dissipation rate: The rate with which pheromones disappear. Putting the value to zero will probably cause all fields to have maximum pheromone concentration after some time, thus eliminating every gradient, while a too high value will eliminate all concentration what also will make the gradient disappear.
- The maximum pheromone concentration: An artificial limit, might be unnecessary.
- The minimum pheromone concentration: Allowing concentrations below zero makes the AGVs spread out more, as the repulsive effect becomes stronger. This can be interesting in a large layout with few AGVs.
- AGV speed: A higher AGV speed will probably increase the system performance in the same way an increased AGV number does.
- Loading speed per bay/ Unloading speed per bay/ Machine speed per bay: Those parameters are connected to the number of loading bays, unloading bays and machines and have thus been studied indirectly.

6. Conclusion

The simulation program proofed that a distributed intelligence approach can be used for the task assignment to Automated Guided Vehicles. Intelligent behaviour emerged from simple agents following simple rules. The different experiments showed, that special attention has to be put in the system design, as for example a too high job generation rate can cause the system to break down. With the experiences made it could now be attempted to design an optimized system, maximizing the production while keeping the cost as low as possible. Any such attempt can then be verified with the simulator before being implemented in a real world situation.

For future work, it would be interesting to simulate as well the distributed intelligence approach without central unit that computes the virtual pheromone, in order to see if its behaviour is similar to the approach simulated.

References

[1] Danny Weyns, Nelis Boucke, Tom Holvoet. Gradient Field-Based Task Assignment in an AGV Transportation System. *Proceedings of the fifth international joint conference on Autonomous agents and multi agent systems, Pages: 842-849*, 2006

[2] Tuan Le-Anh, René M.B.M. De Koster. Multi-Attribute Dispatching Rules For Agv Systems With Many Vehicles. *Erim Report Series Research in Management*, August 2004.

[3] Shen WM, Will P, Galstyan A, Chuong CM. Hormone-inspired self-organization and distributed control of robotic swarms, 2004

[4] McLurkin James D., Stupid Robot Tricks: A Behavior-Based Distributed Algorithm Library for Programming Swarms of Robots, 2004

[5] Dorigo M, Sahin E.: Special issue: Swarm robotics, 2004

[6] F. Mondada, A. Guignard, M. Bonani, D. Bär, M. Lauria, D. Floreano. SWARM-BOT: From Concept to Implementation, 2003

[7] Owen Holland, Chris Melhuish. Stigmergy, Self-Organization, and Sorting in Collective Robotics, 1999

[8] Craig Reinolds
http://www.red3d.com/cwr/boids/

Annex

agv.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Funcion:          AGV
% Description:      Does one programm step on all AGVs
% Dependencies:     none
% Author:           Stefan Bracher
% Date:             17.03.2007
% Status:           REL
%
% Syntaxe:          [o, d_i, agv_list]=agv(o, d_i, agv_list, min_v, max_v, agv_v)
%
% Input:            o:          Obstacle map
%                   d_i:        Pheromone maps
%                               (2+n)D Matrix, n=number of different pheromones
%                   agv_list:   List of all the agvs. One agv per line.
%                               [x, y, status, idle time]
%                               x, y: Position
%                               status:  1= idle: ready to pick up a job
%                                        2= busy:job to bring to machine
%                                        3=  job to bring to unload bay
%                               idle time: Time running around with no load
%                   min_v:      The minimal pheromone value at a point
%                   max_v:      The maximal pheromone value at a point
%                   agv_v:      Speed of the AGVs [m/sec]
%
% Output:           o:          Updated Obstacle map
%                   d_i:        Updated Pheromone maps
%                   agv_list:   Updatet AGV list
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [o, d_i, agv_list]=agv(o, d_i, agv_list, min_v, max_v, agv_v)

size_d_i=size(d_i);                         % Get the dimensions of the floor
size_pos=size(agv_list);                    % Get the length of the AGV list
agv_n=size_pos(1);                          % -> Number of agvs


for i=1:agv_n                               % For every AGV
    if (agv_list(i, 3)==1)                  % If AGV is idle
        agv_list(i, 4)=agv_list(i, 4)+1/agv_v ;    % increase idle time
```

```
 end

 old_pos=agv_list(i, 1:2);                    % old coordonates of the agv

 % RULE 1: IF NO GRANDIENT PRESENT, DO RANDOM WALK
 % Random direction
 if (round(rand)==1)                          % move along x
    go_to=old_pos+[1, 0];                     % +x
    if (round(rand)==1)
       go_to=old_pos+[-1, 0];                 % -x
    end
 else                                         % move along y
    go_to=old_pos+[0, 1];                     % +y
    if (round(rand)==1)
       go_to=old_pos+[0, -1];                 % -y
    end
 end

 % RULE 2 FOLLOW THE GRADIENT OF YOUR CURRENT STATUS
 d_i_go_to=d_i(go_to(1), go_to(2), agv_list(i, 3));          % d_i Pheromone content at the designatet new position
 d_i_alt=d_i(old_pos(1)+1, old_pos(2), agv_list(i, 3));      % An alternative spot
 if ((d_i_alt>d_i_go_to)&&(o(old_pos(1)+1, old_pos(2))==0)) % If there is more pheromone and no wall
    go_to=old_pos+[1, 0];                                    % change destination
    d_i_go_to=d_i(go_to(1), go_to(2), agv_list(i, 3));       % d_i Pheromone content at the designatet new position
 end

 d_i_alt=d_i(old_pos(1)-1, old_pos(2), agv_list(i, 3));      % An alternative spot
 if ((d_i_alt>d_i_go_to)&&(o(old_pos(1)-1, old_pos(2))==0)) % If there is more pheromone and no wall
    go_to=old_pos+[-1, 0];                                   % change destination
    d_i_go_to=d_i(go_to(1), go_to(2), agv_list(i, 3));       % d_i Pheromone content at the designatet new position
end

d_i_alt=d_i(old_pos(1), old_pos(2)+1, agv_list(i, 3));      % An alternative spot
if ((d_i_alt>d_i_go_to)&&(o(old_pos(1), old_pos(2)+1)==0))  % If there is more pheromone and no wall
    go_to=old_pos+[0, 1];                                    % change destination
    d_i_go_to=d_i(go_to(1), go_to(2), agv_list(i, 3));       % d_i Pheromone content at the designatet new position
 end

 d_i_alt=d_i(old_pos(1), old_pos(2)-1, agv_list(i, 3));      % An alternative spot
 if ((d_i_alt>d_i_go_to)&&(o(old_pos(1), old_pos(2)-1)==0))  % If there is more pheromone and no wall
    go_to=old_pos+[0, -1];                                   % change destination
    d_i_go_to=d_i(go_to(1), go_to(2), agv_list(i, 3));       % d_i Pheromone content at the designatet new position
 end

 % RULE 3: AVOID OBSTACLES
```

```
    if (o(go_to(1), go_to(2))==0)                    % If on the target place is no obstavle
        agv_list(i, 1:2)=go_to;                      % go there
        o(agv_list(i, 1), agv_list(i, 2))=1;         % Drawing the AGV on the obstacle map
        o(old_pos(1), old_pos(2) )=0;                % Freenng the old space
    end

    % RULE 4: DONT FLOCK (PUT AN ANTY PHEROMONE AROUND YOU)
      d_i(go_to(1)+1, go_to(2), agv_list(i, 3))=d_i(go_to(1)+1, go_to(2), agv_list(i, 3))-1/agv_v; % Decrease the pheromone
    if (d_i(go_to(1)+1, go_to(2), agv_list(i, 3))<min_v)    % Don't exeed minimal pheromone level
        d_i(go_to(1)+1, go_to(2), agv_list(i, 3))=min_v;
    end

       d_i(go_to(1)-1, go_to(2), agv_list(i, 3))=d_i(go_to(1)-1, go_to(2), agv_list(i, 3))-1/agv_v; % Decrease the pheromone
    if (d_i(go_to(1)-1, go_to(2), agv_list(i, 3))<min_v)    % Don't exeed minimal pheromone level
        d_i(go_to(1)-1, go_to(2), agv_list(i, 3))=min_v;
    end

       d_i(go_to(1), go_to(2)+1, agv_list(i, 3))=d_i(go_to(1), go_to(2)+1, agv_list(i, 3))-1/agv_v;   % Decrease the pheromone
    if (d_i(go_to(1), go_to(2)+1, agv_list(i, 3))<min_v)    % Don't exeed minimal pheromone level
        d_i(go_to(1), go_to(2)+1, agv_list(i, 3))=min_v;
    end

       d_i(go_to(1), go_to(2)-1, agv_list(i, 3))=d_i(go_to(1), go_to(2)-1, agv_list(i, 3))-1/agv_v;   % Decrease the pheromone
    if (d_i(go_to(1), go_to(2)-1, agv_list(i, 3))<min_v)    % Don't exeed minimal pheromone level
        d_i(go_to(1), go_to(2)-1, agv_list(i, 3))=min_v;
    end

       d_i(go_to(1), go_to(2), agv_list(i, 3))=0;  % No pheromone at AGV's position
end
```

analysis.m

```
function varargout = analysis(varargin)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Function:               Analysis
%  Description:          Analyses the result of a simulation and displays
%                             it in a GUI
%  Dependencies:   analysis.fig
%  Author:                 Stefan Bracher
%  Date:                   14.04.2007
%  Status:                  REL
%
%  Syntaxe:                handles=f1m_4lb_6ulb_10agv(handles)
%
%  Input:                  varargin:    varargin{1}.t:                      Elapsed simulation time
%                                            varargin{1}.machine_list:   List of machines
%                                            varargin{1}.v_work:             The capacity of a machine (units/h)
%                                            varargin{1}.lbay_list:          Loading bay list
%                                            varargin{1}.lspeed:             Loading bay capacity (units/h)
%                                            varargin{1}.ulbay_list:         Unloading bay list
%                                            varargin{1}.ulspeed:          Unoading bay capacity (units/h)
%                                            varargin{1}.agv_list:           AGV list
%                                            varargin{1}.o:       The obstacle map at the end of the simulation
%
%
%
%%  Output:                 varargout:     Empty
%                            GUI:              Analysis on screen
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Edit the above text to modify the response to help analysis

% Last Modified by GUIDE v2.5 13-Apr-2007 22:23:21

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @analysis_OpeningFcn, ...
                   'gui_OutputFcn',  @analysis_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
```

```
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT



% --- Executes just before analysis is made visible.
function analysis_OpeningFcn(hObject, eventdata, handles, varargin)
% Opening function of analysis

% Choose default command line output for analysis
handles.output = hObject;

%----------System information----------
% Simulation time
time=varargin{1}.t;
set(handles.time_output, 'String', time);   % Send to GUI

% Machines
machine_list=varargin{1}.machine_list;
size_machine_list=size(machine_list);
machine_n=size_machine_list(1);
set(handles.machines, 'String', machine_n); % Send to GUI

% Machine Capacity
machine_cap=varargin{1}.v_work*machine_n;
set(handles.machine_cap, 'String', machine_cap); % Send to GUI

% Loading bays
lbay_list=varargin{1}.lbay_list;
size_lbay_list=size(lbay_list);
lbay_n=size_lbay_list(1);
set(handles.lbays, 'String', lbay_n); % Send to GUI

% Loading bays Capacity
lbay_cap=varargin{1}.lspeed*lbay_n;
set(handles.lbay_cap, 'String', lbay_cap); % Send to GUI

% Unloading bays
ulbay_list=varargin{1}.ulbay_list;
size_ulbay_list=size(ulbay_list);
```

```
ulbay_n=size_ulbay_list(1);
set(handles.ulbays, 'String', ulbay_n); % Send to GUI

% Unloading bays Capacity
ulbay_cap=varargin{1}.ulspeed*ulbay_n;
set(handles.ulbay_cap, 'String', ulbay_cap); % Send to GUI

% AGVs
agv_list=varargin{1}.agv_list;
size_agv_list=size(agv_list);
agv_n=size_agv_list(1);
set(handles.agvs, 'String', agv_n); % Send to GUI

% Maximal Capacity
max_cap=max([machine_cap, lbay_cap, ulbay_cap]);
set(handles.max_cap, 'String', max_cap);

% Total units treated
units_tot=sum(ulbay_list(:, 4));
set(handles.units_tot, 'String', units_tot);

% Performance
sys_perf=units_tot/(time*max_cap/(60*60))*100;
set(handles.sys_perf, 'String', sys_perf);

% Floormap
axes(handles.obstacles);
imagesc(varargin{1}.o);
%---------------------------------------------------------------------%

%----------Machine information----------------------%
% Idle Time
set(handles.machine_max_idle_time_out, 'String', max(machine_list(:, 5)));
set(handles.machine_min_idle_time_out, 'String', min(machine_list(:, 5)));
set(handles.machine_average_idle_time_out, 'String', mean(machine_list(:, 5)));

  % blocked time
set(handles.machine_max_blocked_time_out, 'String', max(machine_list(:, 6)));
set(handles.machine_min_blocked_time_out, 'String', min(machine_list(:, 6)));
set(handles.machine_average_blocked_time_out, 'String', mean(machine_list(:, 6)));

  %peformance
  machine_performance=(time*ones(machine_n, 1)-machine_list(:, 6)-machine_list(:, 5))/time;
  set(handles.machine_max_performance_out, 'String', max(machine_performance)*100);
  set(handles.machine_min_performance_out, 'String', min(machine_performance)*100);
```

```
  set(handles.machine_average_performance_out, 'String', mean(machine_performance)*100);

%----------AGV information----------------------%
% Idle Time
set(handles.agv_max_idle_time, 'String', max(agv_list(:, 4)));
set(handles.agv_min_idle_time, 'String', min(agv_list(:, 4)));
set(handles.agv_average_idle_time, 'String', mean(agv_list(:, 4)));

  %peformance
 agv_performance=(time*ones(agv_n, 1)-agv_list(:, 4))/time;
 set(handles.agv_max_performance_out, 'String', max(agv_performance)*100);
 set(handles.agv_min_performance_out, 'String', min(agv_performance)*100);
 set(handles.agv_average_performance_out, 'String', mean(agv_performance)*100);

 %----------Loading_Bay information----------------------%
% Idle Time
set(handles.lbay_max_blocked_time_out, 'String', max(lbay_list(:, 4)));
set(handles.lbay_min_blocked_time_out, 'String', min(lbay_list(:, 4)));
set(handles.lbay_average_blocked_time_out, 'String', mean(lbay_list(:, 4)));

  %peformance
 lbay_performance=(time*ones(lbay_n, 1)-lbay_list(:, 4))/time;
 set(handles.lbay_max_performance_out, 'String', max(lbay_performance)*100);
 set(handles.lbay_min_performance_out, 'String', min(lbay_performance)*100);
 set(handles.lbay_average_performance_out, 'String', mean(lbay_performance)*100);

 %----------UnLoading_Bay information----------------------%
% Idle Time
set(handles.ulbay_max_idle_time_out, 'String', max(ulbay_list(:, 5)));
set(handles.ulbay_min_idle_time_out, 'String', min(ulbay_list(:, 5)));
set(handles.ulbay_average_idle_time_out, 'String', mean(ulbay_list(:, 5)));

  %peformance
 ulbay_performance=(time*ones(ulbay_n, 1)-ulbay_list(:, 5))/time;
 set(handles.ulbay_max_performance_out, 'String', max(ulbay_performance)*100);
 set(handles.ulbay_min_performance_out, 'String', min(ulbay_performance)*100);
 set(handles.ulbay_average_performance_out, 'String', mean(ulbay_performance)*100);

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes analysis wait for user response (see UIRESUME)
% uiwait(handles.figure1);
```

```
% --- Outputs from this function are returned to the command line.
function varargout = analysis_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
```

default_floormap.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function:       Default Floormap
% Description:    Builds the shop layout  with:
%                     2 machines
%                     1 loading bay
%                     2 unloading bays
%                     1 AGV
%                     some walls
%                     some doors
% Dependencies:   none
% Author:         Stefan Bracher
% Date:           13.04.2007
% Status:         REL
%
% Syntaxe:        handles=default_floormap(handles)
%
% Input:          handles.*.*
%
%
% Output:         handles.t:          Elapsed simulation time
%                 handles.o:          Map of the obstacles:
%                                      2D Matrix, 1=0bstacle, 0=no obstacle
%                 handles.d_i:        Pheromone maps
%                                      (2+n)D Matrix, n=number of different pheromones
%                 handles.lbay_list:  Loading bay list
%                                      n*[x-position, y-position, time_until loaded, blocked time]
%                                      n= number of loading bays (details see loading_bay.m)
%                 handles.ulbay_list:  Unloading bay list
%                                      n*[x-position, y-position, time_until unloaded, treated units, idle time]
%                                      n= number of loading bays (details see uloading_bay.m)
%                 handles.machine_list:  Machine
%                                      n*[x-position, y-position, status, time until job completion, idle time, blocked time]
%                                      n= number of machines (details see machine.m)
%                 handles.agv_list:  AGV list
%                                      n*[x-position, y-position, status, idle time]
%                                      n=number of AGVs (details see agv.m)
%                 handles.*.*         All the other  unchanged members of handles
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function handles=default_floormap(handles)

handles.t=0;                    % Elapsed simulation time=0

handles.o=zeros(40, 40);        % Making a 40x40 obstacle matrix. 0=free, 1=obstacle
```

```
% Put walls
handles.o(1:40 ,1:2)=1;
handles.o(1:40, 39:40)=1;
handles.o(1:2, 1:40)=1;
handles.o(39:40, 1:40)=1;
handles.o(25, 1:40)=1;
handles.o(1:40, 20)=1;

% Make doors
handles.o(25, 5:7)=0;
handles.o(25, 35:37)=0;
handles.o(10:12, 20)=0;

% Initiate 40x40 fields for 3 different pheromones
handles.d_i(1:40, 1:40, 1)=zeros(40);
handles.d_i(1:40, 1:40, 2)=zeros(40);
handles.d_i(1:40, 1:40, 3)=zeros(40);

% Loading bay placement
handles.lbay_list=[];                       % Clear old placements
handles.lbay_list(1, 1:4)=[31, 6, 0, 0];    % Loading bay 1
handles.o(30:32, 3:5)=1;                    %   Obstruction by Loading bay 1

%Unloading bay placement
handles.ulbay_list=[];                                  % Clear old placements
handles.ulbay_list(1, 1:5)=[31, 24, 0, 0, 0];   % Unloading bay 1
handles.o(30:32, 21:23)=1;                      % Obstruction by Unloading bay 1
handles.ulbay_list(2, 1:5)=[31, 35, 0, 0, 0];   % Unloading bay 2
handles.o(30:32, 36:38)=1;                      % Obstruction by Unloading bay 2

% Machine placement
handles.machine_list=[];                        % Clear old placements
handles.machine_list(1, 1:6)=[6, 6, 0, 0, 0, 0];  % Machine 1
handles.o(5:7, 3:5)=1;                          % Obstruction by Machine 1
handles.machine_list(2, 1:6)=[6, 35, 0, 0, 0, 0];  % Machine 2
handles.o(5:7, 36:38)=1;                        % Obstruction by Machine 2

% AGV placement
handles.agv_list=[];                        % Clear old placements
handles.agv_list(1, 1:4)=[35, 10, 1, 0];        % AGV 1, idle
```

dissipation_2D.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Function:        dissipation_2D
%  Description:     Performs a pheromone like dissipation of one timestep of
%                   the values of a matrice A
%  Dependencies:    none
%  Author:          Stefan Bracher
%  Date:            17.03.2007
%  Statut:          REL
%
%  Syntaxe:         A_new=dissipation_2D(A, O, drate, min_v, max_v)
%
%  Input:           A:        A matrix to witch the algorithm has
%                             to be applied
%                   O:        An obstacle matrix.
%                             1=Obstacle, 0=free
%                   drate     Rate of dissapearing of values, 0=continous
%                   max_v     maximal value
%                   min_v=0   minimal value
%
%  Output:          A_new:    The updatet matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function A_new=dissipation_2D(A, O, drate, min_v, max_v)

size_A=size(A);                  % Get the dimensions of the matrix
A_new=A;                         % Build the new Matrix, being equal

for x=1:size_A(1)
   for y=1:size_A(2)
   v_gone=0;                     % Initialisation of the value that left
   myvalue=A(x, y);              % The value at coordonates x, y
   v_transfer=myvalue/6;         % The value to be transferred

   % Neigbour #1
   n_x=x+1;                      % The neigbours x coordinate
   n_y=y;                        % The neigbours y coordinate
   if ((n_x<=size_A(1))&&(n_x>0)&&((n_y<=size_A(2))&&(n_y>0)))
      % if we are still in the matrix
    n_v=A_new(n_x, n_y);         % The updated value at the neigbours coordonates

   if (~O(n_x, n_y))             % If there is no obstacle at the coordonates
         n_new_v=n_v+v_transfer;   % Calculate value to transfer
```

```
   if (n_new_v>max_v)                    % if maximum value exeeded...


      n_new_v=max_v;                  % reduce to the maximum value
            end
            if (n_new_v<min_v)                % if minimal value exeeded...
                  n_new_v=min_v ;             % reduce to the maximum value
            end
           v_gone=v_gone+v_transfer;      % Update the value that left the field


A_new(n_x, n_y)=n_new_v;                % Transfer the value
      end
end

 % Neigbour #2
   n_x=x-1;                                % The neigbours x coordinate
   n_y=y;                                  % The neigbours y coordinate
   if ((n_x<=size_A(1))&&(n_x>0)&&((n_y<=size_A(2))&&(n_y>0)))
      % if we are still in the matrix
    n_v=A_new(n_x, n_y);                   % The updated value at the neigbours coordonates

   if (~O(n_x, n_y))                       % If there is no obstacle at the coordonates
           n_new_v=n_v+v_transfer;         % Calculate value to transfer


   if (n_new_v>max_v)                    % if maximum value exeeded...


      n_new_v=max_v;                  % reduce to the maximum value
            end
            if (n_new_v<min_v)                % if minimal value exeeded...
                  n_new_v=min_v ;             % reduce to the maximum value
            end
           v_gone=v_gone+v_transfer;      % Update the value that left the field


A_new(n_x, n_y)=n_new_v;                % Transfer the value
      end
end

 % Neigbour #3
   n_x=x;                                  % The neigbours x coordinate
   n_y=y+1;                                % The neigbours y coordinate
   if ((n_x<=size_A(1))&&(n_x>0)&&((n_y<=size_A(2))&&(n_y>0)))
```

```
      % if we are still in the matrix
    n_v=A_new(n_x, n_y);                    % The updated value at the neigbours coordonates

   if (~O(n_x, n_y))                        % If there is no obstacle at the coordonates
            n_new_v=n_v+v_transfer;          % Calculate value to transfer


   if (n_new_v>max_v)                     % if maximum value exeeded...


      n_new_v=max_v;                    % reduce to the maximum value
            end
            if (n_new_v<min_v)              % if minimal value exeeded...
                  n_new_v=min_v ;           % reduce to the maximum value
            end
           v_gone=v_gone+v_transfer;        % Update the value that left the field


A_new(n_x, n_y)=n_new_v;                 % Transfer the value
      end
end

 % Neigbour #4
   n_x=x;                                % The neigbours x coordinate
   n_y=y-1;                                 % The neigbours y coordinate
   if ((n_x<=size_A(1))&&(n_x>0)&&((n_y<=size_A(2))&&(n_y>0)))
      % if we are still in the matrix
    n_v=A_new(n_x, n_y);                    % The updated value at the neigbours coordonates

   if (~O(n_x, n_y))                        % If there is no obstacle at the coordonates
            n_new_v=n_v+v_transfer;          % Calculate value to transfer


   if (n_new_v>max_v)                     % if maximum value exeeded...


      n_new_v=max_v;                    % reduce to the maximum value
            end
            if (n_new_v<min_v)              % if minimal value exeeded...
                  n_new_v=min_v ;           % reduce to the maximum value
            end
           v_gone=v_gone+v_transfer;        % Update the value that left the field


A_new(n_x, n_y)=n_new_v;                 % Transfer the value
      end
```

```
end



    % The new color of the field itself
   my_new_v=A_new(x, y)-v_gone-A_new(x, y)*drate;

    if (my_new_v>max_v)                    % if maximum value exeeded...


        my_new_v=max_v;                   % reduce to the maximum value
              end
              if (my_new_v<min_v)                  % if minimal value exeeded...
                   my_new_v=min_v ;                % reduce to the maximum value
              end
   A_new(x, y)=my_new_v;                  % Assign value
    end
end
```

do_step.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function:          Do step
% Description:       Does a simulation step of 1 sec
% Dependencies:      loading_bay.m, uloading_bay.m, machine.m, dissipation_2D.m, agv.m
% Author:            Stefan Bracher
% Date:              10.04.2007
% Status:            REL
%
% Syntaxe:
%
% Input:             handles.t:          Elapsed simulation time
%                    handles.o:          Map of the obstacles:
%                                        2D Matrix, 1=0bstacle, 0=no obstacle
%                    handles.d_i:        Pheromone maps
%                                        (2+n)D Matrix, n=number of different pheromones
%                    handles.lbay_list:  Loading bay list
%                                        n*[x-position, y-position, time_until loaded, blocked time]
%                                        n= number of loading bays (details see loading_bay.m)
%                    handles.ulbay_list: Unloading bay list
%                                        n*[x-position, y-position, time_until unloaded, treated units, idle time]
%                                        n= number of loading bays (details see uloading_bay.m)
%                    handles.machine_list:  Machine
%                                        n*[x-position, y-position, status, time until job completion, idle time, blocked time]
%                                        n= number of machines (details see machine.m)
%                    handles.agv_list:   AGV list
%                                        n*[x-position, y-position, status, idle time]
%                                        n=number of AGVs (details see agv.m)
%                    handles.agv_v:      AGV speed (m/sec)
%                    handles.v_diss:     Pheromone dissipation speed (m/sec)
%                    handles.*.*         All the other unchanged members of handles
%
%
% Output:            handles.*.*         Updated handles
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function handles=do_step(handles)

handles.t=handles.t+1;          % Simulation time update

for i=1:handles.v_diss          % Depending on the dissipation speed per second do
```

```
% Loading bay step

[handles.d_i, handles.agv_list, handles.lbay_list]=loading_bay(handles.d_i, handles.agv_list, handles.lbay_list, handles.lspeed, handles.min_v, handles.max_v,
handles.v_diss);

% Unoading bay step

[handles.d_i, handles.agv_list, handles.ulbay_list]=uloading_bay(handles.d_i, handles.agv_list, handles.ulbay_list, handles.ulspeed, handles.min_v,
handles.max_v, handles.v_diss);

% Machine step

[handles.d_i, handles.agv_list, handles.machine_list]=machine(handles.d_i, handles.agv_list, handles.machine_list, handles.v_work, handles.min_v,
handles.max_v, handles.v_diss);


% Dissipation pheromone 1

handles.d_i(1:40, 1:40, 1)=dissipation_2D(handles.d_i(1:40, 1:40, 1), handles.o, handles.drate, handles.min_v, handles.max_v); % Do dissipation

% Dissipation pheromone 2

handles.d_i(1:40, 1:40, 2)=dissipation_2D(handles.d_i(1:40, 1:40, 2), handles.o, handles.drate, handles.min_v, handles.max_v); % Do dissipation

% Dissipation  pheromone 3

handles.d_i(1:40, 1:40, 3)=dissipation_2D(handles.d_i(1:40, 1:40, 3), handles.o, handles.drate, handles.min_v, handles.max_v); % Do dissipation

pause(0)

end


for i=1:handles.agv_v       % Depending on the AGV speed per second do

% AGV step

[handles.o, handles.d_i, handles.agv_list]=agv(handles.o, handles.d_i, handles.agv_list, handles.min_v, handles.max_v, handles.agv_v);

end
```

loading_bay.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Funcion:          Loading bay
%  Description:      Does one programm step on all Loading bays
%  Dependencies:     none
%  Author:           Stefan Bracher
%  Date:             19.03.2007
%  Status:           REL
%
%  Syntaxe:          [d_i, agv_list, lbay_list]=loading_bay(d_i, agv_list, lbay_list, lspeed, min_v, max_v, v_diss)
%
%  Input:            d_i:          Pheromone maps
%                                  (2+n)D Matrix, n=number of different pheromones
%                    agv_list:     List of all the agvs. One agv per line.
%                    lbay_list:    List of all the Loading bays. One bay per line.
%                                  [x, y, time until loaded, blocked time]
%                                  x, y: Position
%                                  time until loaded: Remaining time until part is ready for pick up
%                                  blocked time:     Total time spent waiting for pick up
%                    lspeed:       Time needed to load a part
%                    min_v:        The minimal pheromone value at a point
%                    max_v:        The maximal pheromone value at a point
%                    v_diss:       Dissipation speed of the pheromone
%
%  Output:           d_i:          Updated Pheromone maps
%                    agv_list:     Updatet AGV list
%                    lbay_list:    Updatet lbay list
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [d_i, agv_list, lbay_list]=loading_bay(d_i, agv_list, lbay_list, lspeed, min_v, max_v, v_diss)

size_lbay_list=size(lbay_list);                 %Determine the number of loading bays
lbay_n=size_lbay_list(1);                       % Number of loading bays

for i=1:lbay_n                                  % For all loading bays
   time_until_loaded=lbay_list(i,3);            % Time until the job is loaded
   bay_x=lbay_list(i,1);                        % x-postition
   bay_y=lbay_list(i,2);                        % y-postition



    if (time_until_loaded<0)                    % job ready
       lbay_list(i,4)=lbay_list(i,4)+1/v_diss;  % Increase blocked time
       d_i(bay_x, bay_y, 1)=d_i(bay_x, bay_y, 1)+10;  % Increase pheromone 1 ("job" waiting pheromone)
      if (d_i(bay_x, bay_y, 1)>max_v)           % pheromone overflow protection
```

```
        d_i(bay_x, bay_y, 1)=max_v;
      end
     size_agv=size(agv_list);
      agv_n=size_agv(1);                                  % Number of agvs

      for y=1:agv_n                                       % Check if there is an agv ready to pick up the job
       agv_x=agv_list(y,1);
       agv_y=agv_list(y,2);
       agv_status=agv_list(y,3);

       if ((agv_x==bay_x)&&(agv_y==bay_y)&&(agv_status==1))     % AGV at bay and ready for pick up
         agv_list(y, 3)=2;                                          % Load job on AGV
         time_until_loaded=round(60*60*v_diss/lspeed);              % Start loading new job
        end
      end

   else
   time_until_loaded=time_until_loaded-1;             % Decrease time until loaded
   end

% Save data
lbay_list(i,3)=time_until_loaded;
end
```

machine.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Funcion:          Machine
%  Description:      Does one programm step on all Machines
%  Dependencies:     none
%  Author:           Stefan Bracher
%  Date:             19.03.2007
%  Status:           REL
%
%  Syntaxe:          [d_i, agv_list, machine_list]=machine(d_i, agv_list, machine_list, v_work, min_v, max_v, v_diss)
%
%  Input:            d_i:             Pheromone maps
%                                     (2+n)D Matrix, n=number of different pheromones
%                    agv_list:        List of all the agvs. One agv per line.
%                    machine_list:    List of all Machines. One Machine per line.
%                                     [x, y, status, time untill competion, idle time, blocked time]
%                                     x, y: Position
%                                     status: Machine status: 0=idle, 1= working or done
%                                     time until completion: Remaining time until part is ready for pick up
%                                     idle time:      Total time spent waiting for a part
%                                     blocked time:   Total time spent waiting for pick up
%                    v_work:          Time needed finish a part
 %                   min_v:           The minimal pheromone value at a point
 %                   max_v:           The maximal pheromone value at a point
 %                   v_diss:          Dissipation speed of the pheromone
%
%  Output:           d_i:             Updated Pheromone maps
%                    agv_list:        Updatet AGV list
%                    machine_list:    Updatet Machine list
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [d_i, agv_list, machine_list]=machine(d_i, agv_list, machine_list, v_work, min_v, max_v, v_diss)

size_machine_list=size(machine_list);                %Determine the number of loading bays
machine_n=size_machine_list(1);                      % Number of loading bays

for i=1:machine_n                                    % For all loading bays
   time_until_done=machine_list(i,4);                % Time until the job is loaded
   m_x=machine_list(i,1);                            % x-postition
   m_y=machine_list(i,2);                            % y-postition
   busy=machine_list(i,3);                           % status of the machine


 % Machine Idle: Ready to take a job
    if (busy==0)                                     % machine idle
```

```
      machine_list(i, 5)=machine_list(i, 5)+1/v_diss;          % Increase Idle time
      d_i(m_x, m_y, 2)=d_i(m_x, m_y, 2)+10;                    % Increase pheromone 2 ('ready to take a job')
     if (d_i(m_x, m_y, 2)>max_v)                               % pheromone overflow protection
        d_i(m_x, m_y, 2)=max_v;
     end

    size_agv=size(agv_list);
     agv_n=size_agv(1);                                        % Number of agvs

     for y=1:agv_n                                             % Check if there is an agv who wants to deliver a job
       agv_x=agv_list(y,1);
       agv_y=agv_list(y,2);
       agv_status=agv_list(y,3);

       if ((agv_x==m_x)&&(agv_y==m_y)&&(agv_status==2))      % AGV at machine and delivering

         agv_list(y, 3)=1;                                       % AGV empty again
         time_until_done=round(60*60*v_diss/v_work);             % Start loading new job
         busy=1;
       end
     end
  end

  % Machine Working or done
  if ((time_until_done<0)&&(busy==1))                        % machine did job (machine blocked)
    machine_list(i, 6)= machine_list(i, 6)+1/v_diss;         % Increase blocked time
     d_i(m_x, m_y, 1)=d_i(m_x, m_y, 1)+10;                   % Increase pheromone 1 ('something to deliver')
    if (d_i(m_x, m_y, 1)>max_v)                              % pheromone overflow protection
        d_i(m_x, m_y, 1)=max_v;
    end
    size_agv=size(agv_list);
     agv_n=size_agv(1);                                      % Number of agvs

     for y=1:agv_n                                           % Check if there is an agv ready to pick up the job
       agv_x=agv_list(y,1);
       agv_y=agv_list(y,2);
       agv_status=agv_list(y,3);

      if ((agv_x==m_x)&&(agv_y==m_y)&&(agv_status==1)) % AGV at bay and ready to pick up
         agv_list(y, 3)=3;                                   % AGV loaded with finished job
         busy=0;                                             % Machine ready to take a new job
       end
     end
  else
  time_until_done=time_until_done-1;                         % Decrease time until done
```

```
    end

% Save data
machine_list(i,4)=time_until_done;
machine_list(i,3)=busy;
end
```

main.m

```
function varargout = main(varargin)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Function:               Main function
%  Description:         GUI callback organisation for the simulation
%  Dependencies:   main.fig, floormaps,   analysis.m, do_step.m
%  Author:                  Stefan Bracher
%  Date:                     14.04.2007
%  Status:                    REL
%
%  Syntaxe:                 varargout = main(varargin)
%
%  Input:                    varargin:  Empty
%
%
%
%%  Output:               varargout:     Empty
%                              GUI:              Analysis on screen
%                              video.avi      File in the folder (if video option is selected)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Edit the above text to modify the response to help main

% Last Modified by GUIDE v2.5 14-Apr-2007 23:49:44

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @main_OpeningFcn, ...
                   'gui_OutputFcn',  @main_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```
% --- Executes just before main is made visible.
function main_OpeningFcn(hObject, eventdata, handles, varargin)
% Opening function
% Choose default command line output for main
handles.output = hObject;

% My Parameter initiation
handles.t=0;                    % Simulation time=0
handles.stop=0;             % Don't interrupt any loops
handles.message='Hello: To see how I work, read the readme file in my main folder or just try me out.';
handles.graphics=0;    % Graphics turned off per default
handles.video=0;            % Video off per default
handles.v_diss=10;      % Default dissipation speed
handles.drate=0.01;      % Rate of destruction of pheromone (mass continuum: drate=0)
handles.min_v=0;          % Minimal value of pheromones at a place
handles.max_v=255;    % Maximum values of pheromones at a place
handles.agv_v=1;          % AGV Speed
handles.lspeed=30;      % Loading speed (units/h)
handles.v_work=30;      % Working speed machine (units/h)
handles.ulspeed=30;    % Unloading speed (units/h)

% Load default floormap
handles=default_floormap(handles);
handles.message='Default floormap is loaded.';

handles=update_screen(handles);         % Update screen
guidata(hObject, handles);                      % Update handles structure

% UIWAIT makes main wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = main_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes during object creation, after setting all properties.
```

```
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% Popupmenu for layout selection
handles=guidata(gcbo);          % Get the guidata
val = get(hObject,'Value');         % Returns the number of the selected item
str = get(hObject, 'String');       % Returns possible items
switch str{val};                      % Different codes to excecute depending on selection
case 'default'
   handles=default_floormap(handles);
   handles.message='Default floormap is loaded.';
   out=handles.lbay_list
 case 'richman'
   handles=richmans_floormap(handles);
   handles.message='Richmans floormap is loaded.';
    case 'optimized'
   handles=optimized_floormap(handles);
   handles.message='optimizes.';
   %---------------------------------------------------------------------
  case '10m_4lb_6ulb_10agv'
   handles=f10m_4lb_6ulb_10agv(handles);
   handles.message='10m_4lb_6ulb_10agv floormap is loaded.';
     case '8m_4lb_6ulb_10agv'
   handles=f8m_4lb_6ulb_10agv(handles);
   handles.message='8m_4lb_6ulb_10agv floormap is loaded.';
     case '6m_4lb_6ulb_10agv'
   handles=f6m_4lb_6ulb_10agv(handles);
   handles.message='6m_4lb_6ulb_10agv floormap is loaded.';
     case '4m_4lb_6ulb_10agv'
   handles=f4m_4lb_6ulb_10agv(handles);
   handles.message='4m_4lb_6ulb_10agv floormap is loaded.';
     case '2m_4lb_6ulb_10agv'
```

```
handles=f2m_4lb_6ulb_10agv(handles);
handles.message='2m_4lb_6ulb_10agv floormap is loaded.';
 case '1m_4lb_6ulb_10agv'
handles=f1m_4lb_6ulb_10agv(handles);
handles.message='1m_4lb_6ulb_10agv floormap is loaded.';
 %--------------------------------------------------------------------
    case '5m_1lb_6ulb_10agv'
handles=f5m_1lb_6ulb_10agv(handles);
handles.message='5m_1lb_6ulb_10agv floormap is loaded.';
      case '5m_2lb_6ulb_10agv'
handles=f5m_2lb_6ulb_10agv(handles);
handles.message='5m_2lb_6ulb_10agv floormap is loaded.';
      case '5m_4lb_6ulb_10agv'
handles=f5m_4lb_6ulb_10agv(handles);
handles.message='5m_4lb_6ulb_10agv floormap is loaded.';
      case '5m_6lb_6ulb_10agv'
handles=f5m_6lb_6ulb_10agv(handles);
handles.message='5m_6lb_6ulb_10agv floormap is loaded.';
      case '5m_8lb_6ulb_10agv'
handles=f5m_8lb_6ulb_10agv(handles);
handles.message='5m_8lb_6ulb_10agv floormap is loaded.';
      case '5m_10lb_6ulb_10agv'
handles=f5m_10lb_6ulb_10agv(handles);
handles.message='5m_10lb_6ulb_10agv floormap is loaded.';
      case '5m_12lb_6ulb_10agv'
handles=f5m_12lb_6ulb_10agv(handles);
handles.message='5m_12lb_6ulb_10agv floormap is loaded.';
%--------------------------------------------------------------------
         case '5m_4lb_1ulb_10agv'
handles=f5m_4lb_1ulb_10agv(handles);
handles.message='5m_4lb_1ulb_10agv floormap is loaded.';
            case '5m_4lb_2ulb_10agv'
handles=f5m_4lb_2ulb_10agv(handles);
handles.message='5m_4lb_2ulb_10agv floormap is loaded.';
            case '5m_4lb_4ulb_10agv'
handles=f5m_4lb_4ulb_10agv(handles);
handles.message='5m_4lb_4ulb_10agv floormap is loaded.';
            case '5m_4lb_6ulb_10agv'
handles=f5m_4lb_6ulb_10agv(handles);
handles.message='5m_4lb_6ulb_10agv floormap is loaded.';
            case '5m_4lb_8ulb_10agv'
handles=f5m_4lb_8ulb_10agv(handles);
handles.message='5m_4lb_8ulb_10agv floormap is loaded.';
            case '5m_4lb_10ulb_10agv'
handles=f5m_4lb_10ulb_10agv(handles);
```

```
handles.message='5m_4lb_10ulb_10agv floormap is loaded.';
            case '5m_4lb_12ulb_10agv'
handles=f5m_4lb_12ulb_10agv(handles);
handles.message='5m_4lb_12ulb_10agv floormap is loaded.';
  %--------------------------------------------------------------------
          case '5m_4lb_6ulb_1agv'
handles=f5m_4lb_6ulb_1agv(handles);
handles.message='5m_4lb_6ulb_1agv floormap is loaded.';
   case '5m_4lb_6ulb_2agv'
handles=f5m_4lb_6ulb_2agv(handles);
handles.message='5m_4lb_6ulb_2agv floormap is loaded.';
   case '5m_4lb_6ulb_4agv'
handles=f5m_4lb_6ulb_4agv(handles);
handles.message='5m_4lb_6ulb_4agv floormap is loaded.';
   case '5m_4lb_6ulb_8agv'
handles=f5m_4lb_6ulb_8agv(handles);
handles.message='5m_4lb_6ulb_8agv floormap is loaded.';
   case '5m_4lb_6ulb_12agv'
handles=f5m_4lb_6ulb_12agv(handles);
handles.message='5m_4lb_6ulb_12agv floormap is loaded.';
   case '5m_4lb_6ulb_16agv'
handles=f5m_4lb_6ulb_16agv(handles);
handles.message='5m_4lb_6ulb_16agv floormap is loaded.';
   case '5m_4lb_6ulb_20agv'
handles=f5m_4lb_6ulb_20agv(handles);
handles.message='5m_4lb_6ulb_20agv floormap is loaded.';
    %--------------------------------------------------------------------
 case 'many doors'
handles=f_many_doors(handles);
handles.message='many doors (5m_4lb_6ulb_10agv) floormap is loaded.';
    case 'no walls'
handles=f_no_walls(handles);
handles.message='no walls (5m_4lb_6ulb_10agv) floormap is loaded.';
      case 'paths'
handles=f_paths(handles);
handles.message='paths (5m_4lb_6ulb_10agv) floormap is loaded.';
      case 'circular arrangement'
handles=f_circular_arrangement(handles);
handles.message='circular arrangement (5m_4lb_6ulb_10agv) floormap is loaded.';
       case 'close circular arrangement'
handles=f_close_circular_arrangement(handles);
handles.message='close circular arrangement (5m_4lb_6ulb_10agv) floormap is loaded.';
       case 'circular path'
handles=f_circular_path(handles);
handles.message='circular path (5m_4lb_6ulb_10agv) floormap is loaded.';
```

```
end
handles=update_screen(handles);        % Update screen
guidata(gcbo, handles);         % save changes to structure



% --- Executes on button press in run_forever.
function run_forever_Callback(hObject, eventdata, handles)
handles=guidata(gcbo);          % Get the guidata
handles.stop=0;                   % Enable 24h run
handles.message='Continous run: The simulation will run until the "Stop"-Button is pressed.';
guidata(gcbo, handles);         % save changes to structure

while (handles.stop==0)
   if (handles.graphics==1)
      pause(0.2);
   else
      pause(0);
   end

handles=guidata(gcbo);          % Get the guidata
handles=do_step(handles);   % Do a step
handles=update_screen(handles);        % Update screen
guidata(gcbo, handles);         % save changes to structure
end

% --- Executes on button press in stop_button.
function stop_button_Callback(hObject, eventdata, handles)
handles=guidata(gcbo);          % Get the guidata
handles.stop=1;                   % Stop any ongoing excecution
handles.message='Simulation stopped.';
handles=update_screen(handles);        % Update screen
guidata(gcbo, handles);         % save changes to structure



% --- Executes on button press in Reset_button.
function Reset_button_Callback(hObject, eventdata, handles)
handles=guidata(gcbo);          % Get the guidata
handles.t=0;                      % Simulation time=0
handles.message='Time reinitialisated.';
handles=update_screen(handles);        % Update screen
guidata(gcbo, handles);         % save changes to structure
```

```
% --- Executes on button press in analyse_button.
function analyse_button_Callback(hObject, eventdata, handles)
analysis(handles);      % Open the analysis window


% --- Executes on button press in run_one_step.
function run_one_step_Callback(hObject, eventdata, handles)
handles=guidata(gcbo);          % Get the guidata
handles=do_step(handles);   % Do a step
handles.message='One timestep (1 sec) excecuted.';
handles=update_screen(handles);         % Update screen
guidata(gcbo, handles);         % save changes to structure



% --- Executes on button press in run6h.
function run6h_Callback(hObject, eventdata, handles)
handles=guidata(gcbo);          % Get the guidata
handles.stop=0;                         % Enable 24h run
handles.message='The simulation will run until 6h elapsed. To pause or stop, just press the "Stop"-Button at any time.';
guidata(gcbo, handles);         % save changes to structure

while ((handles.t<(60*60*6))&&(handles.stop==0))
   if (handles.graphics==1)       % Allow interruptility
      pause(0.2);
   else
      pause(0);
   end
handles=guidata(gcbo);          % Get the guidata
handles=do_step(handles);   % Do a step
handles=update_screen(handles);         % Update screen
guidata(gcbo, handles);         % save changes to structure
end


% --- Executes on button press in configurate.
function configurate_Callback(hObject, eventdata, handles)

% --- Executes on Graphics on selected
function graphics_on_Callback(hObject, eventdata, handles)
handles=guidata(gcbo);          % Get the guidata
if (get(hObject,'Value') == get(hObject,'Max'))
handles.graphics=1;
handles.message='Graphics turned on (Slower excecution).';
else
```

```
handles.graphics=0;
handles.message='Graphics turned off (Faster excecution).';
end
handles=update_screen(handles);        % Update screen
guidata(gcbo, handles);          % save changes to structure

% Hint: get(hObject,'Value') returns toggle state of graphics_on


% --- Executes during object creation, after setting all properties.
function drate_input_CreateFcn(hObject, eventdata, handles)
% hObject    handle to drate_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function drate_input_Callback(hObject, eventdata, handles)
% Change the dissipation rate
handles=guidata(gcbo);          % Get the guidata
handles.drate=str2double(get(hObject,'String'))/handles.v_diss;   % Get and set the value
handles.message='Dissipation rate updated.';   % Set a message
handles=update_screen(handles);                                   % Update screen
guidata(gcbo, handles);                                % save changes to handle



% --- Executes during object creation, after setting all properties.
function max_v_input_CreateFcn(hObject, eventdata, handles)
% hObject    handle to max_v_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
```

```
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function max_v_input_Callback(hObject, eventdata, handles)
% Change the maximal pheromone concentration
handles=guidata(gcbo);          % Get the guidata
handles.max_v=str2double(get(hObject,'String'));   % Get and set the value
handles.message='Maximal pheromone concentration updated.';   % Set a message
handles=update_screen(handles);                                % Update screen
guidata(gcbo, handles);                                % save changes to handle


% --- Executes during object creation, after setting all properties.
function min_v_input_CreateFcn(hObject, eventdata, handles)
% hObject    handle to min_v_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function min_v_input_Callback(hObject, eventdata, handles)
% Change the minimal pheromone concentration
handles=guidata(gcbo);          % Get the guidata
handles.min_v=str2double(get(hObject,'String'));   % Get and set the value
handles.message='Minimal pheromone concentration updated.';   % Set a message
handles=update_screen(handles);                                % Update screen
guidata(gcbo, handles);                                % save changes to handle


% --- Executes during object creation, after setting all properties.
function diss_v_input_CreateFcn(hObject, eventdata, handles)
% hObject    handle to diss_v_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function diss_v_input_Callback(hObject, eventdata, handles)
% Change the dissipation speed
handles=guidata(gcbo);          % Get the guidata
handles.v_diss=str2double(get(hObject,'String'));    % Get and set the value
handles.message='Dissipation speed updated.';   % Set a message
handles=update_screen(handles);                                  % Update screen
guidata(gcbo, handles);                                  % save changes to handle



% --- Executes during object creation, after setting all properties.
function agv_v_input_CreateFcn(hObject, eventdata, handles)
% hObject    handle to agv_v_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function agv_v_input_Callback(hObject, eventdata, handles)
% Change the AGV speed
handles=guidata(gcbo);          % Get the guidata
handles.agv_v=str2double(get(hObject,'String'));   % Get and set the value
handles.message='AGV speed updated.';   % Set a message
handles=update_screen(handles);                                  % Update screen
guidata(gcbo, handles);                                  % save changes to handle
```

```
% --- Executes during object creation, after setting all properties.
function lspeed_input_CreateFcn(hObject, eventdata, handles)
% hObject    handle to lspeed_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function lspeed_input_Callback(hObject, eventdata, handles)
handles=guidata(gcbo);          % Get the guidata
handles.lspeed=str2double(get(hObject,'String'));   % Get and set the value
handles.message='Loading speed updated.';   % Set a message
handles=update_screen(handles);                                 % Update screen
guidata(gcbo, handles);                                 % save changes to handle


% --- Executes during object creation, after setting all properties.
function ulspeed_input_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ulspeed_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function ulspeed_input_Callback(hObject, eventdata, handles)
handles=guidata(gcbo);          % Get the guidata
handles.ulspeed=str2double(get(hObject,'String'));   % Get and set the value
handles.message='Unloading speed updated.';   % Set a message
```

```
handles=update_screen(handles);                         % Update screen
guidata(gcbo, handles);                                 % save changes to handle


% --- Executes during object creation, after setting all properties.
function v_work_input_CreateFcn(hObject, eventdata, handles)
% hObject    handle to v_work_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function v_work_input_Callback(hObject, eventdata, handles)
handles=guidata(gcbo);          % Get the guidata
handles.v_work=str2double(get(hObject,'String'));   % Get and set the value
handles.message='Loading speed updated.';   % Set a message
handles=update_screen(handles);                         % Update screen
guidata(gcbo, handles);                                 % save changes to handle


% --- Executes on button press in save_video.
function save_video_Callback(hObject, eventdata, handles)
handles=guidata(gcbo);          % Get the guidata
if (get(hObject,'Value') == get(hObject,'Max'))
handles.video=1;
handles.message='Saving frames to video (Watch memory!!!!!!)';
else
handles.video=0;
handles.message='Video turned off.';
end
handles=update_screen(handles);         % Update screen
update_screen(handles)
guidata(gcbo, handles);                                 % save changes to handle

% --- Executes on button press in make_videofile.
function make_videofile_Callback(hObject, eventdata, handles)
handles=guidata(gcbo);          % Get the guidata
```

```
handles.message='Saving video to file. This may take some time...)';
handles=update_screen(handles);         % Update screen
movie2avi(handles.film, 'video');           % Merge Film to a "video.avi".
clear handles.film;                         % delete the frames
guidata(gcbo, handles);          % save changes to structure
% hObject    handle to make_videofile (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

unloading_bay.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Funcion:          Unloading bay
%  Description:      Does one programm step on all Unloading bays
%  Dependencies:     none
%  Author:           Stefan Bracher
%  Date:             19.03.2007
%  Status:           REL
%
%  Syntaxe:          function [d_i, agv_list, ulbay_list]=uloading_bay(d_i, agv_list, ulbay_list, ulspeed, min_v, max_v, v_diss)
%
%  Input:            d_i:          Pheromone maps
%                                  (2+n)D Matrix, n=number of different pheromones
%                    agv_list:     List of all the agvs. One agv per line.
%                    ulbay_list:   List of all the Unloading bays. One bay per line.
%                                  [x, y, time until unloaded, units treatd]
%                                  x, y: Position
%                                  time until unloaded: Remaining time until part is unloaded
%                                  units treated:   Total number of units unloaded during simulation
%                                  idle time:   Total time spent waiting for delivery
%                    ulspeed:      Time needed to unload a part
%                    min_v:        The minimal pheromone value at a point
%                    max_v:        The maximal pheromone value at a point
%                    v_diss:       Dissipation speed of the pheromone
%
%  Output:           d_i:          Updated Pheromone maps
%                    agv_list:     Updatet AGV list
%                    ulbay_list:   Updatet ulbay list
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [d_i, agv_list, ulbay_list]=uloading_bay(d_i, agv_list, ulbay_list, ulspeed, min_v, max_v, v_diss)

size_ulbay_list=size(ulbay_list);                 %Determine the number of unloading bays
ulbay_n=size_ulbay_list(1);                       % Number of unloading bays

for i=1:ulbay_n                                   % For all unloading bays
   time_until_unloaded=ulbay_list(i,3);           % Time until the job is unloaded
   bay_x=ulbay_list(i,1);                         % x-postition
   bay_y=ulbay_list(i,2);                         % y-postition



    if (time_until_unloaded<0)                    %ready
       d_i(bay_x, bay_y, 3)=d_i(bay_x, bay_y, 3)+10;    % Increase pheromone 3 (ready to take finished item)
```

```
    if (d_i(bay_x, bay_y, 3)>max_v)                    % pheromone overflow protection
       d_i(bay_x, bay_y, 3)=max_v;
    end

    ulbay_list(i, 5)=ulbay_list(i, 5)+1/v_diss;        % Add idle time
    size_agv=size(agv_list);
    agv_n=size_agv(1);                                 % Number of agvs

    for y=1:agv_n                                      % Check if there is an agv ready to bring a job
     agv_x=agv_list(y,1);
     agv_y=agv_list(y,2);
     agv_status=agv_list(y,3);

     if ((agv_x==bay_x)&&(agv_y==bay_y)&&(agv_status==3))      % AGV at bay and ready to deliver

       agv_list(y, 3)=1;                               % Unload the agv
       ulbay_list(i, 4)=ulbay_list(i, 4)+1;            % Add a unit to do done units
       time_until_unloaded=round(60*60*v_diss/ulspeed);   % Start unloading
      end
     end


  else
  time_until_unloaded=time_until_unloaded-1;         % decrease time until unloaded
  end

% Save data
ulbay_list(i,3)=time_until_unloaded;
end
```

update_screen.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Function:           Update screen
%  Description:        Updates the GUI
%  Dependencies:       main.fig
%  Author:             Stefan Bracher
%  Date:               14.03.2007
%  Status:             REL
%
%  Syntaxe:            handles=update_screen(handles)
%
%  Input:              handles.t:          Elapsed simulation time
%                      handles.o:          Map of the obstacles:
%                                          2D Matrix, 1=0bstacle, 0=no obstacle
%                      handles.d_i:        Pheromone maps
%                                          (2+n)D Matrix, n=number of different pheromones
%                      handles.lbay_list:  Loading bay list
%                                          n*[x-position, y-position, time_until loaded, blocked time]
%                                          n= number of loading bays (details see loading_bay.m)
%                      handles.ulbay_list:  Unloading bay list
%                                          n*[x-position, y-position, time_until unloaded, treated units, idle time]
%                                          n= number of loading bays (details see uloading_bay.m)
%                      handles.machine_list:  Machine
%                                          n*[x-position, y-position, status, time until job completion, idle time, blocked time]
%                                          n= number of machines (details see machine.m)
%                      handles.agv_list:  AGV list
 %                                         n*[x-position, y-position, status, idle time]
 %                                         n=number of AGVs (details see agv.m)
 %                     handles.agv_v:      AGV speed (m/sec)
  %                    handles.v_diss:   Pheromone dissipation speed (m/sec)
 %                     handles.*.*         All the other  unchanged members of handles
 %                     handles.film:       Video frames
%
%
%  Output:             handles.*.*         Updated handles
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function handles=update_screen(handles)

% Update the time
h=floor(handles.t/(60*60));
min=floor((handles.t-h*60*60)/60);
sec=handles.t-h*60*60-min*60;
set(handles.time_sec, 'String', num2str(sec));
set(handles.time_min, 'String', num2str(min));
```

```
set(handles.time_h, 'String', num2str(h));

%Update parameters
set(handles.drate_input, 'String', num2str(handles.drate));
set(handles.max_v_input, 'String', num2str(handles.max_v));
set(handles.min_v_input, 'String', num2str(handles.min_v));
set(handles.diss_v_input, 'String', num2str(handles.v_diss));
set(handles.lspeed_input, 'String', num2str(handles.lspeed));
set(handles.ulspeed_input, 'String', num2str(handles.ulspeed));
set(handles.v_work_input, 'String', num2str(handles.v_work));

% Update the message
set(handles.output_message, 'String', handles.message);

if (handles.graphics==1)          % If graphics turned on
% Print the figures
axes(handles.floor_map);
set(gcf,'DoubleBuffer','on');
imagesc(handles.o);

if (handles.video==1)             % If video shooting is on
handles.film(handles.t+1) = getframe;          % Save frame to film
end

axes(handles.pheromone_job_waiting);
caxis([handles.min_v handles.max_v])
colormap(jet)
image(handles.d_i(1:40, 1:40, 1))
axes(handles.pheromone_idle);
image(handles.d_i(1:40, 1:40, 2))
axes(handles.pheromone_unload);
image(handles.d_i(1:40, 1:40, 3))
end
```