

UNIVERSITY OF PARDUBICE
Faculty of Electrical Engineering and Informatics

Implementation of a high-level control system
for a teaching aid aimed as a supporting tool when
teaching state-space search method

Bc. Filip Majerík

Master thesis
2017

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2016/2017

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Filip Majerík**
Osobní číslo: **I15217**
Studijní program: **N2646 Informační technologie**
Studijní obor: **Informační technologie**
Název tématu: **Implementace nadřazeného řídicího systému laboratorní úlohy určené pro výuku problematiky prohledávání stavového prostoru**
Zadávací katedra: **Katedra softwarových technologií**

Z á s a d y p r o v y p r a c o v á n í :

V rámci diplomové práce bude navržen software zajišťující veškeré funkce nadřazeného řídicího systému, který je součástí laboratorní úlohy určené jako podpůrný prostředek při výuce problematiky prohledávání stavového prostoru. Navržený software bude zajišťovat jak zpracování dat získaných z kamerového systému, tak komunikaci s kamerovým systémem a mobilním robotem, stejně jako interakci s uživatelem. Software bude navržen tak, aby umožňoval snadnou aplikaci různých algoritmů prohledávání stavového prostoru. Za tímto účelem bude navrženo standardizované rozhraní, které bude poskytovat těmto algoritmům informaci o struktuře bludiště, ve kterém se robot pohybuje, stejně jako o aktuální poloze robota. Zpětně bude od algoritmů získávat plán cesty. Nadřazený systém pak zajistí převod plánu do formy srozumitelné pro robota a jeho následné odeslání robotovi. V rámci této diplomové práce bude vytvořen jeden ukázkový algoritmus pro hledání nejkratší cesty v bludišti. Navržené řešení bude ověřeno v reálných podmínkách.

Rozsah grafických prací:

Rozsah pracovní zprávy:

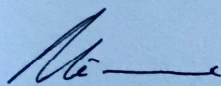
Forma zpracování diplomové práce: **tištěná**

Seznam odborné literatury: **viz příloha**

Vedoucí diplomové práce: **Ing. Pavel Škrabánek, Ph.D.**
Katedra řízení procesů

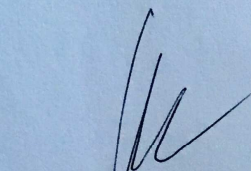
Datum zadání diplomové práce: **31. října 2016**

Termín odevzdání diplomové práce: **17. května 2017**



Ing. Zdeněk Němec, Ph.D.
děkan

L.S.



prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2016

Příloha zadání diplomové práce

Seznam odborné literatury:

*YOO, Terry S. (ed.). *Insight into images: principles and practice for segmentation, registration, and image analysis*. Wellesley: A K Peters, 2004. ISBN 15-688-1217-5.

*BURGER, Wilhelm. a Mark. BURGE. *Principles of digital image processing: fundamental techniques*. London: Springer, 2009. ISBN 978-184-8001-916.

*BURGER, Wilhelm a Mark BURGE. *Principles of digital image processing: core algorithms*. London: Springer, 2009. *Undergraduate topics in computer science*. ISBN 18-480-0190-8.

*RUSSELL, Stuart J. a Peter NORVIG. *Artificial intelligence: a modern approach*. 3rd ed. Harlow: Pearson Education, 2014. ISBN 978-1-29202-420-2.

*IHNÁT, Vojtěch. *Návrh a realizace komunikačního protokolu a softwaru pro bezdrátový přenos dat mezi robotickým vozítkem a MATLABem*. Pardubice, Czech Republic, 2015. *Bakalářská práce*. Univerzita Pardubice. Vedoucí práce Pavel Škrabánek.

*ŠKRABÁNEK, Pavel a Petr DOLEŽEL. *Attractive robot's design suitable for image processing*. In: *Proceedings of the Mendel 2014: 20th International Conference on Soft Computing*. Brno, Czech Republic: University of Technology, 2014, s. 217-222. ISBN 978-80-214-4984-8. ISSN 1803-3814.

*ŠKRABÁNEK, Pavel. *Labyrinth arrangement analysis based on image processing*. In: *Mendel 2015: Recent Advances in Soft Computing*. Springer International Publishing, 2015, 305 - 316. ISBN 978-3-319-19823-1. ISSN 2194-5357.

*ŠKRABÁNEK, Pavel, VODIČKA, Pavel a YILDIRIM-YAYILGAN, Sule. *Control System of a Semi-Autonomous Mobile Robot*. In: *14th IFAC International Conference on Programmable Devices and Embedded Systems (PDeS)* [online]. Brno/Lednice: VUT Brno, 2016, s. 10 [cit. 2016-10-03].

Author's Declaration

I hereby declare:

This thesis was prepared separately. All the literary sources and the information I used in the thesis are listed in the bibliography. I got familiar with the fact that the rights and obligations arising from the Act No. 121/2000 Coll., Copyright Act, apply to my thesis, especially with the fact that the University of Pardubice has the right to enter into a license agreement for use of this thesis as a school work pursuant to § 60, Section 1 of the Copyright Act, and the fact that should this thesis be used by me or should a license be granted for the use to another entity, the University of Pardubice is authorized to claim a reasonable contribution from me to cover the costs incurred during making of the thesis, according to the circumstances up to the actual amount thereof.

I am aware that my thesis will be accessible to the public in the University Library and via the Digital Library of the University of Pardubice in agreement with the article 47b of the Act No. 111/1998 Coll., on Higher Education Institutions, and on the Amendment and Supplement to some other Acts (the Higher Education Act), as subsequently amended, and with the University Pardubice's directive no. 9/2012.

In Pardubice on 15. 05. 2017

Bc. Filip Majerík

Acknowledgements

I am grateful to the many people that have provided inspiration and support during the course of this work. Mainly, I would like to thank my supervisor Ing. Pavel Škrabánek PhD. for his cool head, his great access and good cooperation. Thanks also to my coworkers what help me with some parts of programming and to Ing. Karel Šimerda for consulting on some problems. And thanks also to Paul Charles Hooper, B.A. Dip.Ed. for his time while consulting the English grammar.

Next I would like to thank my friends for accepting my angry feels and my worst days. Last but not the least, I would like to thank my family: my parents and to my wife Andrea Blažíčková witch daughter Žofie for supporting me spiritually throughout writing this thesis and my life in general.

Thank to everyone.

Název

Implementace nadřazeného řídicího systému laboratorní úlohy určené pro výuku problematiky prohledávání stavového prostoru

Anotace

Tato diplomová práce se zabývá detekcí objektů v reálném čase za pomoci knihovny OpenCV. Prvně jsou popsány základy plánování a plánovací algoritmy. Následně jsou popsány základní techniky zpracování obrazů a základní principy pro detekování objektů v obraze. Dále je rozebrán návrh a implementace učební pomůcky – software nadřazeného řídicího systému.

Klíčová slova

Počítačové vidění, umělá inteligence, plánovací algoritmy, plánování cesty, zpracování obrazu

Title

Implementation of a high-level control system in a teaching aid aimed as a supporting tool when teaching state-space search methods

Annotation

This master thesis is focused to the object detection in a real time using the OpenCV library. At first is described the basis of path-planning and path-planning algorithms. Next are described a basic techniques for an image processing and a basic principles for the image detection. Last is analysis of a problem and a software design with its implementation as teaching aid – high-level control system software.

Keywords

Computer vision, artificial intelligence, path-plan algorithms, path-planning, image processing

Content list

List of acronyms	10
List of pictures	11
List of tables	12
Introduction	13
1 Description of the laboratory work.....	14
1.1 Purpose of the teaching aid.....	14
1.2 Components of the laboratory work	14
1.2.1. Maze	15
1.2.2. Camera system.....	15
1.2.3. Mobile robot	16
1.2.4. High-level control system.....	16
2 The Path-planning	18
2.1 Short introduction to path-planning.....	18
2.2 Search for path-plan.....	18
2.2.1. Uninformed methods	20
2.2.2. Uninformed methods comparison	26
2.2.3. Informed methods.....	27
2.2.4. Heuristic function	30
2.3 Configuration space and decomposition	31
3 Image Processing	36
3.1 Digital image representation.....	36
3.2 Point operations	39
3.2.1. Brightness transformations	40
3.2.2. Histogram equalization.....	41
3.2.3. Thresholding operations	43
3.3 Geometric transformations	44
3.3.1. Translation	46
3.3.2. Scale	46
3.3.3. Rotation	47
3.3.4. Shear	47
3.4 Composite affine transformation	48
3.5 Perspective transformation	48

3.6	Image smoothing	49
3.7	Shape analysis	51
4	Problem analysis.....	54
4.1	Objectives of teaching aid	55
4.2	Requirements	55
4.2.1.	Functional requirements	55
4.2.2.	Non-functional requirements.....	55
4.3	Ready-to-use tools	56
4.3.1.	Maze detector analysis	56
4.3.2.	Robot detection.....	56
5	Software design.....	57
5.1	Modules	57
5.1.1.	GUI & environment.....	57
5.1.2.	Camera system module.....	58
5.1.3.	Maze layout analysis	58
5.1.4.	Robot observer.....	60
5.1.5.	Client manager.....	60
5.1.6.	Robot communication module.....	62
5.2	Analytic Class diagram.....	63
5.3	Implementation	64
6	Experimental verification of the solution.....	66
6.1	Maze detection.....	66
6.2	Student's routine interface.....	66
6.3	Getting path-plan from routine and transform to a sequence of steps.....	66
6.4	Robot detection.....	67
7	Evaluation of verification.....	68
7.1	Maze detection.....	69
7.2	Robot observing.....	70
7.3	Implementation of client application with A* algorithm	72
	Conclusion	74
	Literature	75
	Attachment A – Example usage of communication protocol	78
	Attachment B – Accompanying CD.....	79

List of acronyms

BFS – Breadth-first search
CV – Computer vision
DFS – Depth-first search
DSLR camera – Digital Single-Lens Reflex camera
FIFO – First-in, First-out (queue type)
HRCT – High-resolution computed tomography
HLCS – High-level control system¹
IP camera – Internet Protocol Camera
LIFO – Last-in, First-out (queue type)
RGB – Red Green Blue²
RTSP – Real Time Streaming Protocol

¹ It is part of this thesis

² In usage with color space/images

List of pictures

Figure 1 – An example of a correctly assembled maze captured by the camera system.....	15
Figure 2 – An example of a correctly installed camera's system	15
Figure 3 – Control System Prototype	17
Figure 4 - Simplified graph of Romania (Russell, et al., 2010)	19
Figure 5 – Examples of possible start and goal state representations in the 8-puzzle (Russell, et al., 2010).....	20
Figure 6 - Order which the nodes are expanded by BFS.....	21
Figure 7 - Breadth-first algorithm pseudocode (Sedgewick, et al., 2011).....	22
Figure 8 - Breadth-first search on a simple binary tree, partial of the solution is indicated from left to right	22
Figure 9 – Order which the nodes are expanded by DFS.....	23
Figure 10 - Depth-first algorithm pseudocode (Sedgewick, et al., 2011)	24
Figure 11 - Depth-first search on a simple binary tree, partial solution is indicated from left to right	24
Figure 12 - Depth-first iterative deeping pseudocode (Sedgewick, et al., 2011)	25
Figure 13 - A schematic view of a bidirectional search	25
Figure 14 - Big-O Complexity Chart.....	27
Figure 15 - Greedy search with dead end problem.....	28
Figure 16 - Greedy search recursive algorithm pseudocode	28
Figure 17 - A* (A Start) algorithm pseudocode	29
Figure 18 - Euclidean distance	31
Figure 19 - Manhattan distance	31
Figure 20 - Example of exact cell decomposition (Latombe, 1991)	32
Figure 21 - Adjacency relations between the cells (Latombe, 1991)	33
Figure 22 - Fixed cell decomposition problem.....	33
Figure 23 - Workspace decomposed by adaptive cell decomposition method (Siegwart, et al., 2004).....	34
Figure 24 - Comparison of resulted path-plan for these two methods.	34
Figure 25 - Most used bitmap grids – (a) Square grid. (b) Hexagonal grid.	37
Figure 26 - (a) 400 x 400px, (b) 100 x 100px	38
Figure 27 - Vectorized photo of Theodore Roosevelt	39
Figure 28 – Apply the function P to f(x,y)	40
Figure 29 - Gray-scale transformations	41
Figure 30 - Histogram equalization (Left) Original image. (Right) Equalized image.	42
Figure 31 - Histogram equalization for Fig. 30	42
Figure 32 - Image thresholding (1) Original image, (2) Simple thresholding, (3) Otsu method and (4) Poly curve fitting (Chandrakala, et al., 2016)	44
Figure 33 - Spatial transformation (Rhody, 2005)	45
Figure 34 - Example of translation	46
Figure 35 - Example of scale - $S_x, S_y = 2$ (Enlargement)	47
Figure 36 - Example of rotation - $\theta = 45^\circ$	47

Figure 37 - Example of shear along x-axis.....	48
Figure 38 - Result – Translation, Scale 2x, Rotation $\theta = 75^\circ$	48
Figure 39 - Example of apply a perspective transformation	49
Figure 40 - Example image with applied Gaussian filter	50
Figure 41 - 2D Gaussian function graph	51
Figure 42 - Sub image divide	56
Figure 43 - Final graphic user interface.....	57
Figure 44 - Maze with 3 corner markers and calculated last corner.....	59
Figure 45 - Maze with 4 corner (a) before perspective transformation (b) after.....	60
Figure 46 - GUI form for routine selection	61
Figure 47 – Comparison of trajectories for 270° rotation and 90° rotation to same final direction	62
Figure 48 - Analytic Class diagram.....	63
Figure 49 - HLCS Source Packages	64
Figure 50 - Maze with undesirable light.....	70
Figure 51 - Bad detected maze while is too much rotated.....	70
Figure 52 – Key points of sample robot markers (colored circles)	71
Figure 53 – Sample key pointed maze with markers.....	71
Figure 54 - Sample bad key point association.....	72
Figure 55 - Simple routine GUI.....	72
Figure 56 - Example of generated path by testing routine	73
Figure 57 - Example output during the communication with routine	73

List of tables

Table 1 - Evaluation of tree-search versions of algorithm (Even, 2011) (Cormen, 2009) (Sedgewick, et al., 2011)	26
Table 2 - Comparison of raster and vector model	39
Table 3 - Evaluation of the maze detection test scenario	68
Table 4 - Evaluation of the student's routine interface test scenario	68
Table 5 - Evaluation of the getting path-plan from routine and transform to a sequence of steps test scenario	69
Table 6 - Evaluation of the robot detection test scenario	69

Introduction

Recently, one of the most rapidly increasing parts of information technology science is artificial intelligence and other associated areas such as image processing or computer vision. In order to strengthen a student understanding of these areas, various teaching aids have been developed. A teaching aid aimed at path-planning practicing has been designed at the University of Pardubice, Faculty of Electrical Engineering and Informatics. The teaching aid supports development of various student skills. Beside a path-planning theory, the students practice programming, and they must experience in optimization of algorithms. One of the most complex parts of the teaching aid is a high-level control system. This system has been developed within my thesis.

The high-level control system is a central part of the teaching aid, which ensures communication with a user, a mobile robot, a camera and a path-planning routine. Moreover, the control system must provide additional services such as image processing, object detection, or object localization.

Since the teaching aid is a core topic of this thesis, a detailed description of the aid is logically a content of the first section of the thesis. The teaching aid is aimed at practicing path-planning theory. Thus, a brief description of search methods, heuristic functions, configuration space and decomposition methods is given in the following section. The high-level control system, gathers information about the real world using the camera; hence, image-processing related tasks are very important in the context of this work. Thus, the next section deals with image processing fundamentals. This section gives an overview of various image-processing operations, as well as an introduction into a digital image representation. These three sections represent the theoretical part of the thesis.

A practical part of the thesis deals with design and implementation of the teaching aid. An analysis of requirements of the high-level control system is covered at first. Description of the process of design can be found in the following section. Evaluation of the proposed solution is explained in the penultimate section. Finally, a conclusion is given in the last section of the thesis.

1 Description of the laboratory work

The basis of this laboratory work is to provide to students a training environment for their experiences in programming and artificial intelligence. A little bit about computer software communication over sockets is also provided. This teaching aid was described in an article, Control System of a Semi-Autonomous Mobile Robot (Škrabánek, et al., 2016) and based on next two articles, High-Level Control System for a Path-Planning Teaching Aid (Škrabánek, et al., 2016) and Attractive Robot's design suitable for image processing (Škrabánek, et al., 2014).

This laboratory work is based on a camera system which gives information about how the problem looks. The base high-level control system communicate with clients (problem solvers), robot (executes command from control system) and maze.

However, the maze is not connected to any computer. For work with maze, a camera system is used which detects how the maze looks and gives information about robot position.

Students can implement any path-planning algorithm to their software. It depends on their imagination. Cooperation with this laboratory work can give feedback to the student on how good is his/her implemented algorithm.

1.1 Purpose of the teaching aid

A student can implement any path-planning algorithm in their routine. The application can communicate with the control system by the socket interface communication.

A basic requirement on the control system is an ability to create a simple communication interface with any programming language. Thanks to this, students can choose their favorite language. The proposed solution will be language independent thanks to the socket communication.

In such a way, students can use our communication interface to get all information about teaching aid configuration and give back to our interface their solution for this problem. Once the student's application returns a solution it is displayed in our high-level control system for checking, and robots will move forward according to this solution.

If a student's solution is not good, the robot may collide with a wall in the maze. This behavior is expected. If the path-plan is good, the robot safely moves from a start (actual) position to a target position.

1.2 Components of the laboratory work

The laboratory work composes of several components. All of these components have their roles in this system. We can divide the components to three groups. In the first one, space is composed from partitions called maze. The second group is the computer system. This computer system is composed of camera, network, high-level control system and client solvers (student routines). The last item consists of the mobile robot.

1.2.1. Maze

The maze is constructed from a building kit and consists of black and red labeled partitions, posts and small yellow square markers. Using the partitions and posts, a maze of various sizes and layouts can be constructed. In Figure 1 is shown one possible layout of the maze constructed by a building kit. An outer shape of the maze must be rectangular. The yellow markers must be always placed in corners. The markers are key for a localization of the maze in images. As will be explained later, meeting of these requirements is fundamental for correct image processing.



Figure 1 – An example of a correctly assembled maze captured by the camera system.

1.2.2. Camera system

The camera system consists of the camera and a camera stand with a movable arm. Maximum height of the stand is 3 meters and the arm is 1.5 meter long. For correct usage, the arm must be placed in a horizontal position and a camera's swath must be perpendicular to the maze. It means that the camera will be parallel with a base of the maze in an ideal state. In other cases, some unwanted distortions of the images may occur. The distortions may prevent a maze layout detection.

The camera is fixed on the end part of the arm. Within this thesis, an IP camera D-Link DCS-935L was used. The camera can capture images or stream a live video. It supports an RGB or a gray scale mode. The maximal resolution of the camera is 1280×720 pixels. The camera supports a WiFi network connection using a RTSP protocol.

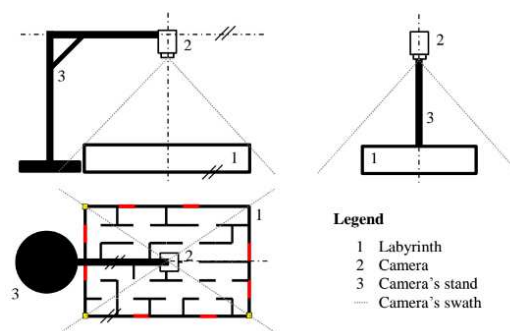


Figure 2 – An example of a correctly installed camera's system³

³ Image taken from the material owned by Ing. Pavel Škrabánek Ph.D.

1.2.3. Mobile robot

The mobile robot is one of the main components of this teaching aid. It is based on an Arduino building kit. A robot can communicate with our system by Bluetooth technology. All commands for the robot are received via this communication interface.

Received commands are in step format. There are only three type of steps – rotate, forward and stop. These commands are sent to a robot by the control system. Commands are in order due to selected solution received from the student routine. More about a robot navigation and control can be found in (Škrabánek, et al., 2016).

Steps which will make a clash with walls are not checked by the control system. The reason for this is a demonstration of a student's solution performance which might be incorrect.

A robot can use only one set of solution steps in any moment. Other solutions are invalid after the first step, because the original problem is changed.

1.2.4. High-level control system

The high-level control system consists of computer software and a hardware platform. The system has a responsibility for all that is done in this laboratory work. The control system must handle a few basic tasks which have been determined.

Maze detection is the first one of these tasks. The control system must detect the maze and convert it to a graph representation. How the conversion is done, is described in the next sections.

The second task is to display the problem for the student. Control application is divided to two panels. In the first one is a graphical representation of the detected maze with actual configuration (robot position and walls) and on the second panel, operation buttons are shown.

The next task is robot observing. This part of control system cares about robot position in the maze and updates graphic maze representation. This part of the control system must process every real robot rotation and move.

The last task is a communication with student routines. This is needed because one of requirements is implementation of any Path-planning algorithm. Socket communication was chosen, because it seems to be simplest for implementation of other algorithms. This system must provide information to client applications about a problem configuration over a defined text protocol.

Not one Path-planning algorithm is implemented in control system software.

Bc. Filip Majenk - Control System Prototype

File Action Show Log Client List

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120
121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140
141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180
181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200

Figure 3 – Control System Prototype

2 The Path-planning

Path-planning is the process of searching for a path from one position to the target position with knowledge about the maze. It is assumed that both positions are known and both of these positions are in the maze. It means that the robot must be able to achieve target position from start position. (Kanniah, et al., 2013)

2.1 Short introduction to path-planning

Path-planning is a relatively common problem in many computer systems which have to solve some decision problem. (Russell, et al., 2010) This problem can be almost anything. In computer space we are able to talk about problems shown on a graph. (Kanniah, et al., 2013) There is a very easy solution how the problem can be transformed to a graph. After transforming, the search algorithm starts to find the path to solve the problem. Path-planning is one type of graph problem for a robot in maze. The graph is a discrete map of the problem.

The result of the path-planning algorithm is a sequence of steps (Kanniah, et al., 2013), which is needed to be done to achieve the goal. A successful conclusion of the problem is when all steps are finished. In this case, the optimal path that satisfies the criteria defined by the task or defined by the problem, can be discussed. (Kanniah, et al., 2013)

However, not all path problem solvers work in the same way. Some solvers search only for a part of path and other look at the complete path. Both types have some advantages and disadvantages. For example, partial solvers are faster than other solvers, but they do not always warrant finding of the shortest path. On the other hand, complete path solvers are able to search for the shortest path that satisfies a given criterion. But such a search process can cost so much time and memory. Memory and time cost depends on problem space size. And this cost can grow awfully fast with only one small change in space or with one small resize of space. (Kanniah, et al., 2013)

2.2 Search for path-plan

The basic Path-planning problem is defined by knowledge about space, where a path is needed to be found. The basic knowledge can be, for example, a space with obstacles. There is a defined problem to find the path with no collision with obstacles while the sequence of steps are being realized. The process of looking for a sequence of steps that reaches the goal is called a search. (Russell, et al., 2010)

Searching for a path-plan can be achieved by different planning methods. The two most popular methods are uninformed methods, informed methods. All of these methods can work with cell decomposed space and a search path plan in it. And also they work with graph representation of physical space. (Siegwart, et al., 2004)

The goal formulation must exist before the path-planning search starts. Goal formulation is achieved by current situation - for example it can be based on user decision

about the target position of the robot. When a goal exists, problem formulation can be started. Problem formulation is the process of deciding which actions are needed to reach the goal.

Driver path-plan problem

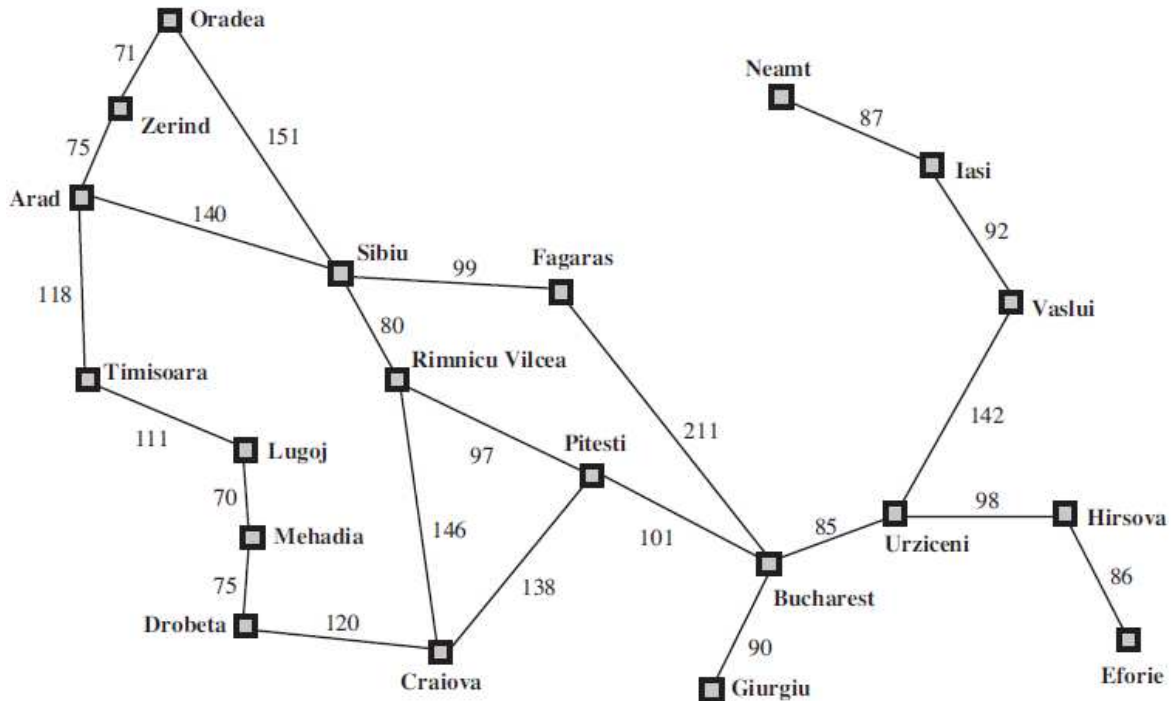


Figure 4 - Simplified graph of Romania (Russell, et al., 2010)

In Fig. Figure 4 a graph representation of Romania roads between towns is shown. In this graph, a path-plan from one town to another town can be searched for. Path-planning strategy is dependent on the problem goal, defined criteria and selected method.

The criteria are conditions which must be successfully achieved by the Path-plan. One of most used criterion is the shortest length of path. In another case for example, fastest path can be found, but all information to reach this criterion must be given in the graph.

Toy Problem – 8 puzzle

8-puzzle is one type of childhood logical toys. But it's also one common example for Path-planning. On this toy exists one blank space and eight numbered tiles. The dimension of the space is 3×3. The goal state can be achieved by moving tile to a blank space. Every move can be called as step. And all steps together in sequence from first to last is called a path-plan. (Russell, et al., 2010)

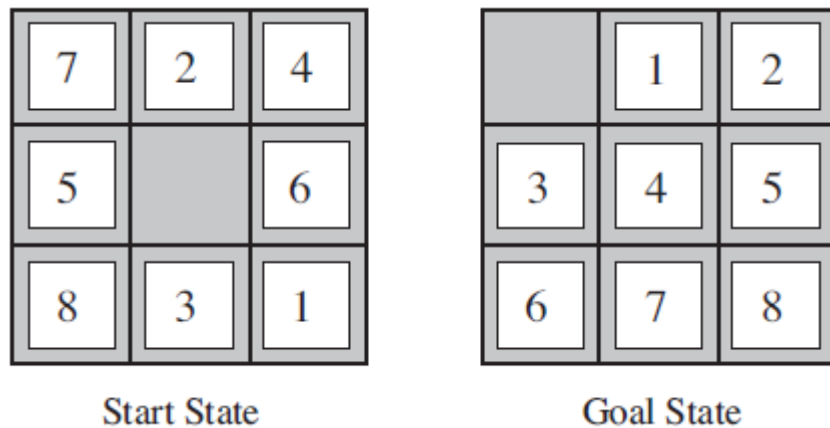


Figure 5 – Examples of possible start and goal state representations in the 8-puzzle (Russell, et al., 2010)

Standard problem formulation for the 8-puzzle is:

- **State (Node):** One of the possible states of puzzle configuration. The state is determined by numbered tile positions and position of the blank place. There are $9! = 362,880$ states but only $9!/2 = 181,440$ are reachable from the initial state.
- **Initial state:** Any puzzle state can be marked as the initial state. It is a start state for searching a Path-plan.
- **Actions (Steps):** In this puzzle exists 4 possible actions – slide Left, Right, Up and Down. A subset of possible actions are defined by the blank position. Action Left cannot be done while the blank is at the rightmost column. Top action cannot be done while the blank is at bottom row and so on.
- **Goal state:** Any admissible states can be marked as the goal state.
- **Action cost:** Every action cost is 1.

For this toy exists one rule which need to be respected. No one tile can be moved in any other way except sliding to blank place.

2.2.1. Uninformed methods

Uninformed methods are methods which have no information about the problem other than its definition. Thanks to that, they are often called blind methods. „All they can do is generate a successor and distinguish a goal state from a non-goal state.” (Russell, et al., 2010) Methods are distinguished by the order in which nodes are expanded. The Path-plan is constructed from a sequence of states. Sequence of states are in an order which need to be visited to achieve a goal state. (Kanniah, et al., 2013)

We can set criteria to evaluate an algorithm performance. To evaluate criteria these questions must be answered:

- **Completeness** – If one or more solutions exist, is there a guarantee to find them?
- **Optimality** – If a solution is found will it be the optimal⁴ one?
- **Time complexity** – How long does it take to find a solution? (Usually measured in count of expanded nodes.)
- **Space complexity** – How much memory is needed to perform the search?

Breadth-first search

A breadth first search (BFS) is an algorithm for traverse or search on a tree or a graph data structure. The strategy for searching in a structure starts from the initial state of the problem. Simply all nodes are expanded at actual depth in the search structure before any nodes from the next level are expanded. (Russell, et al., 2010)

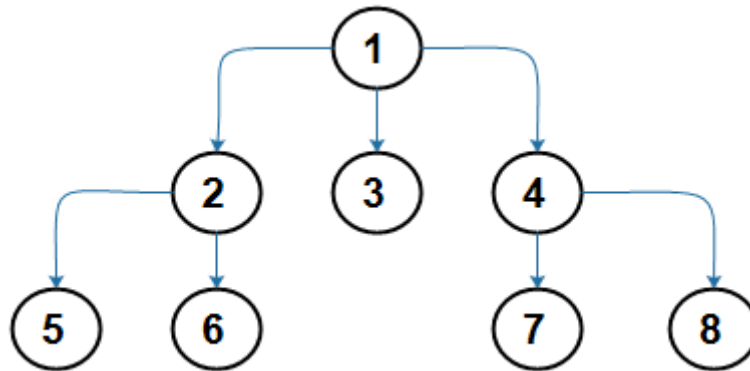


Figure 6 - Order which the nodes are expanded by BFS

Breadth first search works all the time with the shallowest unexpanded node in a chosen expansion. This is guaranteed by use of a FIFO queue collection. Thus, new nodes are inserted to the end of the queue. This algorithm works with one more collection. The second collection is used for explored nodes.

In every step the front node is taken from the FIFO queue. This node is compared with the goal node to determine if the goal is achieved or not. If not, this node is stored to expanded collection, and after it is expanded, all its unexplored child nodes are inserted to the end of FIFO queue.

The algorithm can end in two different situations. The first one is to reach a goal state when the algorithm return path contains all parents from last explored node to root. The second situation can occur when all nodes are expanded and processed, and searched the target node does not occur in the structure.

⁴ Optimal solution mean that solution has the lowest path cost among all solutions. (Sedgewick, et al., 2011)

```

begin doBreathFristSearch( Node root, Node goal):
    create FIFO as fifo, Collection as explored

    root.parent = NULL, fifo.insert(root)
    while fifo is not empty
        actual = fifo.first()
        explored.add(actual)
        if actual == goal return actual;
        else
            foreach Node child of actual
                if child not in explored
                    child.parent = actual
                    fifo.insert(child)

```

Figure 7 - Breadth-first algorithm pseudocode (Sedgewick, et al., 2011)

Figure 7 shows pseudocode of a BFS algorithm. In a short, this algorithm does traverse from root (initial state) and search for a goal (target state). During the search process, the FIFO queue is used. To this queue is new node is inserted by insert() method and the first node (the oldest element) is removed by the first() method. Variable child is a successor of the actual node. For every child node, their parent is set. Each iteration is checked for a goal state. If a goal state is found the actual node is returned.

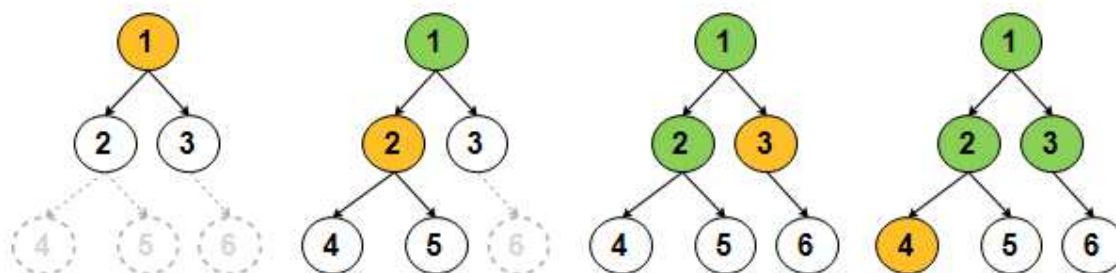


Figure 8 - Breadth-first search on a simple binary tree, partial of the solution is indicated from left to right

Fig. Figure 8 shows a simplified working algorithm. The actual node is represented by orange color and explored states are green. Unpainted nodes are in the FIFO queue and wait for exploration. Nodes indicated by a dotted line are nodes which are not now available. These nodes are waiting for exploring of their parents to be queued to FIFO. The next step will only occur if their parent is not a goal state.

The basic problem of BFS algorithm is memory requirements. Memory requirements grow with exponential-complexity in every subsequent depth level. Problems with more than 16 depths are not solvable in real life on a personal computer. According to this knowledge can be said BFS is usable for every problem, but only if a problem has minimal states. (Siegwart, et al., 2004)

Depth-first search

Another representative of the uninformed method is the depth-first search (DFS). This method is also for search and traverse on a tree or in a graph structure such as BFS. The difference from BFS is primarily in a used queue. BFS uses a FIFO queue and DFS uses LIFO. The strategy of BFS is based on work with the deepest node where no successors exist. (Even, 2011)

The basic version of this algorithm can fail on the graph structure, because there exists loops between nodes. (Cormen, 2009) This problem can be solved by additional extra memory collection to avoid these loops.

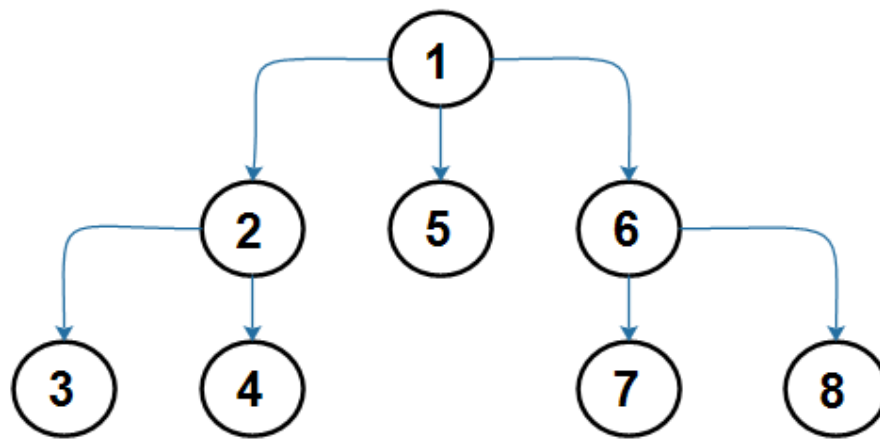


Figure 9 – Order which the nodes are expanded by DFS

This method uses two collections. The first one is an open LIFO queue collection where unexplored nodes are stored. The second closed collection has stored explored nodes. Every step acquires the front element from the LIFO (it mean last inserted node) queue. The node is expanded only if this actual node is not a goal state. All successors of this node are inserted to an open queue if they are not in the closed queue. (Even, 2011) (Siegwart, et al., 2004)

A basic DFS algorithm can finish in three situations. The first one is a successful search for a goal state which returns a Path-plan for it. The second one is full expansion of all states and no goal state found. And the last one is a "dead cycle" between two states. When DFS uses a closed queue the last situation is prevented from appearing. (Even, 2011)

In Figure 10, the pseudocode of DFS is shown. In short, this algorithm traverses from a root node (initial state) to a goal (target state). Firstly, two collections (one is an open LIFO queue and second one is a closed collection), with an ability to check contained states, are created. After collections are created, the root node is inserted into FIFO by a method insert(). While the LIFO queue is not empty, the loop is repeated. In each iteration of the loop, the first node (last inserted) in the LIFO queue is moved from this queue to the closed queue, and compared with the goal state. If actual node is a goal, the algorithm will return

this actual node. From the returned node, a path to root is calculated using parent relations. Otherwise, each successor is added to the open queue, only if this successor is not contained in a closed collection.

```

begin doDepthFirstSearch( Node root, Node goal):
    create LIFO as open, Collection as closed

    root.parent = NULL, open.insert(root)
    while open is not empty
        actual = open.first()
        closed.add(actual)
        if actual == goal return actual;
        foreach Node successor of actual
            if successor not in closed
                open.insert(successor)

```

Figure 10 - Depth-first algorithm pseudocode (Sedgewick, et al., 2011)

In Fig. Figure 11 a simplified working algorithm is shown. The actual node is represented by orange color and explored states are green. Unpainted nodes are in the LIFO queue and wait for exploration. Nodes indicated by a dotted line are nodes which are not now available. These nodes are waiting for exploration of their predecessor to be queued in LIFO. The next steps only occur if their predecessors are not goal state.

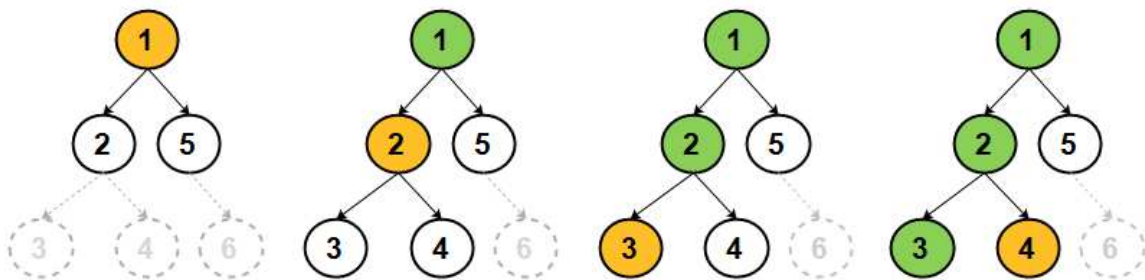


Figure 11 - Depth-first search on a simple binary tree, partial solution is indicated from left to right

The main advantage of the DFS algorithm is memory requirements, which are diametrically different from the breadth-first algorithm. The primary disadvantage of the depth-first search algorithm is that, it is not guaranteed to find the solution and if the solution is found, it might not be the optimal solution. (Even, 2011)

The depth-first search has got a modification with limit for maximum depth. This modification is based on criteria in which level is the search stopped. This variant can be again modified with iteration, when every iteration increases maximal depth. This modification is called Depth-first iterative deepening, and it finishes when goal state is achieved or when maximal allowed depth is reached. (Even, 2011)


```

begin doIterativeDepthFirstSearch( int maximalDepth, Node root, Node goal ):
    int depth = 1
    while (depth <= maximalDepth):
        if(goal == (returnedGoal = doDepthFirstSearch(root, goal, maximalDepth)))
            return returnedGoal
        else:
            depth++

```

Figure 12 - Depth-first iterative deeping pseudocode (Sedgewick, et al., 2011)

This pseudocode expects implementation of the Depth-first iterative search which ends when maximal level is reached. A search with every new maximal depth is done again from the root and all queues are cleared. When the goal is reached, the algorithm does not continue to the next depth level.

“In general, iterative deepening is the preferred uninformed search method when the search space is large and the depth of the solution is not known.” (Russell, et al., 2010)

Bi-directional search

The main idea of bi-directional search (BIS) is to start from both sides simultaneously. One part searches for a path from the initial state to goal, and the second part searches from the goal to initial state. (Cormen, 2009) When both search parts meet in some node – the path from initial state to goal is achieved. In bidirectional search combination of DFS and BFS algorithm can be used. (Russell, et al., 2010)

Solution of this algorithm may not be optimal. Additional research is needed to test if any shortcuts in structure exist. (Even, 2011) The second problem, which makes this algorithm weak, is a high space complexity. (Cormen, 2009)

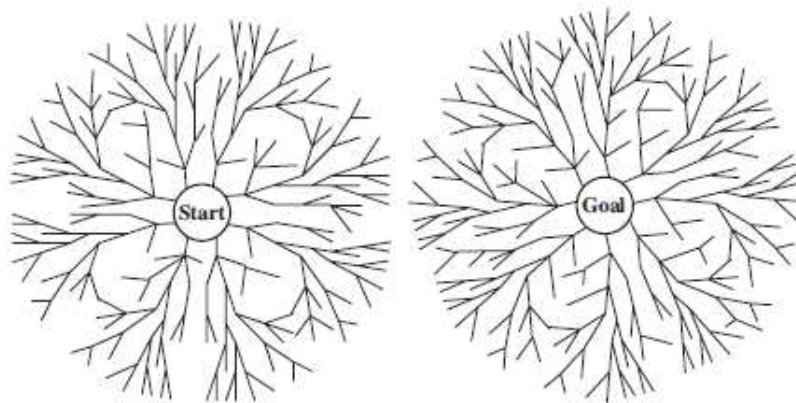


Figure 13 - A schematic view of a bidirectional search

2.2.2. Uninformed methods comparison

Before comparison of these algorithms can be started. Some variables, which are needed to be known, must be defined:

- branching factor b – maximum number of successors of any node, (Russell, et al., 2010)
- depth d – maximal depth from root where can be found goal state, (Even, 2011)
- maximum length m – maximum length of any path in the space, (Russell, et al., 2010)
- depth limit l – maximal depth – used only for Depth limited and Iterative Deeping algorithms.

Criterion	Breadth-First	Depth-First	Depth limited	Iterative Deepening	Bidirectional
Complete?	Yes	No	No	Yes	Yes
Time	$O(b^d)$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(bm)$	$O(b^l)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes	No	No	Yes	Yes

Table 1 - Evaluation of tree-search versions of algorithm (Even, 2011) (Cormen, 2009) (Sedgewick, et al., 2011)

Every complete algorithm must have a finite b . The bidirectional method must moreover use, in either direction, the breadth-first search algorithm, or a combination of BFS and iterative deepening search algorithms. BFS, Iterative Deepening and Bidirectional algorithms are optimal only if step costs are all identical. (Russell, et al., 2010)

Big O notation

Big O notation is used to classify algorithms according to how long it takes to find a solution or how much space is needed according to the input problems (Mohr, 2007).

The most common notations are $O(1)$ – constant; $O(\log n)$ – logarithmic; $O(n)$ – linear; $O(n \log n)$ – log-linear; $O(n^2)$ – quadratic; $O(c^n)$ – exponential and $O(n!)$ – factorial.

Common notations are sorted from slowest growth to fastest growth. That means that the $O(1)$ has best performance and $O(n!)$ is the worst (Mohr, 2007).

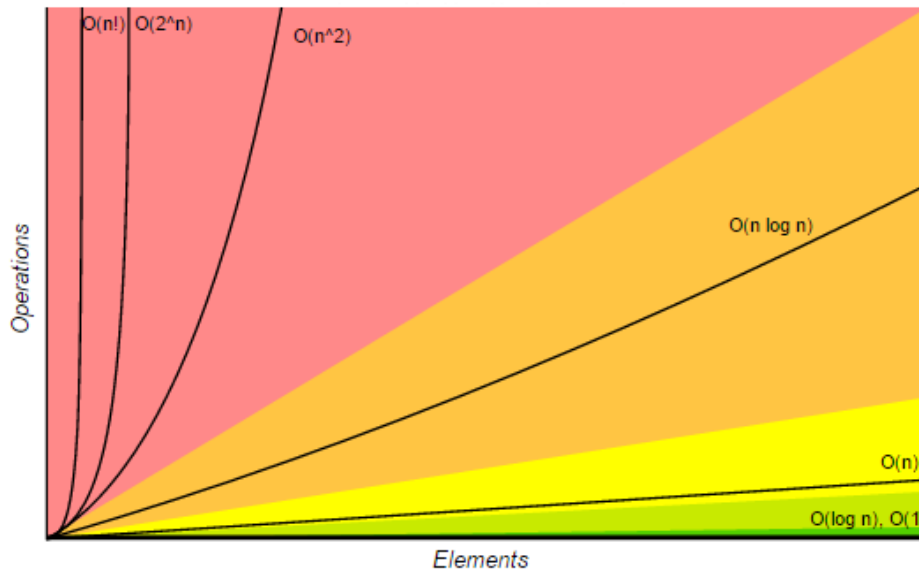


Figure 14 - Big-O Complexity Chart⁵

2.2.3. Informed methods

Informed methods are strategies that know whatever non-goal state is "more promising". Informed methods use a heuristic function which gives weight for next steps. All informed methods use it to decide which next node is the best. Thanks to a heuristic function method, which works with problem-specific knowledge, solutions can more efficiently be found compared to uninformed methods. (Cormen, 2009) (Russell, et al., 2010)

These methods are based, besides the heuristic function $h(n)$, on an evaluation function $g(n)$. The evaluation function is constructed as a cost estimate and helps to select the node which will be evaluated in the next step. If condition $h(n) = 0$ is accomplished, the n is the goal state. (Russell, et al., 2010) The evaluation function returns a number describing the desirability to expand the node. By this number, the algorithm decides which node is the best successor.

Greedy search – Greedy best-first search

Greedy search strategy works in all nodes only with local values of heuristic function $g(n) = f(n)$. In every step, the greedy search expands the node with the biggest value. Due to that, the optimal solution for the problem is not guaranteed. (Even, 2011) The optimal one can be in another branch of the structure. This shows why the algorithm is called "greedy" - at each step, it tries to get as close to the goal as it can. (Russell, et al., 2010)

A greedy search on tree is also incomplete in the finite state space. This appears when the algorithm expands a bad node which has no goal state in its successors. This situation is called a dead end, because a goal state can never be reached.

⁵ Source: <http://bigocheatsheet.com>, Visited: 26. March 2017

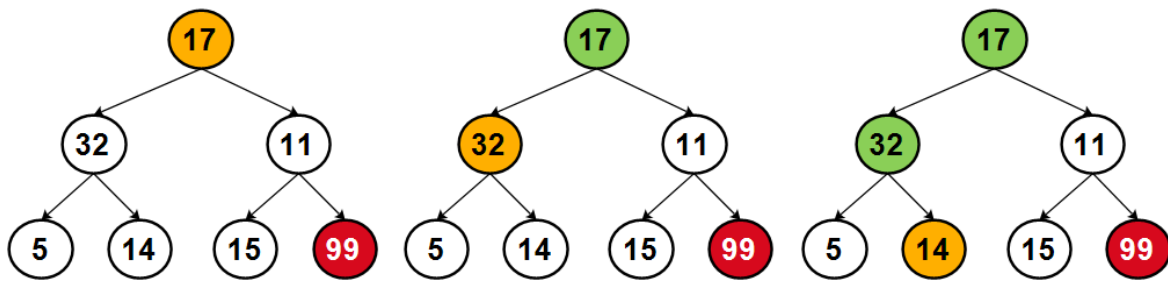


Figure 15 - Greedy search with dead end problem

Fig. Figure 15 - Greedy search with dead end problem shows poor evaluation of a tree with dead end and without reaching the goal state. The expected goal state is painted by red color. Unfortunately, the bad successor 32 was selected in the first step, which causes entry to a dead way. Orange color is actually an expanded node and green are expanded nodes in path.

In the same tree as Fig. Figure 15 - Greedy search with dead end problem, we can define another problem. The problem is based on finding the biggest sum of numbers in the nodes. There will be the same problem because greedy search picks in first expanding iteration, node 32 and never gets the correct result. In this case, the correct result is the sum of $17 + 11 + 99 = 127$, but result $17 + 32 + 14 = 63$ is incorrectly selected. (Hazewinkel, 1995)

The greedy search method is used to construct a tree to find an optimal solution in the Huffman coding tree. (Huffman, 2007)

```

begin doGreedySearch( Node root, Node goal, Functor heuristicFunction )
    if ( root == goal ) return root;
    else if root.getSuccessors() is empty return NULL;

    create Set feasibleSuccessors
    feasibleSuccessors.put( root.getSuccessors() )
    feasibleSuccessors.calculateAndSortByHeuristicFunction( heuristicFunction )

    Node successor = feasibleSuccessors.getFirstByPriority()
    successor.setParent( root )

    return doGreedySearch( successor, goal, heuristicFunction )

```

Figure 16 - Greedy search recursive algorithm pseudocode

This pseudocode is based on recursive algorithm and a priority queue. If the root node entered to the function is a goal state, the node will be returned. Otherwise, the algorithm checks the existence of node successors. If the node has no successor, the method returns NULL. If the node has successors, the priority queue is used to get the most feasible successor using the heuristic function (the nodes sorted according to h). It means that the first node in the priority queue is the feasible successor. Everything is done again for the successor. The algorithm ends when no successor exists or a goal state is reached.

A* (A star) algorithm

A* algorithm is widely used in many systems to solve cost based problems. This algorithm guarantees to find the cheapest path from initial state to goal - if a good heuristic function is supplied. To reach a goal, a simple formula is used (Russell, et al., 2010)

$$f(n) = g(n) + h(n). \quad (1)$$

This equation combines cost to reach actual node $g(n)$ and cost to get from the actual node to goal node. The result of this equation is an estimated cost $f(n)$ of the cheapest path from initial state to reach goal state. The algorithm is complete and optimal, if heuristic function $h(n)$ satisfies certain conditions. Optimality of the A* algorithm is achieved, if heuristic function $h(n)$ is admissible, if graph-search version is optimal, if $h(n)$ is consistent, and if $h(n)$ is consistent. Then values of $f(n)$ along any path are non-decreasing. (Cormen, 2009)

Let us suppose that the node n' is a successor of the node n then

$$g(n') = g(n) + c(n, a, n'), \quad (2)$$

where a is an action. It must hold for these two nodes that (Russell, et al., 2010)

$$f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') \geq g(n) + h(n) = f(n). \quad (3)$$

While evaluating the A* algorithm two conditions are needed to be checked, before a successor is inserted into the open list for future expansion. When popping the best successor, the open collection needs to be sorted by value of calculated $f(n)$.

```
begin doAStarSearch( Node root, Node goal, Functor heuristicFunction)
    create Collection as open, Collection as closed

    open.add(root)
    while open is not empty
        Node bestSuccessor = open.sortAndPopBestSuccessor()
        foreach Node successor of bestSuccessor.getSuccessors()
            successor.setParent( bestSuccessor )
            if ( successor == goal ) return successor;

            successor.setG( bestSuccessor.g + bestSuccessor.getCostTo(successor) )
            successor.setH( heuristicFunction( successor, goal ) )
            successor.setF( successor.getG() + successor.getH() )

            if successor.getF() > open.getLowestFOfNodes() continue;
            else if successor.getF() > closed.getLowestFOfNodes() continue;
            else
                open.add( successor )
                closed.add ( bestSuccessor )
```

Figure 17 - A* (A Start) algorithm pseudocode

Figure 17 shows the pseudocode of A* algorithm. During the process two collections are created. One collection is for open nodes (which are not expanded yet) and closed collection for expanded nodes is the second one. At the start the root is added to open

collection. This is done because the algorithm works, while open collection is not empty. In every iteration the best node is selected from the open collection by `sortAndPopBestSuccessor()` method. Then every successor is compared with goal state and if it is not a goal state, the metrics are calculated by heuristic function and compared with other nodes in closed and opened collections. If its metric is still the lowest one, the node is added to open collection. Every node is added to closed collection.

This algorithm is used most commonly, because it is very fast and optimal – if a good heuristic function is chosen. Very low memory is needed because only necessary states are expanded. We can say – this algorithm is best one. But the overall performance and speed is dependent on the selected heuristic function which can be slowest part of the algorithm.

2.2.4. Heuristic function

In general, heuristic function is part of an algorithm where additional knowledge about the problem is imparted to the search algorithm. There exists only one constraint: a goal node has $h(n) = 0$. A Heuristic function can be called “intelligence” for informed path-planning methods. (Russell, et al., 2010)

The main task of this function $h(n)$ is to return a non-negative real number which represents an estimate cost from the actual node n to a goal node. The Heuristic function $h(n)$ can not underestimate the path from the actual node to next node. In every step the heuristic function must return a lower path-cost than the actual cost is. (Even, 2011)

This function is used to determine which step is the best in the next action. Almost every path-planning algorithm chooses the lowest cost to determine the next state - because there is an assumption about the lowest cost for generated path. If the heuristic function is a good one, the lowest number is nearest to the goal state. (Russell, et al., 2010)

The two most common used heuristic functions are Euclidean distance and Manhattan distance. The value of both of these distances decrease in every step.

Euclidean distance

The Euclidean distance ϱ is defined as straight-line distance between two points in Euclidean space. Distance ϱ between two points A and B is calculated with Pythagoras theorem (Sedgewick, et al., 2011):

$$\varrho(A, B) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}, \quad (4)$$

where x_1, y_1 are coordinates of point A and x_2, y_2 are coordinates of point B. The result of this formula is equal to the distance between place A and place B. The real distance between two real points is shown in Fig. Figure 18 - Euclidean distance.

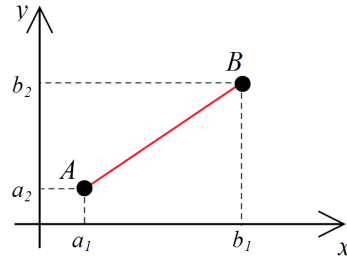


Figure 18 - Euclidean distance

Manhattan distance

Manhattan distance is usable where diagonal moves are restricted. A good example is moving in a city, where going across block of houses is not allowed or 8-puzzle where only horizontal and vertical moves are allowed (Sedgewick, et al., 2011). Distance ϱ between two places A and B is calculated by simple formula:

$$\varrho(A, B) = |x_2 - x_1| + |y_2 - y_1| \quad (5)$$

where x_1, y_1 are coordinates of point A and x_2, y_2 are coordinates of point B.

The result of this formula is equal to the sum of absolute differences of x and y coordinates of places A and B. The real result between two points is calculated by this formula and shown in Figure 19.

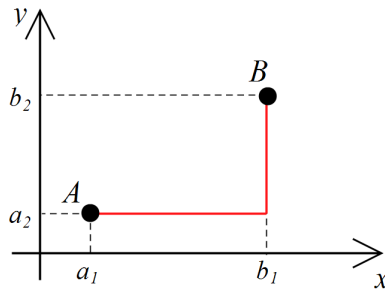


Figure 19 - Manhattan distance

2.3 Configuration space and decomposition

The configuration space is a type of discrete representation of the real world. In this configuration all important information is stored, which is needed for use in path-planning algorithms. In our case 2D space configuration is used. Every state or position of a robot is described by Cartesian coordinates (x_R, y_R) in configuration space C and rotation angle Θ_R . (Škrabánek, et al., 2015)

For a fully specified space configuration we need to be able to recognize where obstacles are in the workspace W . For this case, we need to map the obstacles in configuration space and after this, take into consideration the subset of configuration space C , that makes contact-free configurations. (Latombe, 1991)

Every obstacle B_i , $i = 1$ to q , in the workspace W is mapped in a region C :

$$CB_i = \{q \in C/A(q) \cap B_i \neq \emptyset\} \quad (6)$$

which is called a C-obstacle. The union of all the C-obstacles

$$\bigcup_{i=1}^q CB_i \quad (7)$$

is called the C-obstacle region, and the set

$$C_{free} = C \setminus \bigcup_{i=1}^q CB_i = \left\{ q \in \frac{C}{A(q)} \cap \left(\bigcup_{i=1}^q B_i \right) = \emptyset \right\} \quad (8)$$

is called the free space. (Latombe, 1991)

Space decomposition is a process where the real world is converted to a network of cells. These cells are the potential position of the robot. With this network an adjacency matrix can be constructed. With this matrix neighboring cells are specified. This information is used by a search algorithm to find the path-plan from the initial position to target position. In an adjacency matrix only binary values are stored. One (true) for open path and zero (false) is the unreachable path to the next cell. Count of neighboring cells is not restricted because it depends on the selected type of cell decomposition.

Exact cell decomposition

This methods achieves the decomposition by selecting boundaries between discrete cells based on geometric corners. The free space is externally bounded by a polygon and divided by vertical lines. Vertical lines divide the free space to cells which can be used for path-planning (See Figure 20). The second step is to construct the graph which represents structure - in most cases, adjacency matrix. (Latombe, 1991)

These representations can be compact because each such area is stored as a single node, because it is not dependent on size cell and cell position. There are only two necessary pieces of information - initial cell and target cell. (Siegwart, et al., 2004)

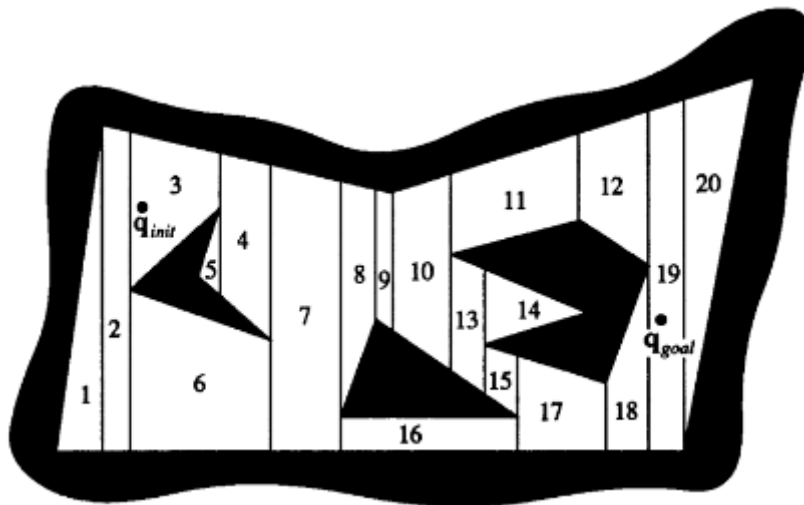


Figure 20 - Example of exact cell decomposition (Latombe, 1991)

The next figure shows how the exact cell decomposition process is transformed to a graph representation. This transformation is done on decomposition space from Figure 20.

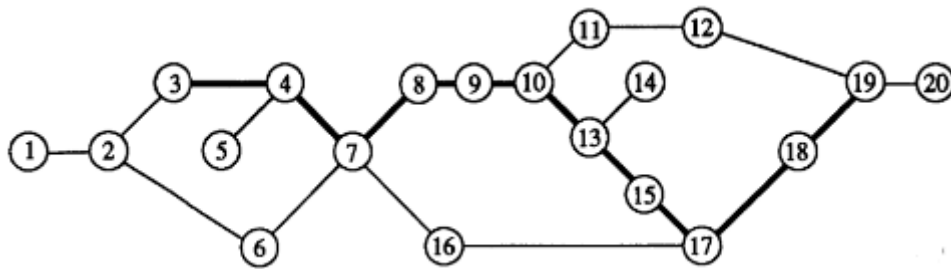


Figure 21 - Adjacency relations between the cells (Latombe, 1991)

Fixed cell decomposition

An alternative to exact cell decomposition is a fixed cell decomposition. This method is based on partitioning a space to equally large cells, in most cases as squares. For this method, there exists one big disadvantage why it is not often used. The disadvantage is that obstacles are not only rectangles, and some cells are marked as an obstacle, but only a small piece from the obstacle is on it. Because of this inaccuracy, an almost empty cell can be identified as an obstacle, and with this inaccuracy can be marked as closed path - which can be open in real space. (Latombe, 1991) (Siegwart, et al., 2004)

There is a recommendation to use the smallest fixed size of squares for elimination of this disadvantage. Fixed cell decomposition with this problem is shown in Figure 22 (right). In the same Figure on the left side, is shown original problem decomposition without marking the cell as an obstacle (black square). (Siegwart, et al., 2004) At first sight, the problem is seen with moving between central and right obstacles after apply obstacle markers. In the result, adjacent structures are marked as a closed path.

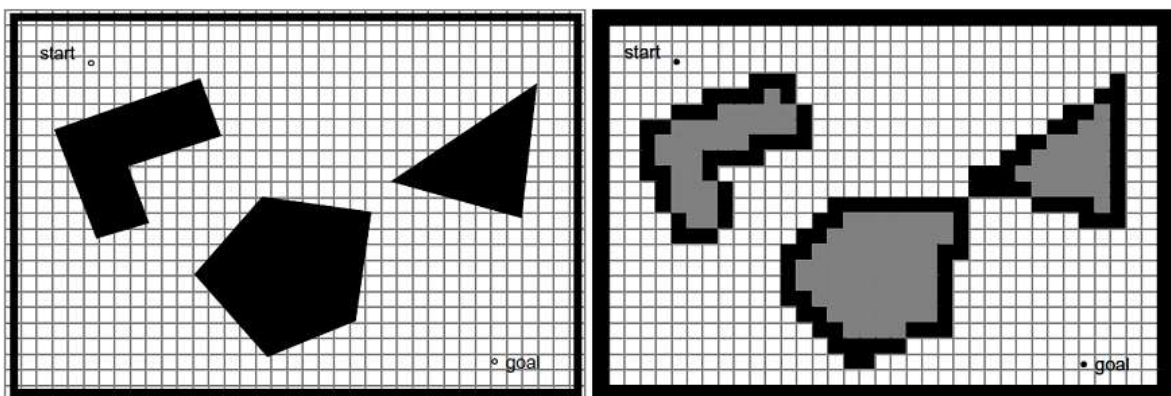


Figure 22 - Fixed cell decomposition problem

Adaptive (variable) cell decomposition

A modified version of fixed cell decomposition is adaptive cell decomposition. This method is improved by gradual decomposition of a bigger cell where obstacles are divided into a

smaller cell. This improvement provides better accuracy for marked obstacles in space and removes the disadvantage of fixed cell decomposition. (Latombe, 1991)

Decomposition by this method is done in these steps:

1. divide full working space to a four big equal sized rectangles,
2. pick all rectangles where is piece of obstacle found,
3. divide all rectangles contains obstacle again to four equal sized rectangles.

This process is done while all obstacles are bordered, or when the maximum level of divide factor is reached. The dividing factor depends on user selection and corresponds to the count of iteration for dividing rectangles. In the last dividing step, obstacles are definitely marked. (Siegwart, et al., 2004)

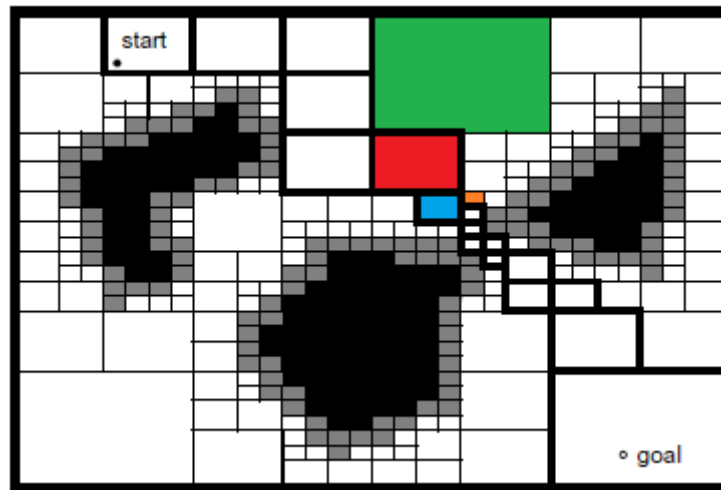


Figure 23 - Workspace decomposed by adaptive cell decomposition method (Siegwart, et al., 2004)

In Figure 23 the decomposed space is shown with divide factor 4. The resulting rectangle sizes for individual steps during the decomposition is also shown. For clarity only one colored rectangle is shown from every step. But in result all equal sized rectangles are generated in one iteration of dividing.

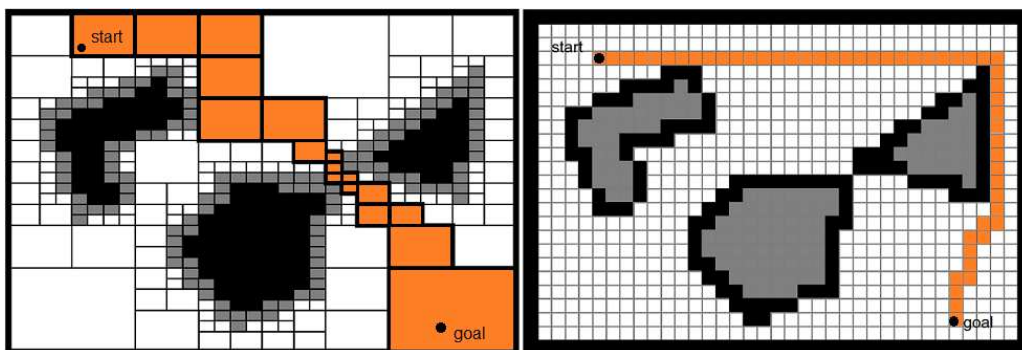


Figure 24 - Comparison of resulted path-plan for these two methods.⁶

⁶ Based by figures from (Siegwart, et al., 2004)

Figure 24 shows the difference between resulted path-plan after adaptive cell decomposition (left) and after fixed cell decomposition (right). On first sight, the path on right is longer thanks to above mentioned disadvantage and the states need to be expanded further.

3 Image Processing

Image processing is a method for image enhancement with algorithms and operations. The output of these is an image ready for analysis. Object detection in the image is one of most common used operations. But for this operation and many other operations, basic knowledge about images and digital image representation is needed. Almost all operations are based on these steps:

- Get image (from camera, video stream or simple computer drawing).
- Store the image in an appropriate format.
- Image preprocessing (white balance, noise reduction, restoration, compression...). This step is based on requirements by final operation.
- Image analysis (segmentation, image registration, matching and comparison).
- Image post processing - perform the requested operation.

The purposes of image processing can be divided into these five groups (Monga, et al., 2015):

- Visualization – transforming digital data into images representing information about the data.
- Image sharpening and restoration – repairing corrupt/noisy image and estimating clean.
- Image retrieval - looking for the image of interest.
- Measurement of pattern - measures various objects in an image.
- Image Recognition – distinguish the objects in an image.

3.1 Digital image representation

There exists two main image representations: vector data model and raster (bitmap) data model. Whereas the raster model is based on graphic pixels, the vector data model is based on polygons, lines and points. These models have some advantages and some disadvantages. (Rafael, et al., 2008)

Raster data model

The raster data model is based on a pixel grid (Figure 25), where each pixel⁷ [px] has one color from a color model. The two most common color models are the RGB model which is designed for adaptive color mixing (in general, for lighted equipment - for example monitors) and CMYK which is designed for places where subtractive mixing of colors is needed (good example are printers). In RGB model every pixel on the raster grid has three basic components – Red, Green and Blue component. In image processing the grayscale color model (only shades of black) or binary color model is often used. (Sonka, et al., 2008)

⁷ Pixel is an image element.

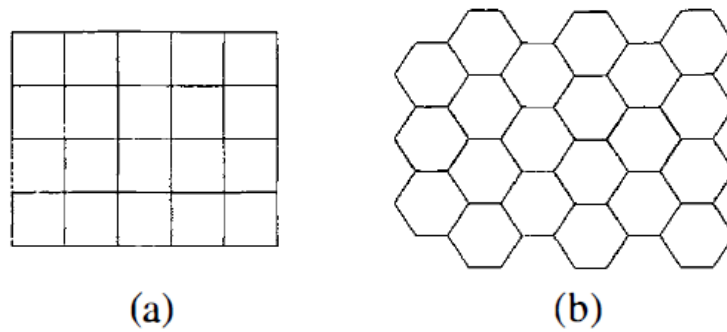


Figure 25 - Most used bitmap grids – (a) Square grid. (b) Hexagonal grid.

The final color count is based on selected color depth. Generally these color depths are used:

- 1-bit – monochrome (2 colors),
- 8-bit – color/grayscale (256 colors or 256 shades of black),
- 16-bit – high color (32,768 colors with transparency or 65,536 colors without transparency),
- 24-bit – true color (16,777,216 colors),
- 32-bit – deep color (4,294,967,296 colors).

Color count is calculated from an easy formula, where n is bit depth (Rafael, et al., 2008):

$$X_c = 2^n. \quad (9)$$

The main advantage of raster data model usage, is the possibility of having more details (shadow detail) in the picture than in the vector data model. But the disadvantage is therefore with higher memory requirements.

General formula for calculating image size in Bytes S_B (Rafael, et al., 2008):

$$S_B = w * h * \frac{c}{8}. \quad (9)$$

where w is width in pixels, h is height in pixels, and b is color bit color depth (8-bit,...).

Due to this disadvantage a compression for saving images is introduced. One version of a compression is loss and lossless. (Sonka, et al., 2008) At present, the ordinary resolution is about $3,500 \times 5,000$ pixels for DSLR cameras (about 17,5Mpx - \pm 52,5MB without compression).

The lossless compression is based on a conversion to the shortest sequence of bytes and after decoding we get back the original file. Resultant compression rate is based on repeating the same sequence of bytes in file. This compression is used in tiff, png, and gif image file types. (Salomon, et al., 2009)

The loss compression is based on removing unimportant information from the image. Unimportant information is information which the human eye can not recognize. After

decoding the original image is not obtained, and some information is irretrievably lost. Amount of lost information can be chosen by the level of compression. In this case, every new file save must be counted with repeatable compression. This algorithm is used in JPEG image file types. (Sonka, et al., 2008) (Rafael, et al., 2008)

The final quality of an image on the raster data model, depends on chosen grid size (resolution), color depth, and compression. Figure 26 shows the difference between a 100×100px image resized to 400×400px, and the original 400×400px image. Both of these images are stored from original 1024×1024 image, selected color depth is 24 bits and JPEG compression level 80%. The original image is stored in PNG format with same color depth. The size of the original image is 2.2MB (lossless compression), image file size (a) on Figure 26 has 48kb and (b) is 1.7kb. The difference between them is seen on first sight – raster model is not usable for image resizing to a bigger resolution. (Rafael, et al., 2008)



Figure 26 - (a) 400 x 400px, (b) 100 x 100px

Vector data model

Vector data model is based on a polygons, which is defined by points, lines and colors. Each point of every polygon has its own definite position on axes. Every line has assigned various attributes and includes stroke color, shape, curve, thickness, and fill.

The opposite of rasterize process is vectorization. In this process, the raster model image is transformed to vector model, but this operation is more difficult than rasterization and in most case cannot be done automatically by a computer. The possible result of vectorization is shown in Figure 27.



Figure 27 - Vectorized photo of Theodore Roosevelt⁸

Raster and vector final summary comparison

Table 2 summarizes the basic characteristic of these two image data models. This summarization is based on gained knowledge while writing the thesis.

Vector	Raster
Polygon based	Raster based
Better for logos, drawings and illustrations	Better for photos
Can be scaled without quality losing	Without quality losing can be only down scaled
Resolution-independent – for every resolution is small file size	Large resolution and higher detailed image is result of large file size
Easy conversion to raster	Vectorization is depending on file complexity – more complex = more time
Special software needed	In many systems is raster native format

Table 2 - Comparison of raster and vector model

3.2 Point operations

Point operations are focused to perform an operations at every pixel on an image. These operations are based on changing image size, colors, contrast, geometry, or local structures of the image. To every pixel is applied a sequence of required functions. Each new pixel value depends exclusively on its previous value at the same position and is independent from any other neighboring pixel values. The point operations are functions only for exactly one pixel - all operations are applied pixel-by-pixel. (Rafael, et al., 2008) (Sonka, et al., 2008)

⁸ Source: <http://vectorsquad.com/?p=945>

A simple formula for Point operations application, where is $P(x)$ required operation, $f(x,y)$ is an input pixel, and $G(x,y)$ is result pixel:

$$G(x, y) = P(f(x, y)) \quad (10)$$

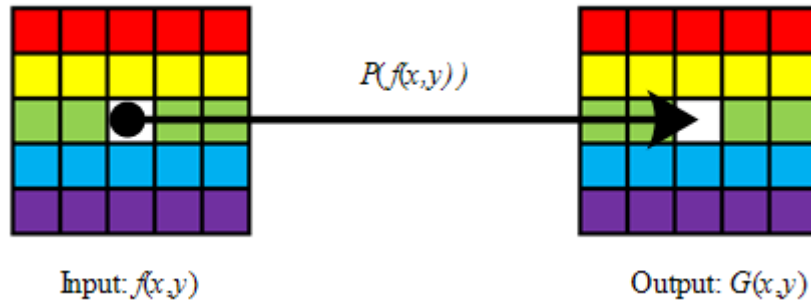


Figure 28 – Apply the function P to $f(x,y)$

3.2.1. Brightness transformations

Brightness transformations belong to a group of picture adjustment. This transformation is one of the basic point operations which is needed to be done before image analysis is performed. Brightness transformation is based on RGB conversion to grayscale image.

These transformations are divided to two classes: **brightness correction** and **gray-scale transformation**. Brightness correction takes into account its original brightness and its position in the image. Gray-scale transformations change brightness without regard to position. (Sonka, et al., 2008)

Brightness correction

If we want to do a brightness correction, we need to get a reference image, where we can get an error coefficient for each pixel. When the reference image exists, and error coefficient $e(x,y)$ can be counted from this image, we can apply brightness correction for each pixel on the input image as:

$$g(x, y) = e(x, y) * f(x, y) \quad (12)$$

But this method can be used only if the degradation process on the image is stable. The stable degradation means, if the degradation is a linear multiplication in the next steps - it is possible to correct the degradation by repeating this correction.

If we change the source of images, a new calibration of error coefficients $e(x,y)$ for each pixel is needed. This method has one problem - brightness value can overflow if the error coefficient is bigger than 1. In this case we need to crop the maximum value to exactly 255. The ideal image for brightness correction has an average value of 128 for brightness of all pixels. (Sonka, et al., 2008)

Gray-scale transformation

Gray-scale transformation is not based on a pixel position. A transformation P of the original brightness p from scale $\langle p_0, p_k \rangle$ into brightness q from a new scale $\langle q_0, q_k \rangle$ is given by:

$$q = P(p) \quad (13)$$

The most common gray-scale transformations are shown in Figure 29, where three main functions are shown. The input of these functions are grayscale images.

- Function (a) enhances the image contrast between brightness values p_1 and p_2 .
- Function (b) is called brightness thresholding and result is black-white image.
- Function (c) is negative transformation.

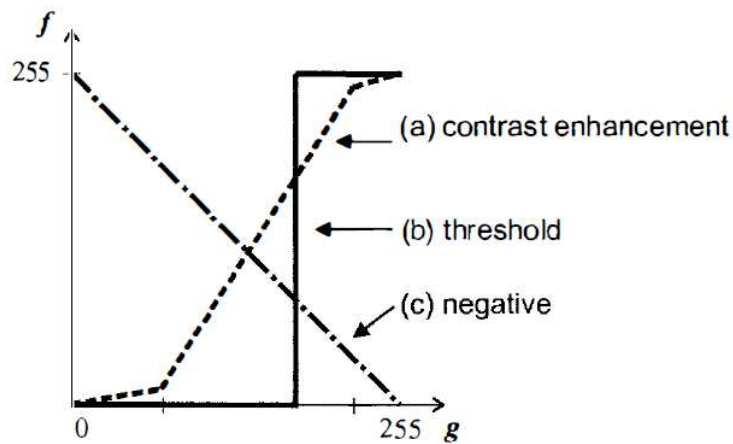


Figure 29 - Gray-scale transformations

All of these gray-scale transformations can be easily done in real-time on conventional desktop computers, or on commonly available mobile phones. (Říha, 2012) To achieve these operations, often only 256 bytes of memory are needed. (Sonka, et al., 2008) The original calculated brightness is the index to the look-up table, where the new brightness is taken. “These transformations are used mainly when an image is viewed by a human observer.” (Sonka, et al., 2008)

3.2.2. Histogram equalization

This technique is based on gray-scale transformation for a contrast enhancement. The aim is to create an image with equally distributed brightness levels over the whole brightness scale. When the brightness value is near maximum it needs to be decreased and conversely enhanced when brightness is near the minimum. Next, denote the input histogram by $H(p)$ and input gray-scale as $\langle p_0, p_k \rangle$. The intention is to find a monotonic pixel brightness transformation $q = P(p)$ such that the desired output histogram $G(q)$ is uniform over the whole output brightness scale. The histogram can be treated as a discrete probability density function. (Sonka, et al., 2008)

$$\sum_{i=0}^k G(q_i) = \sum_{i=0}^k H(p_i) \quad (14)$$

The sums in equation (14) can be interpreted as discrete distribution functions. Denote the rows and column count as N . Then the equalized histogram $G(q)$ corresponds to the uniform probability density function f whose function value is a constant (Sonka, et al., 2008):

$$f = \frac{N^2}{q_k - q_0}. \quad (15)$$

After replacing the left side of the equation (14) with value from equation (15), the histogram can be obtained precisely only for the idealized continuous probability density:

$$N^2 = \int_{q_0}^q \frac{1}{q_k - q_0} ds = \frac{N^2 * (q - q_0)}{q_k - q_0} = \int_{p_0}^p H(s) ds. \quad (16)$$

The desired pixel brightness transformation function P can be derived as:

$$q = P(p) = \frac{q_k - q_0}{N^2} * \int_{p_0}^p H(s) ds + q_0. \quad (17)$$

For a better idea of what histogram equalization is – refer to Figure 30 (Sonka, et al., 2008)Figure 30. This Figure shows HRCT of a lung before and after equalization. Figure 31 shows the difference between these two image histograms.

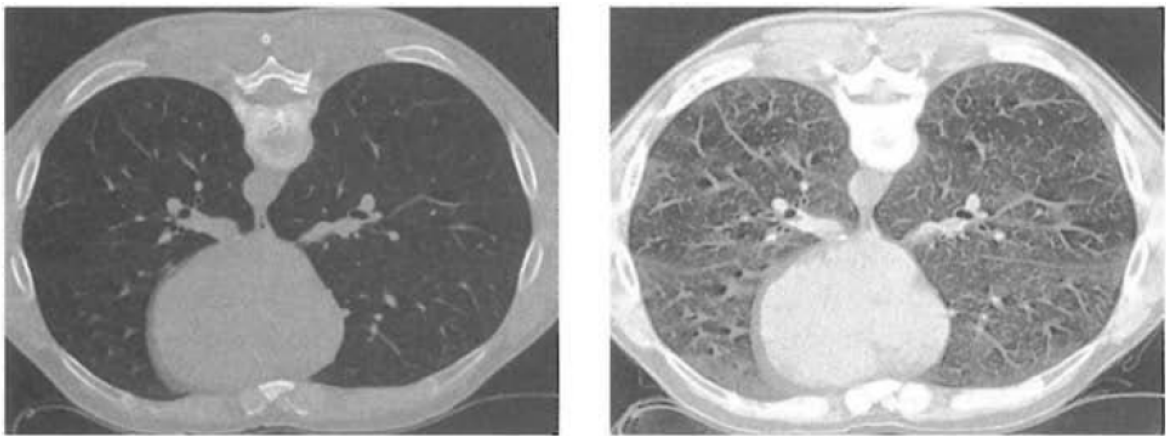


Figure 30 - Histogram equalization (Left) Original image. (Right) Equalized image.

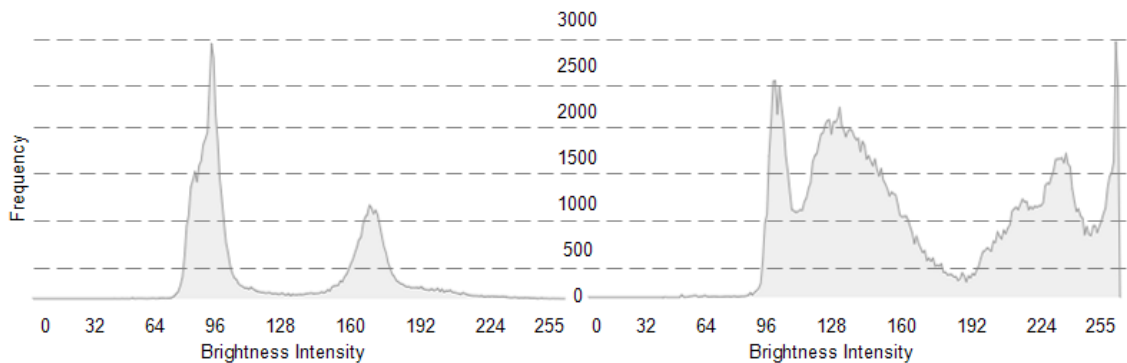


Figure 31 - Histogram equalization for Figure 30

3.2.3. Thresholding operations

The thresholding basic idea is to find an optimal gray-level threshold value for separate objects of interest in an image. Segmentation by thresholding operations are fast and simple for segmenting light objects on a dark background. (Chandrakala, et al., 2016)

Thanks to thresholding we can segment an image to segments, and after it we can detect edges by differences in light and dark segments. Thresholding operations work with gray-level images and after the threshold process we get a binary image in two colors - black and white. (Chandrakala, et al., 2016)

Easily can be said: "If $g(x,y)$ is a threshold version of $f(x,y)$ as some global threshold T , g is equal to 1 if $f(x,y) \geq T$ and zero otherwise." (Das, et al., 2011)

$$g(x,y) = \begin{cases} 1 & \text{if } f(x,y) \geq T \\ 0 & \text{otherwise} \end{cases} \quad (18)$$

Thresholding has a relatively big problem with pixel neighbors. This problem is ignorance of the relationship between neighboring pixels. The result can be computed with misleading errors thanks to image compression, digital noise or shadows in the image. In all of these cases some segments in the image can be lost - some segments can be marked as a "black" segment while it is not. (Das, et al., 2011)

The most commonly used thresholding techniques are simple thresholding, Otsu thresholding and Polynomial Curve Fitting. Another less commonly used method is adaptive thresholding.

Simple thresholding

The simple thresholding technique is based only on checking the condition if the value of the actual pixel is greater or lower. In general, if value T is greater than the pixel value, the pixel is marked as 1 (white - foreground), otherwise it is 0 (black – background). (Chandrakala, et al., 2016)

This technique is useful in discriminating the foreground from the background. If a best-fit threshold value is chosen, the resulting image contains information about the position and shape of the objects of interest (foreground). (Chandrakala, et al., 2016)

Otsu⁹ thresholding

Otsu's method is based on an automatic method of finding the threshold value that minimizes the weight class variance in the histogram. This method operates directly on the gray level histogram, because it will be fast. The histogram is computed only once before the method is executed. The method tries to minimize overlap of image segments by adjusting threshold

⁹ Method is named after Nobuyuki Otsu

value, however this method does not work properly with non-uniform illumination. (Chandrakala, et al., 2016) (Qu, et al., 2010)

Polynomial Curve Fitting

This method is based on segmentation of an image by fitting a polynomial curve to the histogram and finding the point of inflection, to get the minimum value for thresholding. Algorithms based on this method are very robust and efficient in comparison with the others. This method has however, the disadvantage of biasedness and impossibility of making a valid estimate of the residual variation of the curve. (Chandrakala, et al., 2016)

Adaptive thresholding

The next method is adaptive thresholding. This method can accept a grayscale image or color image as input; and output is a binary image representing the segmentation. This method computes a threshold for each pixel in the image. But the same principle is used as in simple thresholding. The value below threshold T is set to the background color and otherwise to the foreground color.

The difference between simple thresholding and adaptive thresholding is calculated with sub image (not only with one pixel). The threshold value for each pixel in this method is found by interpolating the result of the sub images. There is a risk when sub image is too large and bad threshold value is calculated. In this case optimal size of sub image needs to be found. Unless the sub image is well chosen, the method result is very poor and many objects in the image can be marked as background. (Chandrakala, et al., 2016) (Sonka, et al., 2008)

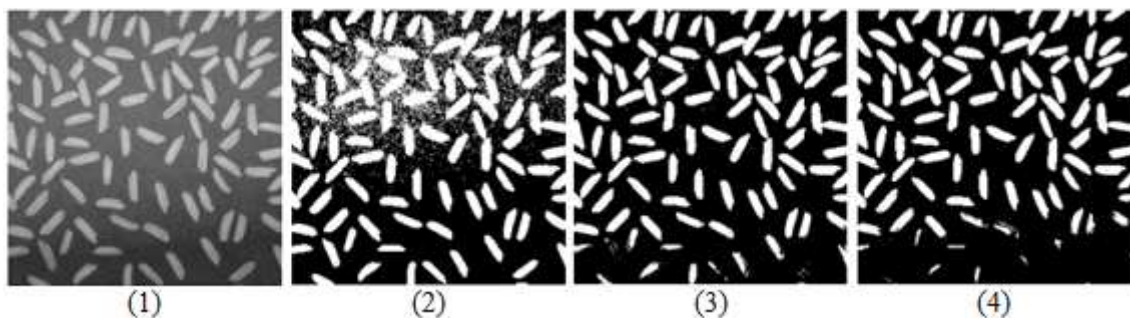


Figure 32 - Image thresholding (1) Original image, (2) Simple thresholding, (3) Otsu method and (4) Poly curve fitting (Chandrakala, et al., 2016)

Figure 32 - Imageshows the basic comparison of these methods on a photo of rice image.

3.3 Geometric transformations

All geometric transformations are based on displacement of pixels from their original position (x,y) to a new position $(x+t_x, y+t_y)$ in the new image (called spatial transformation).

These transformations allow elimination of image distortions or undesirable effects. The distortions might be caused e.g. by a lens distortion or a bad camera orientation.

The transformations can be used also for rotation of image or change of the view angle. Such operations are often used in remote sensing when in maps, there is a need to fix effects which arise by earth rotation, satellite position and so on.

Spatial transformation

In these transformations every pixel (x,y) of the original image is mapped to a new pixel (x', y') in a new coordinate system in the new image. But, the digital image is always a 2D array of image pixels in an implicit discrete grid. By this mapping a place out of the grid can be achieved (see Figure 33). In this case, there must be a final step of calculating pixel brightness from several neighborhood pixels. This brightness is usually calculated as an interpolation of these pixels. (Rhody, 2005)

$$x' = \phi_1(x, y) \quad (19)$$

$$y' = \phi_2(x, y) \quad (20)$$

Functions ϕ_1 and ϕ_2 in equation (19, 20) are functions depends on x and y (pixel coordinates).

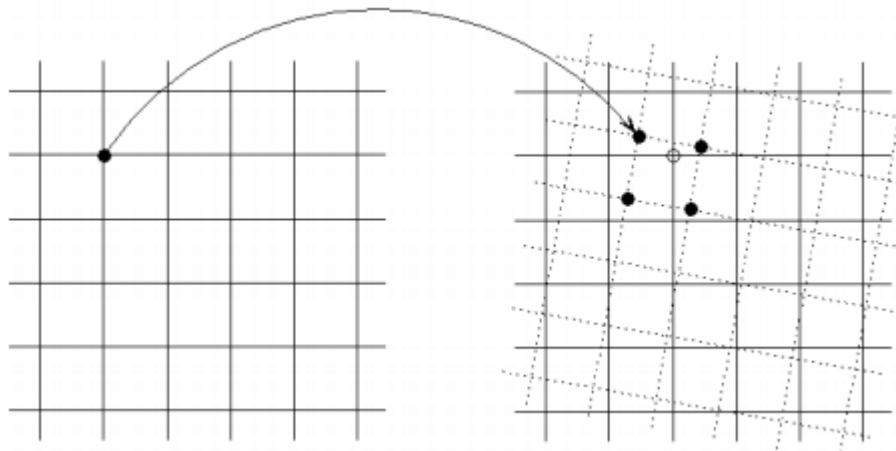


Figure 33 - Spatial transformation (Rhody, 2005)

Affine transformation

All possible geometric transformations are special cases of Affine Mapping. This mapping is a function between two affine spaces which preserves points, straight lines and planes. The general transformation equation in homogeneous coordinate system is:

$$[x' \quad y' \quad w'] = [x \quad y \quad w].M \quad (21)$$

Where M is general 3x3 transformation matrix.

$$\mathbf{M} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \quad (22)$$

In general, an affine transformation is a composition of translations, scales, rotations, and shears. An affine transformation can be represented by these matrix equation:

$$[x' \quad y' \quad 1] = [x \quad y \quad 1] \cdot \begin{bmatrix} m_{11} & m_{12} & 0 \\ m_{21} & m_{22} & 0 \\ m_{31} & m_{32} & 1 \end{bmatrix} \quad (23)$$

The result of the 2D transformations must lie on the same plane, e.g., $w' = w = 1$. W can be any value. Generally, the homogeneous coordinates must be divided by w' order to be left with results in the plane $w' = w = 1$ (Rhody, 2005).

3.3.1. Translation

All points are translated to a new position by adding offsets t_x and t_y to x and y , respectively. This operation is known as “shift”.

The translation transformation matrix is (Říha, 2012):

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} \quad (24)$$

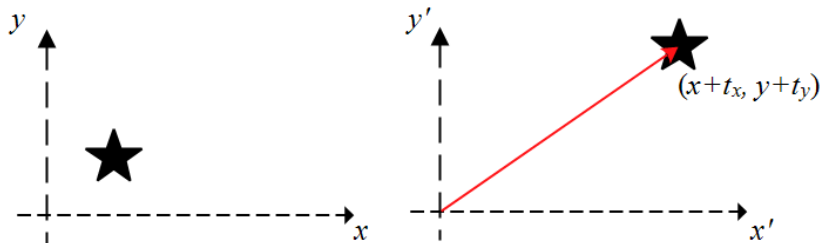


Figure 34 - Example of translation

3.3.2. Scale

Every point is scaled by applying the scale factors S_x and S_y to the x and y coordinates, respectively. To enlarge the original image, the both factors need to be specified as positive values greater than one. To scale down, both factors need to be set to number between zero and one. Negative scale factor values can cause the image to be a reflected, yielded and mirrored image.

This special type of scale is a non proportional resize. This case can appear when the scale factor S_x is not equal to S_y . Then the resultant image proportions are altered.

The scale transformation matrix is (Rhody, 2005):

$$\mathbf{M} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (25)$$

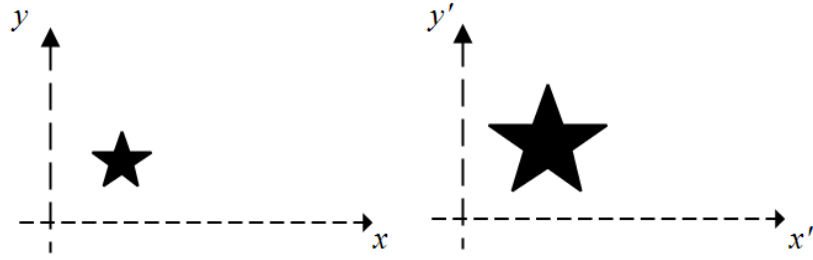


Figure 35 - Example of scale - $S_x, S_y = 2$ (Enlargement)

3.3.3. Rotation

Each point in the $x'y'$ -plane is rotated about the origin through the counterclockwise angle θ . The origin is in most cases, the center of image (Rhody, 2005).

The rotation is defined by this transformation matrix (Rhody, 2005):

$$M = \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (25)$$

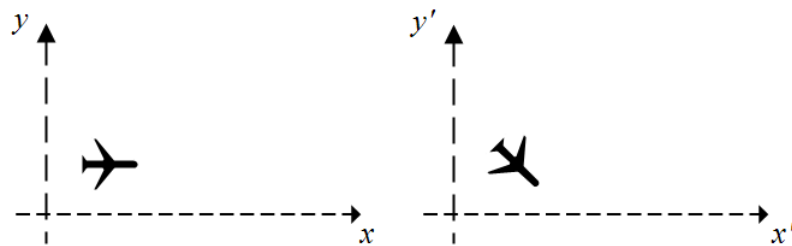


Figure 36 - Example of rotation - $\theta = 45^\circ$

3.3.4. Shear

The last basic affine transformation is Shear. By allowing m_{21} to be nonzero, x is made linearly dependent on both x and y , while y remains identical to y' . A similar operation can be applied along the y -axis to compute new values for y , while x remains unaffected. This effect, called shear, is therefore produced by using the off-diagonal terms (Rhody, 2005).

The shear transformation along the x -axis is:

$$M = \begin{bmatrix} 1 & 0 & 0 \\ H_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (26)$$

The shear transformation along the y -axis is:

$$M = \begin{bmatrix} 1 & H_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (26)$$

Where H_x and H_y is used to make x' , y' linearly dependent on x' and y' .

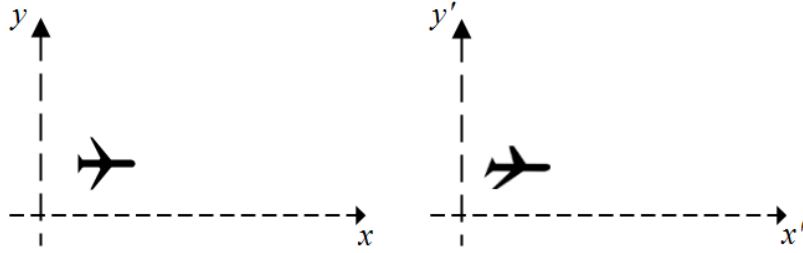


Figure 37 - Example of shear along x-axis

3.4 Composite affine transformation

The transformation matrix of a subsequence of simple affine transformations is:

$$\mathbf{M} = M_1 M_2 M_3 \dots M_n. \quad (27)$$

Then the inverse transform is:

$$\mathbf{M}^{-1} = M_1^{-1} M_2^{-1} M_3^{-1} \dots M_n^{-1} \quad (28)$$

Suppose that you want the composite representation for translation, scale and rotation (in that order) (Rhody, 2005).

$$\begin{aligned} \mathbf{M} &= \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} S_x \cos \theta & S_y \sin \theta & S_x t_x \sin \theta + S_y t_y \sin \theta \\ -S_y \sin \theta & S_y \cos \theta & S_y t_y \sin \theta - S_x t_x \sin \theta \end{bmatrix} \end{aligned} \quad (29)$$

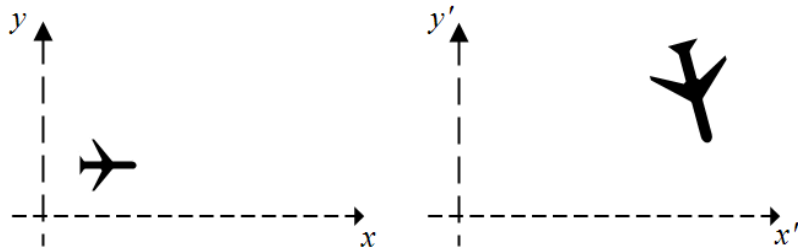


Figure 38 - Result - Translation, Scale 2x, Rotation $\theta = 75^\circ$

3.5 Perspective transformation

A perspective transformation results when m_{13} and m_{23} is nonzero. This transformation is frequently used in conjunction with a projection onto a viewing plane. This operation is known as a perspective projection. The perspective projection of any set of parallel lines which, are not parallel to the projection plane, will converge to a vanishing point – the center of projection. This transformation is useful for rendering more realistic images (Wolberg, 1988). Perspective projection causes perspective distortion (Sonka, et al., 2008).

Perspective transformation is used because work with 3D in computer vision is very difficult. This transformation provides all points along a line, pointing from the optical center towards a scene point, as one single image point. Perspective transformation is a loss-data transformation and inverse transformation is very hard to calculate. (Sonka, et al., 2008)

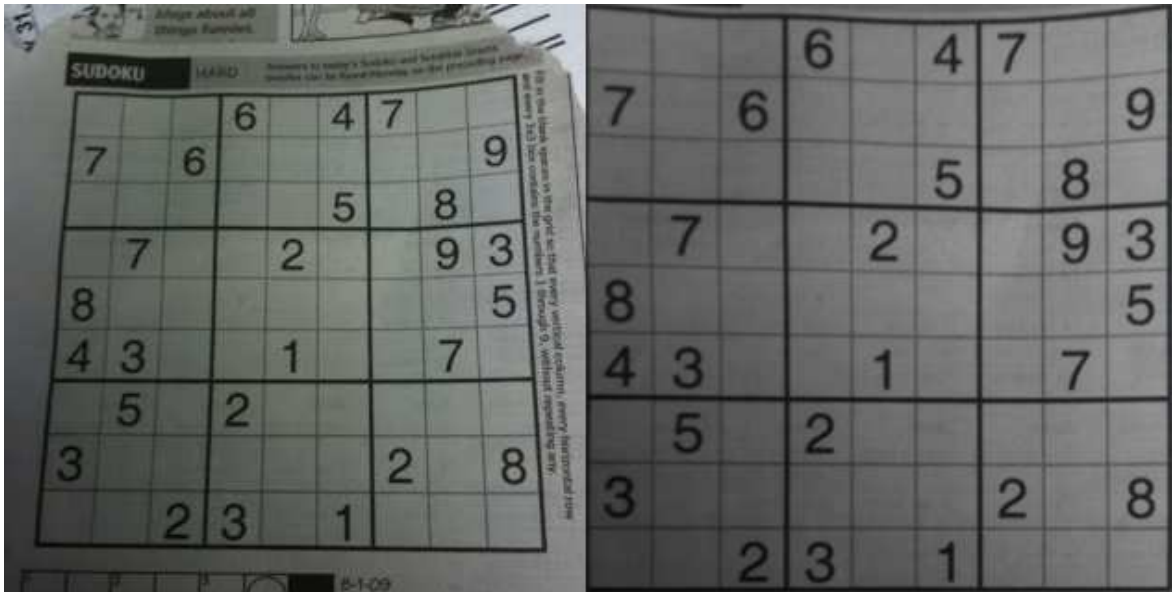


Figure 39 - Example of apply a perspective transformation¹⁰

Figure 39 shows perspective transformation for a Sudoku solver. The plane projection is needed to run the OCR and cell detection. Firstly corners of the base square are detected, and after, the image is transformed with this information to a plane projection. In this case, the transformation is done by OpenCV framework with knowledge of original corner position and supplied information about final corner position. The transformation matrix for this case is discretely calculated during the transformation process.

3.6 Image smoothing

The image smoothing process is in most cases used as a noise reduction filter – this is the key technology of an image enhancement, which can improve quality of a resulted image. So, this step is necessary in much image-processing software. In some literature, smoothing is also called blurring. (Szeliski, 2011) (Goyal, et al., 2012)

To perform a smoothing operation, it is necessary to apply a filter to an image. The most common used filters are a linear filter, a box filter, the Gaussian filter and a median filter. (Phillips, 2000) Every filter calculation is based on a function which is applied to every pixel of an image. This function is based on a kernel - which is in most cases, the center pixel in a filtering segment. The kernel pixel is replaced by a new pixel created by filter calculation. The new pixel is calculated from neighboring pixels. (Szeliski, 2011)

¹⁰ Source: http://opencvpython.blogspot.cz/2012_06_01_archive.html

Linear filter

The most common filter type of smoothing is a linear filter. Resulting pixel value is determined as a weighted sum of input pixel values.

$$G(x, y) = \sum_{k,l} f(x + k, y + l) \cdot h(k, l) \quad (30)$$

Where is $h(k,l)$ the mentioned kernel, which is nothing more than the coefficients of the filter. (Szeliski, 2011)

Gaussian filter



Figure 40 - Example image with applied Gaussian filter

The Gaussian filter is based on Gaussian function. According to this function, the weight of neighboring pixels is decreased by the spatial distance between them and the kernel. The biggest weight value would be in the middle (kernel). This decrease can be shown in Figure 41, where the blue color symbolizes a lower value coefficient and red color is the highest. (Szeliski, 2011)

$$G_0(x, y) = Ae^{\frac{-(x-\mu_x)^2}{2\sigma_x^2} + \frac{-(y-\mu_y)^2}{2\sigma_y^2}} \quad (31)$$

Where is μ the mean and σ represents the variance.

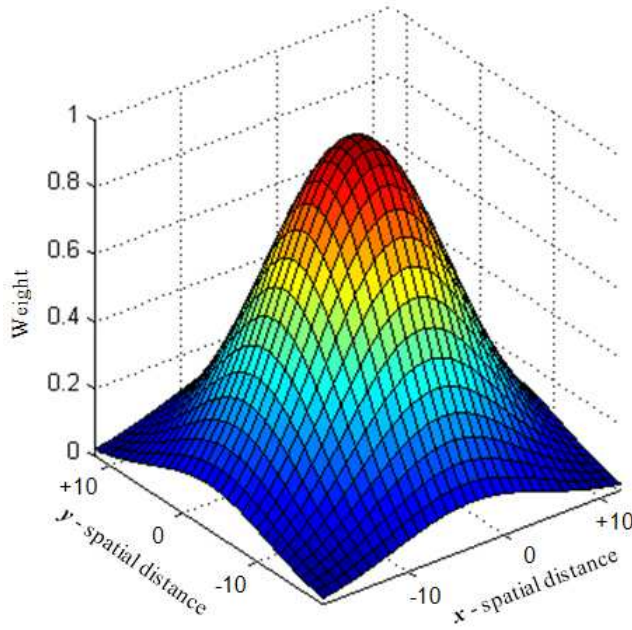


Figure 41 - 2D Gaussian function graph

Box filter

The Box filter is based on a linear filter. The kernel value is still based on equation (30), where $f(x+k, y+l)$ is calculated from a simple matrix with means of neighbor's pixels and coefficient. The matrix and matrix size for filter is specified by the user. There are three examples of 5×5 filter matrix with their coefficient:

$$h_{rect} = \frac{1}{25} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad h_{circ} = \frac{1}{21} \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix} \quad h_{pyr} = \frac{1}{81} \begin{bmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 4 & 6 & 4 & 2 \\ 3 & 6 & 9 & 6 & 3 \\ 2 & 4 & 6 & 4 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{bmatrix}$$

The coefficient is the sum of individual means of elements. These three matrices contains h_{rect} = Uniform rectangular filter, h_{circ} = Uniform circular filter and h_{pyr} = Triangular pyramidal filter for an image smoothing. (Szeliski, 2011)

3.7 Shape analysis

The main goals of shape analysis are shape recognition, categorization, matching, description, decomposition and determination of position and orientation. By shape analysis many entities like flat objects, projections of 3D objects, segmented binary images, planar shapes, curves, and closed contours can be analyzed. (Acharya, et al., 2005)

On shape analysis, many object recognition algorithms are based, because the basic shape of an object is stable and is not expected to change. Every shape is the ensemble of geometrical information of an object which does not change, even though the location, scale

and orientation of the object are changed. Thus the shape is invariant to Euclidean similarity transformations. (Acharya, et al., 2005)

There is several approaches for analysis methods. These approaches can be divided to three main groups (Csetverikov, 2003):

- **Scalar transform** vs **Space domain** methods. In scalar methods objects are compared by statistical pattern recognition algorithms and in space by structural pattern recognition algorithms. Output of scalar transformation are vectors, and output for space domain is another image.
- **Information preserving** methods where is loss of information controllable and **information non-preserving** where is information loss is not controllable.
- The last group is based on **area-based** with points ordered in 2D and **contour-based** methods where points are ordered along contours. Analysis by this kind of approach can be suitable for different kinds of analysis.

Landmark points (Key points)

Landmark points are one way to describe a shape pattern by defining a finite set of points inside the object. These points are classified as (Acharya, et al., 2005):

- **Anatomical landmark points** – assigned by experts – points correspond between organisms in some biologically meaningful way (joints of tissues or bones).
- **Mathematical landmark points** – assigned by mathematical or geometrical property (curvature or an extremum point).
- **Pseudo-landmark points** – set of constructed points on object either on the outline or between other landmark points. May be appropriately defined by the user.

Each landmark points can be described as node or vertex of a polygon enclosing the shape pattern. These points are key points or markers on any object (Acharya, et al., 2005).

Polygon as Shape Descriptor

One of the powerful representations of a planar shape may be the joining of the ordered pair of an n-point polygon in k dimensions. Concatenating each dimension into a $k \times n$ -vector can achieve this. Two dimensional planar shapes may thus be represented as (Acharya, et al., 2005):

$$x = [\{x_1, x_2, \dots, x_n\}, \{y_1, y_2, \dots, y_n\}]. \quad (32)$$

“The location, scale and rotational invariant shape representation may be achieved by establish a coordinate reference with respect to position, scale, and rotation, to which all the shape patterns should be aligned.” (Acharya, et al., 2005) The shape alignment procedure involves four steps (Acharya, et al., 2005):

- Compute the centroid of each shape.
- Resize each shape to equal size.
- Align with respect to position at their centroids.
- Align with respect to orientation by rotation.

Dominant points in Shape description

One method, to detect a set of dominant points, is to determine the curvature at each point, and calculate the resultant cumulative curvature for all the points starting with the last detected dominant point. (Teh, et al., 1989)

The dominant points are points along an image outline that mark important information about the shape. These points are usually points of high curvature. In general, two different approaches exist to find the curvature extrema on a digital curve. (Acharya, et al., 2005)

- **Method 1** – to detect the dominant points directly through angle or corner detection schemes (Teh, et al., 1989).
- **Method II** – to detect a piecewise polygonal approximation of the digital curve depends on certain restrictions for the shape preserve. Thus dominant points correspond approximately to the intersections of adjacent line segments of the polygon. These intersections are known as vertices of break points of the polygon. (Teh, et al., 1989)

4 Problem analysis

The goal of the master thesis is to design the high-level control system of the discussed teaching (see section 1). Logically, the first step of the development process is an analysis of the problem. The outputs of the analysis will allow us consideration of hardware requirements and determination of an appropriate framework.

For this teaching aid we need to distinguish between two roles. The first one role is users, and the second one is a supervisor. The role of the user is need to be allowed connect their path-planning routines, chose the target position, select the path-planning routine, and initialize a path-plan execution. The role supervisor's role must have the same authorization as the user role; however, the supervisor should be allowed for changing the maze layout and carry the robot to any position in the maze.

The high-level control system must provide to the user functionalities from three different domains. The first domain consists of Path-planning related tasks. Specifically, tools for input start and goal position, way to connect a student routine, and transformation of a path-plan to a sequence of steps belong here.

The next domain covers image processing related tasks. These tasks provide an extraction of information from an input camera image about a maze configuration and a robot position. These tasks map the information to a discrete model.

The last domain has a close relation with communication related tasks. It covers a high-level control system interaction with the camera system, the robot, and the student's routine. The communication is supplied by standard communication protocols, e.g. WiFi is used for the communication with the camera, Bluetooth for the communication with the robot. A user vs. high-level control system, uncured by a graphical user interface (GUI), belongs also to this domain.

One of the basic requirements is a simple selection of the path-planning algorithm. The communication with path-planning routines will be realized via sockets. This protocol will be specified in subsection 5.1.5 of this thesis. Next requirement is a simple use of the GUI on tablet or another hardware with a touchscreen. This will allow us to use the teaching aid within open days when its usage by children and youth is expected.

Within the problem analysis, I suggested communication protocol for these applications. Via this protocol, user routines get data from the high-level control system and they can calculate path-plans for the data. If the student routine finished the path-planning process, the path-plan can be sent back via this protocol to the high-level control system, and then system will send a transformed path-plan to a robot. Selection of the path-plan algorithm is completely dependent on the student decision.

4.1 Objectives of teaching aid

The main objective of the teaching aid is to support an understanding of the path-planning problematic by students. Within the practical work, the students improve their knowledge of path-planning algorithms. Further, the teaching aid might be used during the open days.

4.2 Requirements

Based on the problem analysis, we divided requirements on the high-level control system on functional and non-functional requirements.

4.2.1. Functional requirements

The functional requirements are requirements which specify functions and parts of the final application. Following functional requirements were identified:

- **Requirement on a tool for analysis the maze** – this requirement is the base for high-level control system.
- **Requirement on a tool for the robot detection** – this requirement is needed because the high-level control system needs to know a robot's position and its orientation.
- **Requirement on a communication module with the robot** - this requirement is needed, because we need have some interface for communication with robot.
- **Requirement on a communication protocol with the student routine** – this requirement achieve the communication between a high-level control system and a student routine. And a protocol must be simple for an implementation, and easy understandable for implementer.
- **Requirement on a management of an attached student's routine** - this requirement is needed, because in a one moment can be attached more than one routine and we need to be able to switch between them.

4.2.2. Non-functional requirements

The non-functional requirements are requirements which specifies the result application, for example, with appearance restrictions or defines the hosting system. In my case, are defined only these two non-functional requirements:

- **Requirement on an intuitive GUI** – within this requirement is needed to design a full screen application with big icons. This requirement meets the request to usage of the touching device. The big icons is needed for a comfortable control (avoid the miss click).
- **Requirement on a portability of the application between various operating systems.**

4.3 Ready-to-use tools

4.3.1. Maze detector analysis

The most complicated part of the high-level control system is a maze detector. The maze detector ensures analysis of a maze layout using top view images captured by the camera. The analysis of the maze layout is essential for creating the discrete model of the robot's operation environment. We used the exact cell decomposition for the conversion of the environment into the discrete model. In our approach, the adjacency matrix is used for its representation.

Various image processing and computer vision algorithms are employed in the detector developed by Škrabánek. (Škrabánek, 2015) It takes advantage of properly chosen colors of the maze components. An example of the maze is shown in Figure 1. Yellow corner markers, black and red partitions, and white floor are highlighted in this figure.

Simplified, the analysis of the maze layout consist of following steps:

1. determining the position of the maze in the image;
2. enhancement and transformation of the image;
3. measurement of the partition length using the red partitions;
4. conversion of the input image into a binary image,
5. analysis of the maze layout using a sliding window applied on the binary image, resulting in the discrete model.

Size of the sliding window is determined by the size of the partitions. As is shown in Fig. 44, the sliding wind captures a cell of the maze and its nearest surrounding. Presence of partitions is judge using a complex technique which is detailed in (Škrabánek, 2015).

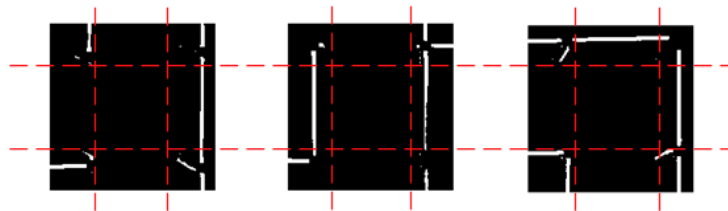


Figure 42 - Sub image divide

4.3.2. Robot detection

A solution aimed at localization of the robot in the maze has been developed by Škrabánek. It allow using various Transformers symbols for robot's marking. (Škrabánek, et al., 2014)

5 Software design

Software design is a process which creates specifications of a software to meet the requirements of the product owner. This process is helpful to programmers to understand the problematic of creating concrete software. In general, this process is done by the product owner or software architect. The programmer only implements the designed result. (Lyytinen, et al., 2007)

The main task while software design is done, is specification of software modules, basic data structures, and their responsibility in resulting software. Every module has a specified collaborative module with which it can communicate. If there is a module without a colleague, there might be a problem in software design. Ideally, every one module is responsible only for one thing, but in most cases it cannot be done easily.

5.1 Modules

In this part of thesis, I will describe all system modules and their basic responsibility and collaborative modules, which are a result of the problem analysis.

5.1.1. GUI & environment

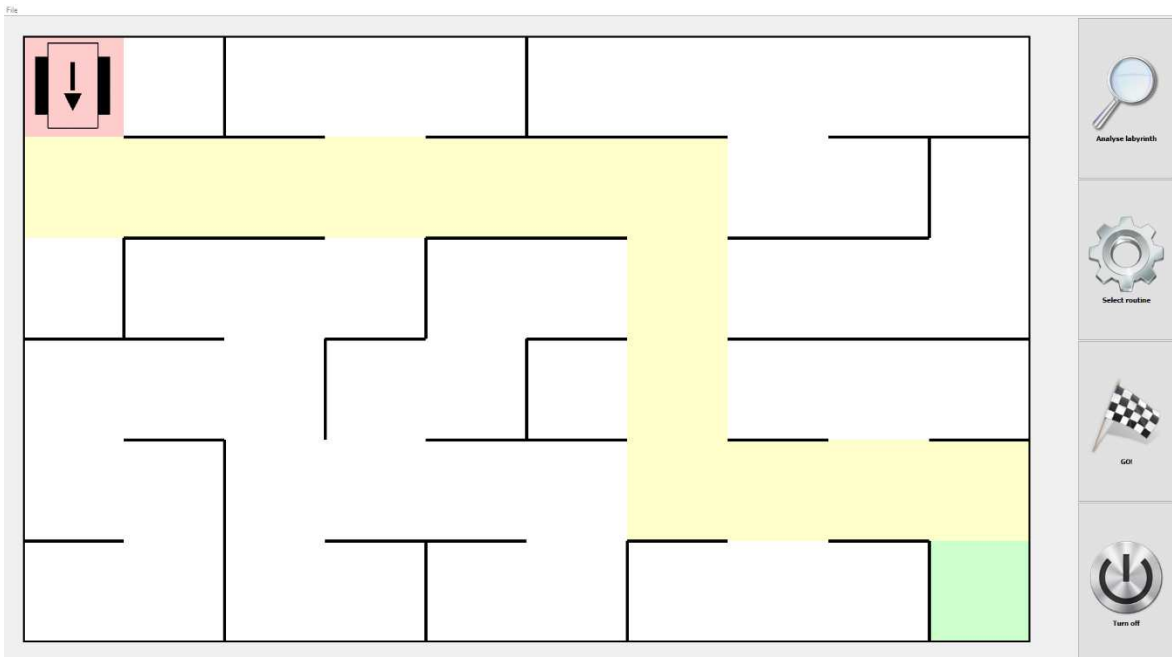


Figure 43 - Final graphic user interface

This is the most important part of the high-level control system. An actual state of the maze is shown to the user by GUI. The adjacency matrix is used for render the maze. The application works under a full screen mode. A robot position is updated every second and the full maze analysis is done after the user clicks on "Analyze maze" button.

There are three more buttons – Select routine, Go, and Turn off. At the start, buttons "Select routine" and "Go!" are disabled. It is because no student routine is connected. When a routine is connected both buttons are allowed. With "Select routine" button, the user is shown a dialog with selection of a student's routines, where he can decide which routine to use. The button "Go!" function, is to submit the step sequence to the robot. Button "Turn off" will close the full application. To select the target position, touch the cell on the screen.

In the maze cells are with different colors. Where red color marks the start cell, green is mark for a target position and yellow is the calculated path by a student's routine. The robot marker marks the robot position in a real maze.

This module collaborates with a maze layout analysis, a robot observer, a robot communication module, and with a client manager. This module has the most workload and the only one which has visual interface.

5.1.2. Camera system module

The camera system module has only one responsibility. This responsibility can be divided to a two functions. The first one, is communication with a digital video camera over RTSP/HTTP protocol, and the second one is to provide an image from a camera to the requesting module. This module has only one external relation with OpenCV framework - which is used to connect with a digital video camera.

If the camera system module returns no image, an error message will be shown. This state can happen while there is no digital video camera connected or if it is not available. Both of these problems can occur because a camera is not directly connected by USB¹¹/FireWire¹² or so on.

This module can communicate with different cameras, which support the RTSP protocol or direct HTTP access to an actual image.

5.1.3. Maze layout analysis

The maze layout analysis is the second most important module. This module has the main responsibility of the maze analysis. This module collaborates with the camera module and with the GUI. This module will work in its own thread, because the analysis process can block the application – when the analysis is not finished quickly (slow image transfer, big maze...).

When the user wants to analyze a maze, this module will request a new image of the maze from camera module and analyze it. After analysis is done, the adjacency matrix will be returned to the GUI module. GUI module will update a graphic of the maze panel and show the new configuration of the maze.

¹¹ Universal Serial Bus for connecting peripherals.

¹² Standard serial bus (known too as IEEE 1394) for connecting peripherals – mostly for cameras.

The module is based on the maze detector which was introduced in subsection 4.3.1. The maze detector is based on OpenCV framework. In this framework is implemented much algorithms to work with an images, cameras, colors and so on. The size of the sliding window was determined as 1.3 times the length of partitions.

The implementation of this module was the most time consuming part of the development process. Within the development process, several challenges arised. They resulted in several modifications.

The main problem of the maze detector is diverse environment. In these environment is some influences which are needed to be prevent or reduce. In relation to physical environment are there sun shine and shadows. In case, of this two are needed to adjust ambient light. This shadows can impair maze detection, because the maze detection is based on colors.

Next problem can be a bad camera position or a non-parallel camera view with base of maze. Both of these errors can misled by operating student or supervisor. In the maze detector I tried to fix this problem by perspective projection based on a corner detection. When this problem appeared some next problem are needed to be solved. This problem is to find the last corner. Paper on whose is based this teaching aid, counts only with three marked corners. But while I tried to calculate last maze corner, I was been confronted the problem with imprecise of it. This problem is shown on Figure 44.

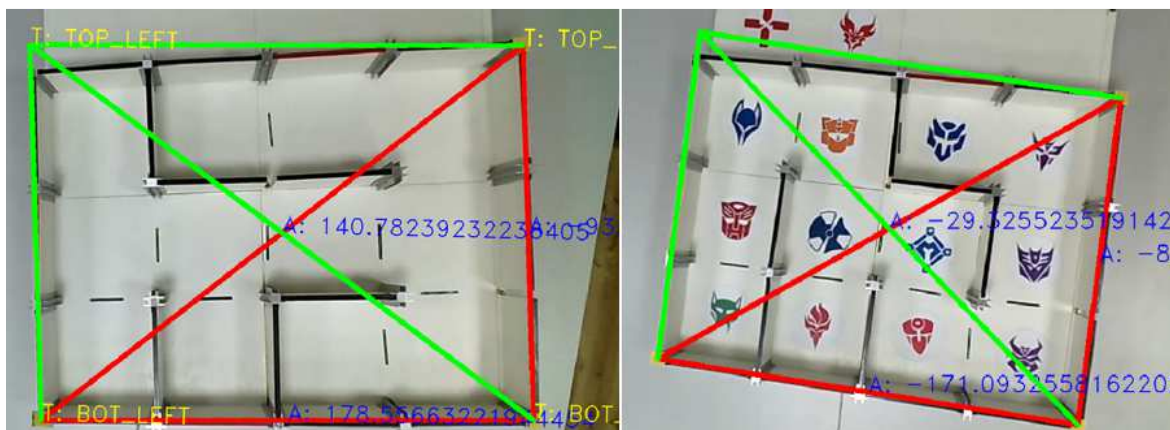


Figure 44 - Maze with 3 corner markers and calculated last corner

To prevent this problem is needed to place marker to last corner and detect every corner. The perspective transformation is done, after detection of corners is completed. Thanks to this transformation can be easily cropped the original image, according to the position of the corners. The result of this process is shown on Figure 45 **Chyba! Nenalezen zdroj odkazů.**

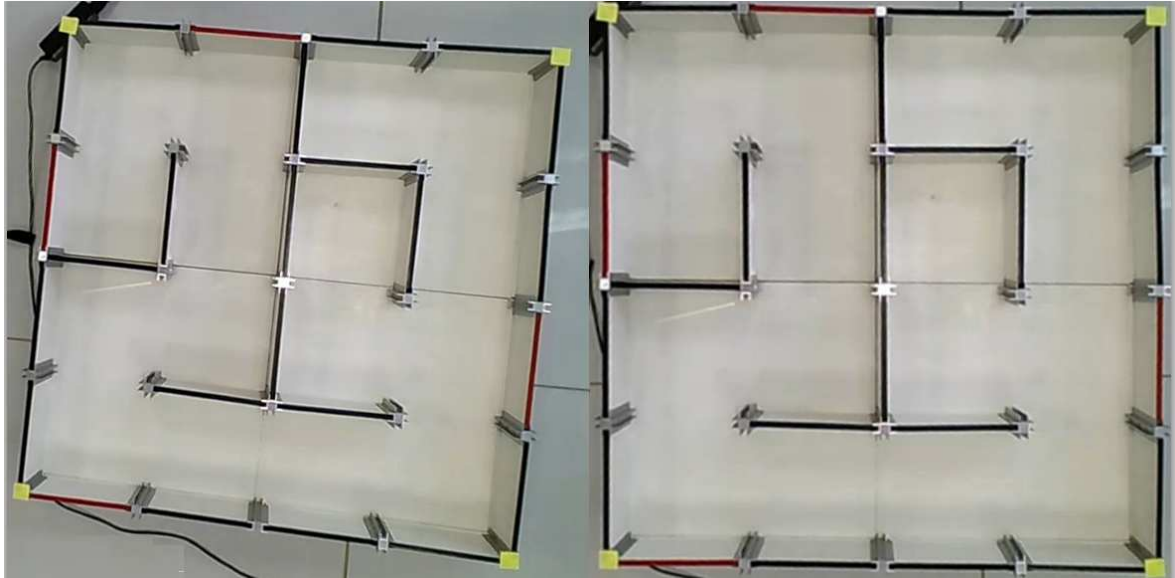


Figure 45 - Maze with 4 corner (a) before perspective transformation (b) after

5.1.4. Robot observer

The robot observer module is active. Active, signifies a standalone thread which contacts GUI when the robot position changes. The base timer for this module is set to 500 milliseconds and its work starts after a first successful analysis of the maze is done.

This module has only one responsibility and it is to observe the robot, and transfer the new position and rotation information to GUI. The GUI will update graphics after accepting this information and repaint the robot position.

5.1.5. Client manager

The client manager module has responsibility for accepting a connection from student routines and communication with this routine. The communication is defined by a simple text protocol which will be described below.

This module is designed as a pool of connections, where a connection is created for every new student's routine. User can switch between these connections (as shown in Figure 46) in the GUI. Every connection has its own name to distinguish routines and own a thread for checking, if connection is still available. To check availability of a routine, the simple ping request, with 5 repeats in 500 milliseconds, is used. Thus, if student's routine does not respond for 3 seconds, it is disconnected, connection is closed, and GUI selection/buttons are updated.

Every communication (excluding an incoming client connection from a student's routine) between high-level control system and a student's routine is started by a HLCS request. This is to avoid a flooding request from a badly written student routine. Every unknown command is thrown out.

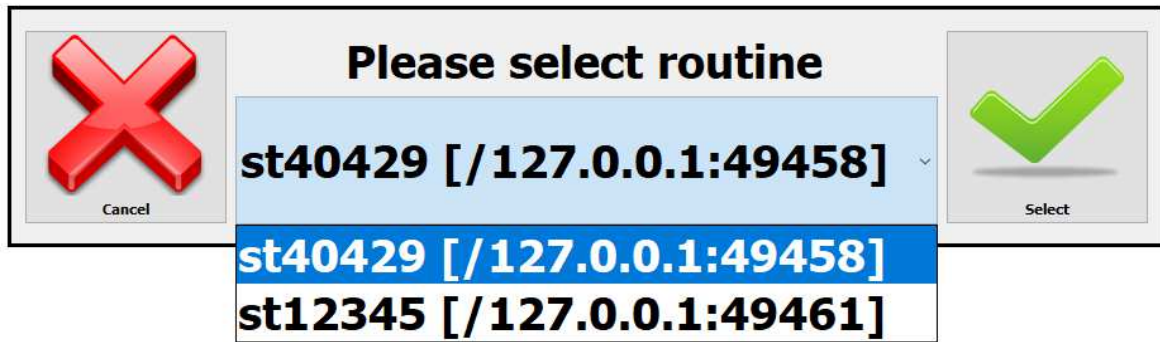


Figure 46 - GUI form for routine selection

Client communication protocol

The server side commands:

- **Name?** – This command is used for a new incoming connection to determine a student routine’s name. The expected response is name in the student NetID format (st#####). This format of NetID is defined by University of Pardubice.
- **ping** – Simple command for determine if is connection still alive. Simple response is “ping”.
- **solveproblem** – This command signalizes a data block with adjacency matrix for an actual maze. Lines with zeros and ones, which are separated by commas, are sent after this command. Every line is for one cell. This command has a no response.
- **dataok** – By this command, client can recognize end of an adjacency matrix. Response can be some commands from client side.
- **QUIT** – This command is used for inform the client about connection close. This command has a no response.

The client side commands (acceptable by server side):

- **from** – This command is used to get an actual position of the robot in maze. The expected response is a numeric cell ID.
- **to** – This command is used to get the required target position. The expected response is a numeric cell ID.
- **dimension** – This command can be used to get the dimension of maze. The expected response is two numeric values of width and height separated by comma (e.g. “5,2”).
- **ok** – This command is used to signalize to HLCS about computing the solution. This command has no response.
- **solution** – This command is needed to be submitted after **ok** command. This command signalizes to HLCS when path-planning is done. After this command server expects next response with cell IDs of path plan. These cell IDs are needed to be sorted by path-plan and separated by commas (e.g. “1,3,5,6,7”).

The example of protocol usage can be found in **Attachment A**.

5.1.6. Robot communication module

The last module is the robot communication module. This module has a responsibility to communicate with a robot and transform the path-plan to a sequence of actions. This transformation is needed because the robot has no information about cell IDs in the maze.

The problem analysis defines only three types of actions for the robot. These actions are LEFT (rotate -90°), RIGHT (rotate 90°), and FORWARD. This three actions are enough to describe how to go over the maze. In the most extreme situation, only two actions can be used – ROTATE (for 90°) and FORWARD, but it can prolong the robots ride (by rotation for 270° degrees – 3 times rotate for 90°). (Škrabánek, et al., 2016)

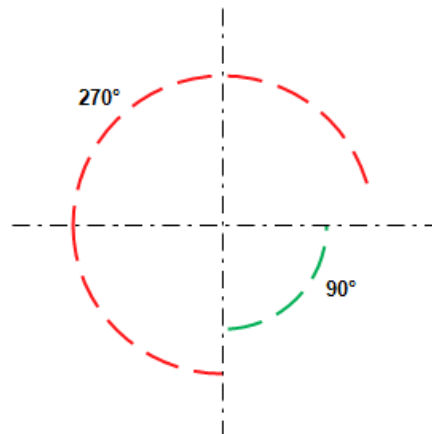


Figure 47 – Comparison of trajectories for 270° rotation and 90° rotation to same final direction

5.2 Analytic Class diagram

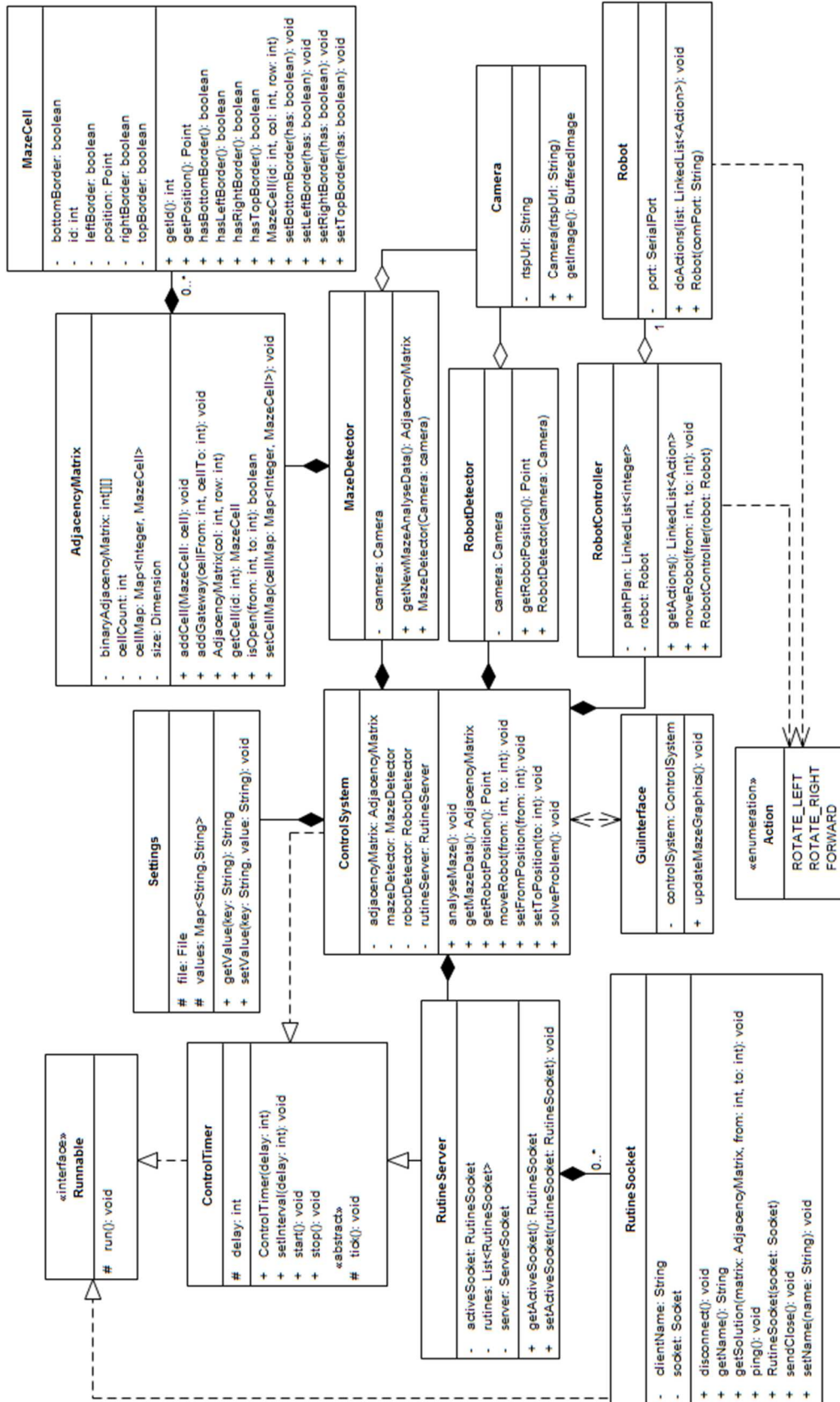


Figure 48 - Analytic Class diagram

The analytic class diagram shows interaction between classes. These classes are designed during the problem analysis process. Either class can be implemented in the final HLCS a bit differently, because the analytic class diagram is only a proposal of final software. From the analytic diagram private methods and language classes are removed.

This analytic class diagram indicates relations between classes, attributes of classes, public or protected class methods, and special classes like interface and enumerations.

Based on this analytic class diagram a high-level control system in JAVA Language is implemented.

5.3 Implementation

The high-level control system implementation can be divided into three parts. These three parts are defined by packages in source code. This division is shown in figure Figure 49 - HLCS Source Packages.

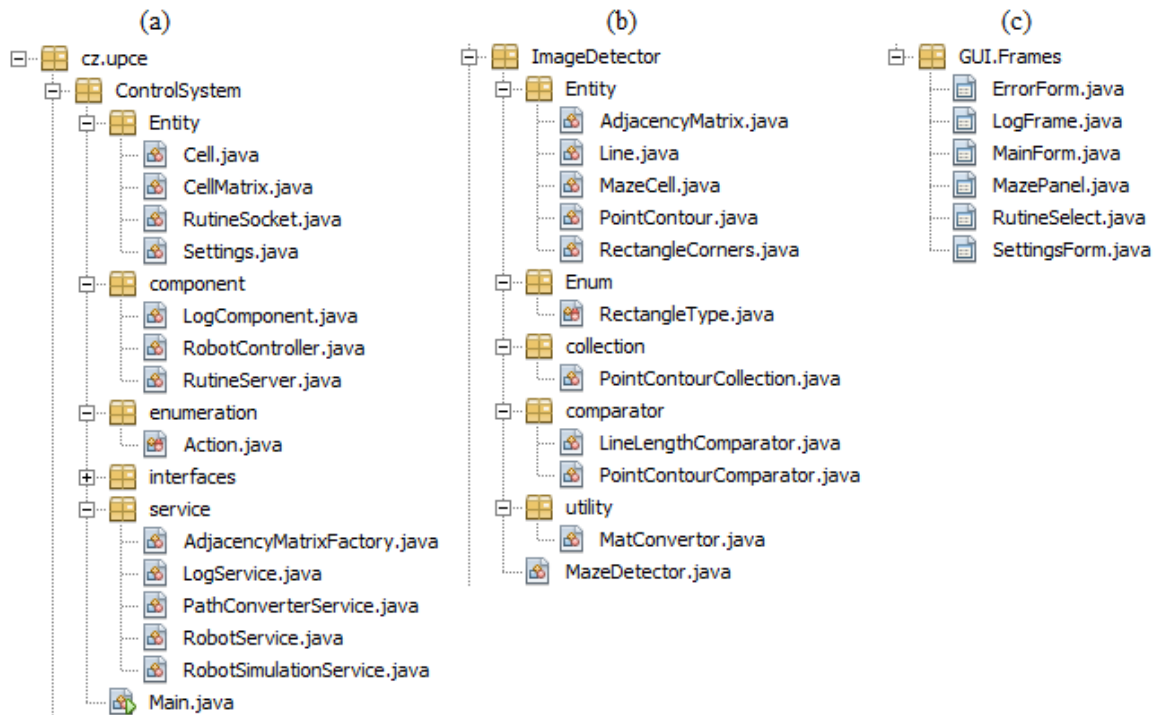


Figure 49 - HLCS Source Packages

There are two main executive parts and one viewing part. The executive parts are (a) Control System and (b) Image Detector; (c) GUI.Frames is viewing part.

From ControlSystem (a) the most important classes are RutineServer (service for connected routines and communication with their clients), RobotService (service for robot), and PathConverterService (provides a path-plan transformation to sequence of steps – Actions). For ImageDetector (b) the important classes are MazeDetector (providing the maze and robot detection) and AdjacencyMatrix (entity class represents adjacency matrix). For

GUI.Frames (c) MazePanel (which showing the entire maze configuration) and MainForm (main full screen window) are important. The starting class is Maze in ControlSystem (a).

6 Experimental verification of the solution

For an experimental verification, the test scenarios are needed. These scenarios are designed to check correctness of a teaching aid. The test scenarios were designed with respect to the specific requirements.

6.1 Maze detection

The basic functionality of a maze detector will be tested in this test scenario.

Assumptions: The software is switched on.

- a) The maze is detected from camera image requested from camera module,
- b) the software shows an error message while maze analysis is obtained and camera is switched off,
- c) the software shows an error message when the maze analysis fails,
- d) the maze is detected from adjacency matrix data file,
- e) the software shows an error message when file does not contain the adjacency matrix data,
- f) the maze is detected from an image file,
- g) the software shows an error message when image file does not contain a maze.

6.2 Student's routine interface

The basic functionality of client manager, and interaction between client manager and GUI will be tested in this scenario.

Assumptions: The software is switched on.

- a) The buttons "Select routine" and "GO!" are disabled, if no routine is connected,
- b) an error message is shown, if touch to cell is performed,
- c) the buttons "Select routine" and "GO!" are enabled, if first routine is connected,
- d) the buttons "Select routine" and "GO!" stay enabled, if next routine is connected,
- e) the buttons "Select routine" and "GO!" stay enabled, if one of these routines is disconnected and any routine is still connected,
- f) dialog "Select routine" is shown after clicking on "Select routine",
- g) the buttons "Select routine" and "GO!" are disabled, if last routine is disconnected,
- h) the routine is removed from "Select routine" dialog if a connection is lost,
- i) if the dialog "Select routine" is opened and last routine is disconnected, the dialog will close.

6.3 Getting path-plan from routine and transform to a sequence of steps

The combination of functions are tested in this scenario. In this case, communication protocol is tested with student routines, robot communication module which transforms the

path-plan to a sequence of steps, and GUI which shows this information. The communication protocol with student routine is also tested within this test case.

Assumption: The software is switched on. Robot communication interface is using simulation mode (software is switched to simulation mode). One or more student routines are connected. The software is in default settings.

- a) The robot marker is shown at GUI and background of marker is red,
- b) the target cell has green background after touch to cell is performed,
- c) the new path-plan is shown by yellow background of cell,
- d) the path-plan indication is repainted, if click to other cell is performed,
- e) a robot communication module does nothing,
- f) the path-plan is transformed to a sequence of steps, after click to “GO!” button is performed,
- g) the simulation mode performs steps in order of the sequence of steps,
- h) the executed path-plan is removed from GUI and new robot position is re-colored to red,
- i) the new position is submitted as new start state, and the scenario can be repeated.

6.4 Robot detection

The basic functionality of a robot detector will be tested in this test scenario.

Assumptions: The software is switched on. The path plan routine is connected. Robot communication interface does not use simulation mode. The maze is successfully analyzed and shown on GUI.

- a) The robot marker is shown on GUI for the robot’s real position,
- b) the robot is rotated according to a real rotation in a maze,
- c) the robot is tracked, while path-plan exists and user clicks on “GO!” button,
- d) the robot position on GUI is updated after another real cell is reached.

7 Evaluation of verification

Results obtained from the verification process according to the test scenarios were summarized in Tables 3 – 6. As follows from the results, the computer vision tasks are the most prone to errors. Specifically, the robot detection module shows the most serious errors; however, the maze layout analysis is also imperfect. These issues will be considered in the following subsections.

Maze detection		
Label	Result	Revealed error
a	inaccurate	Not all mazes are detected
b	OK	
c	OK	
d	OK	
e	OK	
f	inaccurate	Not all mazes are detected
g	inaccurate	Some images fall into detection

Table 3 - Evaluation of the maze detection test scenario

Student's routine interface		
Label	Result	Revealed error
a	OK	
b	OK	Sometime come "lag" before error message is shown.
c	OK	
d	OK	
e	OK	
f	OK	
g	OK	
h	OK	
i	OK	

Table 4 - Evaluation of the student's routine interface test scenario

Getting path-plan from routine and transform to a sequence of steps		
Label	Result	Revealed error
a	OK	
b	OK	
c	OK	
d	OK	
e	OK	
f	OK	
g	OK	
h	OK	
i	OK	

Table 5 - Evaluation of the getting path-plan from routine and transform to a sequence of steps test scenario

Robot detection		
Label	Result	Revealed error
a	inaccurate	The robot is not detected at all places in the real maze.
b	Fail	Robot rotation is not recognizable.
c	Fail	Same as a part of test scenario.
d	partial OK	The robot position is updated, but not all time to a new correct position.

Table 6 - Evaluation of the robot detection test scenario

7.1 Maze detection

During the experimental verification of the maze detection, I tried to find bad implementation in the maze detector. In the course of a test 10 different maze configurations have been tried. 5 of these configurations are square and the others are rectangles. All of these configurations have been tried in different view angles, different rotations, and different positions in the workspace.

Verification is relatively good. Only a few maze configurations were not detected properly. During the tests a problem with light was apparent. Thanks to an ambient light from a window, some walls disappeared thanks to a reflective surface on top of the partition. This problem has been partially solved by the addition of extra light above the maze.

Removing the reflective surface from walls will be more helpful. This problem is shown in Figure 50.

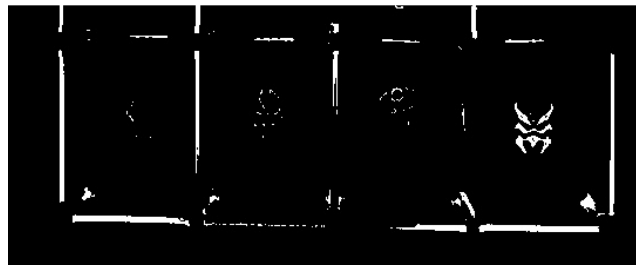


Figure 50 - Maze with undesirable light

The next slight problem is a bad detectable maze which is too much rotated (above approximately 35°). When the maze is rotated too much, detection is not successful, because the maze is "destroyed" by perspective transformation. The result of bad detection is shown in Figure 51.

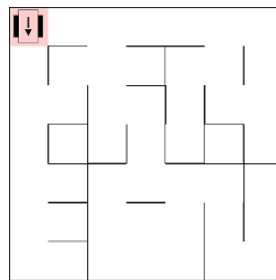


Figure 51 - Bad detected maze while is too much rotated

I was unable to remove this kind of problem, but I identified this problem occurring in perplexity of corner positions. In every detection, I tried to mark all corners by their name (like in **Chyba! Nenalezen zdroj odkazů.**) however this problem of undetected corners remained. This problem can be fixed by rotate the maze.

The last problem is caused by bad camera position. This will be done in most cases by the operating user. Only returning the camera to the proper position is enough to fix this problem.

7.2 Robot observing

Verification of the robot observing module was unsuccessful. During the test I tried to implement two versions of a robot detection. The robot detection is based on robot markers. The robot actual position and his rotation in maze is needed to be found. Unfortunately, all of my implementations were unsuccessful.

The first implementation by a classic detection with a cascade classifier failed when a robot marker is in light shadow and is not always detectable. This implementation is unusable. However, the robot orientation detection fails in all cases.



Figure 52 – Key points of sample robot markers (colored circles)

Secondly I implemented the robot observation with landmark point detection. Everything works well, while I tried to detect the robot in an image, with approximately same size of a robot marker in image. When I used the smaller image as source of key points – these points are incalculable by the detector.



Figure 53 – Sample key pointed maze with markers

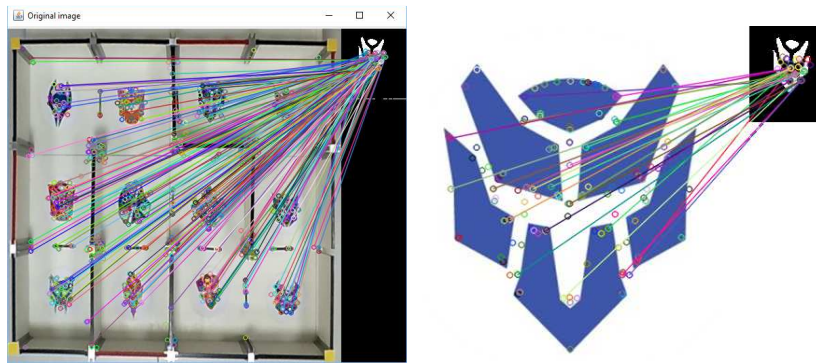


Figure 54 - Sample bad key point association

Insufficient camera resolution might be the problem of both these implementations. In each case, a camera with a better resolution and a better image quality, can fix these problems.

7.3 Implementation of client application with A* algorithm

The verification of a client application ended well. During the testing of the implementation, the testing student's routine met with no problems. My implementation of student's routine is implemented in JAVA programming language with recommended socket implementation from language documentation.

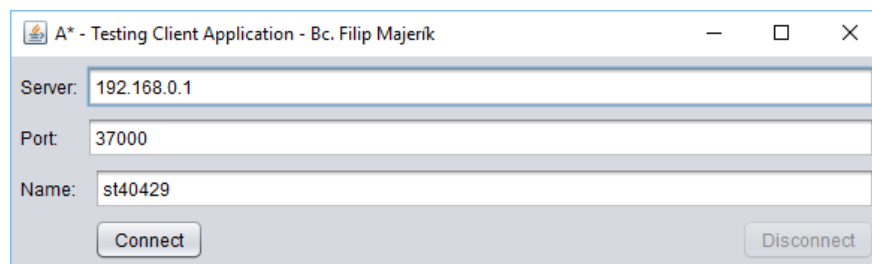


Figure 55 - Simple routine GUI

The check of path-plan is done visually in the high-level control system. I tried to find several routes for a maze and it was performed very well. During this, I tested the path-plan to sequence steps transformation. It is done well. Further, while I tested this part of the high-level control system, I decide to implement the simulation mode, where no robot and camera is needed. In this simulation - mode you can analyze the maze from an image file and the robot will move only on the screen.

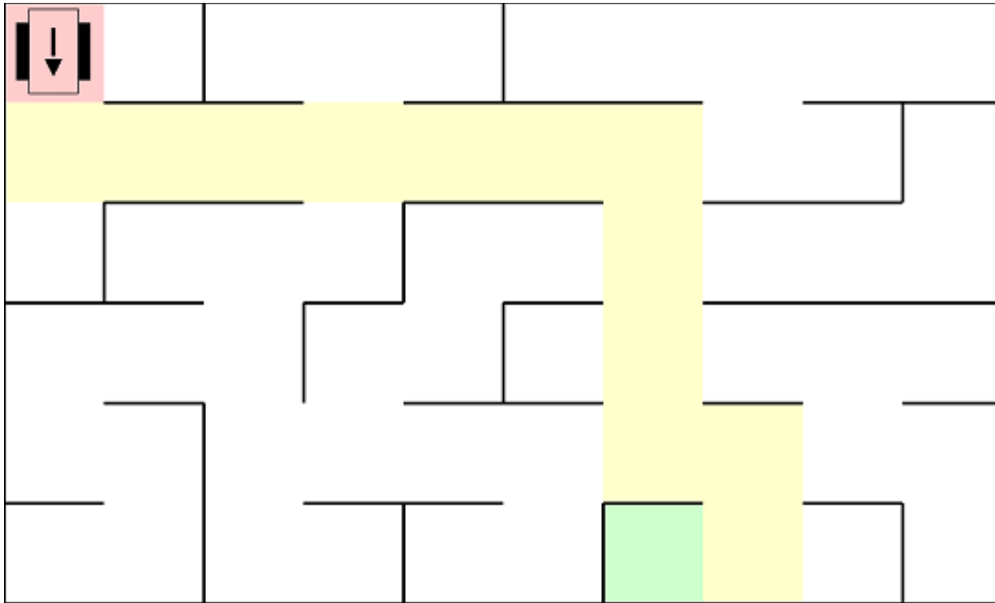


Figure 56 - Example of generated path by testing routine

```

Waiting for solver..
To: 57
From: 1
Dimension: java.awt.Dimension[width=10,height=6]
OK submitted
PROBLEM: 1 -> 57
Time:Sat Apr 29 01:28:30 CEST 2017
From: 1 >> 57
Soloution:: 1,11,12,13,14,15,16,17,27,37,47,48,58,57

```

Figure 57 - Example output during the communication with routine

The generated sequence of steps, for the path, which is shown on Figure 55, is:

FORWARD → LEFT → FORWARD → FORWARD → FORWARD → FORWARD →
FORWARD → FORWARD → RIGHT → FORWARD → FORWARD → FORWARD
→ LEFT → FORWARD → RIGHT → FORWARD → RIGHT → FORWARD

The correctness of this sequence of steps is verified by the simulation robot module which accepts the sequence of step as input data for simulation.

Conclusion

In this thesis I tried to implement the high-level control system for autonomous mobile robot. The maze detection, simulation mode, GUI interface, communication protocol, and test student routine were all successful elements of this thesis. I am especially satisfied with implementation of a simulation mode for testing this software without a robot. Communication with a robot is not a part of this thesis.

Unfortunately, the robot detection did not work according to preconceived ideas. These problems appeared during the implementation of the teaching aid. Such problems can be eliminated by further research.

As is mentioned, it is maybe a problem with a camera which is used. Without replacing the camera, I suggest to use a simple shape detection with a special mark for recognition of a rotation of the robot.

For recapitulation, these problems are - ambient light in class where teaching aid is used, impossibility to detect the last corner of the maze without the marker, problems with camera connection using secured RTSP protocol, and detection of robot position and rotation using a cascade classifier or the key points. For almost all problems, possible solutions are proposed, in this thesis.

This teaching aid can be used for demonstrating path-planning and testing student's routines. The simulation mode gives to student's sufficient possibility to check their correctness of a path-planning routine implementation.

I wish to finish this teaching aid in the near future. The refactoring and optimization of some software parts are needed, with increases precision of the maze detector.

Literature

Acharya, Tinku and Ray, Ajoy K. 2005. *Image Processing: Principles and Applications*. Tuscon, Arizona: John Wiley & Sons, Inc., 2005. ISBN: 978-0-471-71998-4.

Cormen, Thomas H. 2009. *Introduction to Algorithms*. London: MIT Press, 2009. ISBN: 978-0-262-03384-8.

Csetverikov, Dmitrij. 2003. *Basic Algorithms for Digital Image Analysis*. [Web] Budapes, Hungary: Institute of Informatics Eötvös Loránd University, 2003.

Das, Vinu and Thankachan, Nussy. 2011. *Computational Intelligence and Information Technology*. Pune, India: Springer, 2011. ISBN: 978-3-6422-5734-6.

Even, Shimon. 2011. *Graph Algorithms (2nd ed.)*. Cambridge: Cambridge University Press, 2011. ISBN: 978-0-521-73653-4.

Goyal, Aditya, Bijalwan, Akhilesh and Chowdhury, Kuntal. 2012. *A Comprehensive Review of Image Smoothing*. s.l.: International Journal of Advanced Research in Computer Engineering & Technology, 2012. ISSN: 2278-1323.

Hazewinkel, Michiel. 1995. *Encyclopaedia of Mathematics*. Dordrecht: Springer Science, Business Media, 1995. ISBN: 978-0-7923-2975-6.

Huffman, David A. 2007. *A Method for the Construction of Minimum-Redundancy Codes*. s.l.: IEEE, 2007. ISSN: 0096-8390.

Chandrakala, M. and Durga Devi, P. 2016. *Threshold Based Segmentation Using Block*. Telangana, India: International Journal of Innovative Research in Computer, 2016. ISSN: 2320-9801.

Kanniah, Jagannathan, Ercana, M. Fikret and Calderon, Carlos Acosta. 2013. *Practical Robot Design: Game Playing Robots*. New York: CRC Press, 2013. ISBN: 978-1-4398-1033-0.

Kašík, Marek. 2006. Potlačování vad obrazu vzniklých při snímání ve fluorescenční mikroskopii. *Masaryk University*. [Online] 2006. [Cited: 05 06, 2017.] https://is.muni.cz/th/50771/fi_r/rigorous.pdf.

Kaufman, Arie. 1993. *Rendering, Visualization and Rasterization Hardware*. Springer Science, 1993. ISBN: 978-3-540-56787-5.

Latombe, Jean-Claude. 1991. *Robot Motion Planning*. Stanford: Springer Science, 1991. ISBN: 978-1-4615-4022-9.

Lyytinen, Kalle, et al. 2007. *Design Requirements Engineering: A Ten-Year Perspective*. Cleveland: Springer-Verlag Berlin Heidelberg, 2007. ISBN: 978-3-540-92966-6.

- Mohr, Austin. 2007.** *Quantum Computing in Complexity Theory and Theory of Computation.* [http://www.austinmohr.com/Work_files/complexity.pdf] Carbondale: Southern Illinois University at Carbondale, 2007.
- Monga, Priyanka M. and Saurabh., Ghogate A. 2015.** *Scrutiny on Image Processing.* Badnera : s.n., 2015. ISSN: 2277-128X.
- Phillips, Dwayne. 2000.** *Image Processing in C.* Lawrence, Kansas: R & D Publications, 2000. ISBN: 0-13-104548-2.
- Qu, Zhong and Li, Zhang. 2010.** *Intelligent Human-Machine Systems and Cybernetics.* Nanjing: IEEE Computer Society, 2010. ISBN: 978-0-7695-4151-8.
- Rafael, Gonzales C. and Woods, Richard Eugen. 2008.** *Digital Image Processing.* s.l.: Pearson/Prentice Hall, 2008. ISBN: 978-01-3505267-9.
- Rhody, Harvey. 2005.** *Lecture 2: Geometric Image Transformations.* Rochester NY: Center for Imaging Science, 2005.
- Russell, Stuart J. and Norvig, Peter. 2010.** *Artificial Intelligence - A Modern Approach.* New Jersey: Pearson Education, Inc., 2010. ISBN: 978-0-13-604259-4.
- Říha, K. 2012.** *Pokročilé techniky zpracování obrazu, skripka.* Brno: VUT Brno, 2012. ISBN: 978-80-214-4894-0.
- Salomon, David and Motta, Giovanni. 2009.** *Handbook of Data Compression.* s.l.: Springer, 2009. ISBN: 1-84882-902-7.
- Sedgewick, Robert and Wayne, Kevin. 2011.** *Algorithms.* Boston: Pearson Education, Inc., 2011. ISBN: 978-0-321-57351-3.
- Siegwart, Roland and Nourbakhsh, Illah R. 2004.** *Introduction to Autonomous Mobile Robots.* London: MIT Press, 2004. ISBN: 0-262-19502-X.
- Sonka, Milan, Hlavac, Vaclav and Boyle, Roger. 2008.** *Image Processing, Analysis, and Machine Vision.* Toronto: Thomson Learning, 2008. ISBN: 978-0-495-24428-7.
- Szeliski, Richard. 2011.** *Computer Vision.* London: Springer London, 2011. ISBN: 978-1-84882-935-0.
- Škrabánek, Pavel and Doležel, Petr. 2014.** *Attractive Robot's design suitable for image processing.* Pardubice : University of Pardubice, 2014.
- Škrabánek, Pavel and Majerík, Filip. 2017.** *High-Level Control System for a Path-Planning Teaching Aid.* Pardubice: University of Pardubice, 2017.
- Škrabánek, Pavel. 2015.** *Labyrinth Arrangement Analysis Based on Image Processing.* Brno: Springer, 2015. ISBN: 978-3-319-19824-8.

Škrabánek, Pavel, Mariška, Martin and Doležel, Petr. 2015. *The time optimal path-planning of mobile robots motion respecting the time cost of rotation.* Pardubice: s.n., 2015. ISBN: 978-1-4673-6627-4.

Škrabánek, Pavel, Vodička, Pavel and Yildirim-Yayilgan, Sule. 2016. *Control System of a Semi-Autonomous Mobile Robot.* Pardubice : IFAC-PapersOnLine, 2016.

Teh, C. H. and Chin, R. T. 1989. *On the Detection of Dominant Points.* Madison, USA: Institute of Electrical and Electronics Engineers, 1989. ISSN: 0162-8828.

Wolberg, George. 1988. *Geometric Transformation Techniques for Digital Images: A Survey.* New York: Department of Computer Science, Columbia University, 1988.

Attachment A – Example usage of communication protocol

Direction	Data
	<i>Server awaiting connection...</i>
Client->Server	<i>Request for connection...</i>
Server->Client	Name?
Client->Server	st40429
Server->Client	ping
Client->Server	ping
Server->Client	solveproblem
	0,1,0,0,0,1,0,0,0,0...
	...
	0,0,0,0,1,0,0,0,0,0...
	dataok
Client->Server	from
Server->Client	1
Client->Server	to
Server->Client	17
Client->Server	dimension
Server->Client	4,4
Client->Server	ok
Client->Server	solution
Client->Server	1,2,5,3,6,...,17
Server->Client	ping
Client->Server	ping
Server->Client	QUIT
	<i>Client disconnected...</i>

Attachment B – Accompanying CD

The accompanying CD contain:

- Master thesis in electronic version (PDF and DOCX – Word 2013)
- Source code of HLCS application
- OpenCV files for run the application
- Testing image of maze (JPG)
- Two example adjacency matrix sources (TXT)