Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Visual navigation of mobile robot

Master's Thesis

2013 Bc. Ondrej Mikuláš

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Visual navigation of mobile robot

Master's Thesis

Study Program: Informatics
Branch of Study: 9.2.1 Informatics (2508)
Department: Department of Computer Science
Supervisor: RNDr. Andrej Lúčny PhD.

Bratislava, 2013 Bc. Ondrej Mikuláš

66678552

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Bc. Ondrej Mikuláš
Study programme:	Computer Science (Single degree study, master II. deg., full time form)
Field of Study:	9.2.1. Computer Science, Informatics
Type of Thesis:	Diploma Thesis
Language of Thesis:	English

Title: Visual navigation of mobile robot

Aim: The main goal is design of mobile robot control (notebook on two wheeled gear equipped with camera), which operates in bureau premises. Robot processes image from camera and generate non-colliding actions performed by the gear. The recommended approach to the robot control is based on the multi-agent modeling of mind.

Literature: Davies, E.R.: Machine Vision, Elsevier, 2004
Publikácie školiteľa

Keywords: computer vision, cognitive vision, artificial cognitive system, robotics

Supervisor:	RNDr. Andrej Lúčny, PhD.
Department:	FMFI.KI - Department of Computer Science
Head of department:	doc. RNDr. Daniel Olejár, PhD.
Assigned:	08.10.2010

Approved: 25.10.2010 prof. RNDr. Branislav Rovan, PhD.
Guarantor of Study Programme

....

Student Supervisor

66678552

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Bc. Ondrej Mikuláš
Študijný program:	informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor:	9.2.1. informatika
Typ záverečnej práce:	diplomová
Jazyk záverečnej práce:	anglický

Názov:	Vizuálna navigácia mobilného robota
Cieľ:	Cieľom práce je navrhnúť riadenie mobilného robota (notebook s kamerou na podvozku), ktorý operuje v kancelárskych priestoroch. Robot spracúva obraz a vydáva podvozku nekolízne pokyny. Odporúčaný prístup k riadeniu je založený na multiagentovom modelovaní mysle.
Literatúra:	Davies, E.R.: Machine Vision, Elsevier, 2004 Publikácie školiteľa
Kľúčové slová:	počítačové videnie, kognitívne videnie, umelý kognitívny systém, robotika

Vedúci:	RNDr. Andrej Lúčny, PhD.
Katedra:	FMFI.KI - Katedra informatiky
Vedúci katedry:	doc. RNDr. Daniel Olejár, PhD.
Dátum zadania:	08.10.2010

Dátum schválenia: 25.10.2010 prof. RNDr. Branislav Rovan, PhD.
garant študijného programu

....
študent vedúci práce

I hereby confirm that I have independently composed this Master thesis and that no other that the indicated aid and sources have been used.

. . . .

Acknowledgements

First of all, I am deeply grateful to my supervisor Dr. Andrej Lúčny for his valuable help and guidance during the development of my thesis. I also thank Milan Plžík for helping me with the linux driver for chassis of the mobile robot. Last, but not least, I am really grateful to my family, who has supported me and helped me during my studies. A special thanks goes to you all of my friend, you mean very much to me.

Ondrej Mikuláš

Abstrakt

Autor:	Bc. Ondrej Mikuláš
Názov práce:	Vizuálna navigácia mobilného robota
Univerzita:	Univerzita Komenského v Bratislave
Fakulta:	Fakulta matematiky, fyziky a informatiky
Katedra:	Katedra informatiky
Vedúci diplomovej práce:	RNDr. Andrej Lúčny PhD.
Rozsah práce:	48 strán
Dátum:	máj 2013

V tejto práci sme sa venovali vizuálnej navigácii mobilného robota. Naša práca využíva Agent-Space architektúru [Lúčny, 2004], ktorá je vhodná na simuláciu živých systémov. Demonštrujeme, že navigácia môže byť realizovaná aj pomocou obyčajnej kamery a jednoduchých aktuátorov, bez použitia ďalších senzorov. Využili sme vlastnosť architektúry, ktorá nám umožňuje kombinovať viacero jednoduchých správaní do komplexného systému v ktorom spolupracujú.

Celkovo naša práca vyústila v aplikáciu, ktorá realizuje navigáciu v rovnej chodbe a v križovatke chodieb.

KĽÚČOVÉ SLOVÁ: počítačové videnie, kognitívne videnie, umelý kognitívny systém, robotika

Abstract

Author:	Bc. Ondrej Mikuláš
Title:	Visual Navigation of mobile robot
University:	Comenius University in Bratislava
Faculty:	Faculty of Mathematics, Physics and Informatics
Department:	Departement of Computer Science
Supervisor:	RNDr. Andrej Lúčny PhD.
Number of Pages:	48
Date:	May 2013

In this thesis we study visual navigation control of mobile robot. We are using Agent-Space architecture [Lúčny, 2004] that is considered to be suitable for simulation of living systems. In our work we demonstrate, that navigation control can be realized by using ordinary camera and actuators with no additional sensors. We used feature of this architecture which enables combination of several behaviors to complex system in which they cooperate.

Overall our work resulted in application which perform navigation in the straight corridor and in the junction of corridors.

KEYWORDS: computer vision, cognitive vision, artificial cognitive system, robotics

Contents

Chapter 1

Introduction

The problem of building mobile robot, which navigates in its environment is very difficult. There are many problems that need to be solved like design and construction of robot, selection of sensors, control architecture. Then one needs to create control mechanism and select proper navigation techniques. It is almost impossible to create a robot which operates in all kinds of environments. One must choose a specific environment and design a robot according to conditions present in it.

1.1 Motivation

We found motivation for our thesis in nature. Almost all moving creatures in nature have some kind of a navigation system, which enables them moving in an environment without a collision. We would like to build a navigation control and try to simulate navigation systems in nature. Therefore we decided not to use artificial sensors which are common in many navigation systems. We are using just visual information from ordinary camera. We want to create a "mind" which would realise this navigation.

We can probably say that nobody knows how exactly mind works and from which parts it is composed of. There are many models of mind which are inspired by experiments in psychology, cognitive science and other areas. We are focusing on one, where mind is composed from agents and complex behavior is achieved by their cooperation. This model is described in Minsky's well-known book [Minsky, 1986]. We can assume that navigation control in living creatures is not monolytical, but distributed.

1.2 Our Goal

In this thesis we focused on internal structure which realises navigation. We are not trying to build flawless navigation or use best available technologies. Instead, we are trying to build it using specific Agent-Space architecture [Lúčny, 2004] that is considered to be suitable for building complex systems as navigation of robot is. This architecture has biological relevance and is suitable for simulation of living systems.

During work on this thesis we were not using typical components widely used in mobile robotics, such as precise motors or high-quality cameras. These are used for boosting performance in order to simplify navigation control. We want to demonstrate that navigation control can also be performed with the low quality camera and actuators.

The other reason why we chose the Agent-Space architecture is its succesful employment in several areas:

- Pedestrian recognition
- Industrial sphere - complex control of monitoring systems
- Biological simlation of animals behavior.

We will describe Agent-Space architecture in further chapters. We hope that our result will be good enough to allow the architecture to be used for building visually navigated systems.

1.3 Outline

This thesis is composed of several parts. In chapter 2 there is a classification of control architectures and description of Agent-Space architecture. In chapter 3 there are described methods and algorithms from computer vision, which we used in the navigation control. Our approach to the visual navigation control which is realised by Agent-Space architecture is in chapter 4. Implementation details and experimental results, which we get using our navigation control are in chapter 5.

Chapter 2

Related Work

In this chapter, works related to the thesis are mentioned. Visual navigation is source of countless research contributions from domain of control and vision. In [Bonin-Font et al., 2008] there is general classification of navigation techniques. We can classify our approach as mapless indoor navigation by decentralized multi-agent control system.

Our approach is original, thus it is difficult to compare it with other approaches. As we mentioned in chapter 1, we are using low quality components and developing the system based just on the information from an ordinary camera (no Kinect camera, or industrial cameras). It is not related to systems which are using artificial landmarks for navigation or state-of-the-art hardware components. We focused on system which can be found in living system, where complicated tasks are performed with the simple actuators.

As far as we know there were no attempts to create navigation control using Agent-Space architecture. This architecture has connection to behavioral architectures, especially it enables the use subsumption architecture. In the following there is classification and description of control architecture taken from [Busquets, 2003]. Agent-Space architecture and its features are explained in section 2.2

2.1 Control Architectures

A robot which is working in unknown environment has to be able perceive environment, reason about it and select proper action to achieve goals. The way in which this process is done is defined by the control architecture.

The main difference between architectures proposed in the past years relies on whether they are more deliberative or reactive. We can identify three main approaches: hierarchical architectures, behavior-based architectures and hybrid architectures.

2.1.1 Hierarchical architectures

These architectures are also named deliberative control architectures. They are based on top-down methodology, where problem is decomposed to set of modules, sequentially organized. Information from the perception module is sent to modelling module, which kept an internal model of the environment. It initiates the planning module, which using internal model for selecting action. Response to the environment is implemented in execution module, which select proper commands for actuators.

This architecture is used especially in the environment which are known and where robot is created for special purpose (e.g. industrial robots in factories with marked paths, or for robots which are using map of environment for navigation). However, when the task is performed in unknown, unpredictable environment, they fail to succeed, as the planning is usually out of date during execution of action chosen by planning module.

2.1.2 Behavior-based architectures

Behavior-based robotics [Arkin, 1998] appeared in the 1980s in response to the traditional hierarchical approach. Brooks proposed to tightly couple perception to action, and thereby provide a reactive behavior that could deal with any unpredicted situation the robot may encounter.

Behavior-based robotics is a bottom-up methodology, inspired by biological studies, where a collection of behaviors acts in parallel to achieve independent goals. The overall architecture consists of architecture consists of several behaviors reading the sensory information and sending actuator commands to a coordinator that combines them in order to send a single command to each actuator as shown in Figure 2.1.

The most representative of such architectures are Brooks' subsumption architecture [Brooks, 1986] , Maes' action selection [Maes, 1989] and Arkin's motor schemas [Arkin, 1989]. Since then, a many other architectures have been proposed.

Subsumption architecture: The Subsumption architecture, designed by Rodney Brooks [Brooks, 1986], was the first of the Behavior-based architectures. In

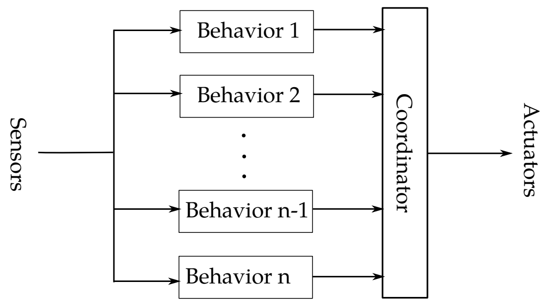


Figure 2.1: Behavior-based architecture

this architecture each behavior is represented as a separate layer, having direct access to sensory information. These layers are organized hierarchically, and higher levels are allowed to subsume, hence the name, lower ones. Competitive mechanism based on priority are used there.

The hierarchical organization permits an incremental design of the system, as higher layers are added on top of an already working control system, with no need of modifying the lower levels.

The main strengths of this architecture are its incremental design methodology, which makes it easy and intuitive to build a system, its hardware retargetability and the support for parallelism since each layer can run independently and asynchronously.

This architecture has limitation, since connections between layers are hard-wired, so they cannot be changed during execution, thus not allowing on-the-fly adaptability of the system to changes in the environment.

In [Lorigo et al., 1997], autonomous vision-based obstacle avoidance system is proposed. The system consists of three independent modules which are reactive and store no information about obstacles. It is rather using the images directly. Median method is used for fusion information from modules. This system draws from the behavior-based subsumption architecture approach for combining routines.

2.1.3 Hybrid Architectures

Although it has been widely demonstrated that behavior-based architectures effectively produce a robust performance in dynamic and complex environment, they are not always the best choice for some tasks. Sometimes the task to be performed needs robot to make some deliberation and keep a model of the environment. But behavior-based architectures do avoid this deliberation and modeling. Thus, a compromise between these two completely opposite views must be reached. This is what hybrid architectures achieve.

In these hybrid architectures there is a part of deliberation, in order to model the world, reason about it and create plans, and a reactive part, responsible of executing the plans and quickly reacting to any unpredicted situation that may arise.

There are several approaches based on multi-agent systems which use fuzzy logic for combination outputs of several agents and using hybrid architecture. Busquets's approach [Busquets et al., 2003] applies decentralized multi-agent systems, where agents bids for action and action with highest bid is performed. Broadcast of information is decentralized, not hierarchical. This approach use also fuzzy logic about angles and distances to perform localization. In this approach are used also artificial landmarks in environment. Opposite to our solution authors aren't using subsumption architecture. Agents here have no hierarchy.

Next approach [Ono et al., 2004] used additional infrared sensors. Authors used blackboard as central repository for all shared information. For localization authors used fuzzy logic. Next approaches [Innocenti et al., 2008] and [Olajubu et al., 2011] used fuzzy logic and multi-agent approach as well.

2.2 Agent-Space architecture

As we mentioned in Chapter 1 we build navigation using Agent-Space architecture. This architecture was proposed in [Lúčny, 2004] as architecture for building complex systems. This architecture is based on multi-agent system and is one of the architectures where modules communicate with each other.

We chose this architecture, because it is suitable for real-time software engineering. Next reason is, that it has biological relevancy. There are principles which we can build on during building navigation control. These principles are described in further parts.

2.2.1 Structure

Control system consists of reactive agents, which communicate indirectly through communication tool called space. It is blackboard like communication, which the agents employ for mutual data exchange.

Space

Space contains data organized in entities, called blocks. All the agents have access to these blocks and can read data from and write data into them. The read and write operations are services provided by the space for agents. Agents access to blocks through their names. Agent A cannot send message directly to the agent B, but can leave message in some block in the space and agent B can read that message. This operation is shown in Figure 2.2. Once data are written in block, the agent, which wrote the data, loses control of it. Other agents can read it or even rewrite it.

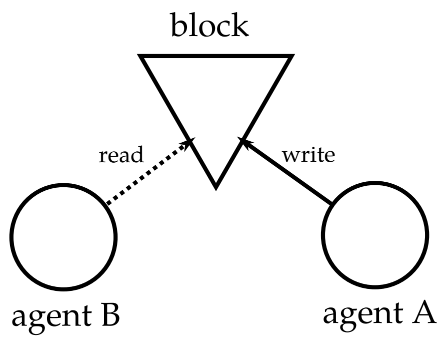


Figure 2.2: Communication between agent A and agent B.

Write operation rewrites data written during previous write operation. On the other hand, read operation does not take data from the block. The agent, which writes the data can specify validity of the data. They can be valid for some particular time, or they can be written in block forever. When validity of a block expires, data stored in it disappears and the block becomes empty.

This architecture does not require any special form of data which are stored in space. Agent, which reads the data must know the type of the data written in the block.

The read and write operations are not the only services provided by space. Other important service is to enable agent, which is writing the data, to specify the priority of data. Priority serves as mechanism for protection of data. Data written in the block with higher priority are not changed by the agent, which writes a data to the block with lower priority.

Very important service provided by space is agents ability to register trigger within the block. When the block is changed, all agents who registered trigger in the block are woken up from its sleep. However trigger is not inevitable mechanism for waking up an agent from sleep. Agents can regularly read block content after employing a timer.

Reactive agents

All the application codes are concentered in relatively small and simple processes called reactive agents. A reactive agent after initialization performs endless cycle *sense* - *select* - *act* to reach relatively simple goal. Situation is better pictured in Figure 2.3.

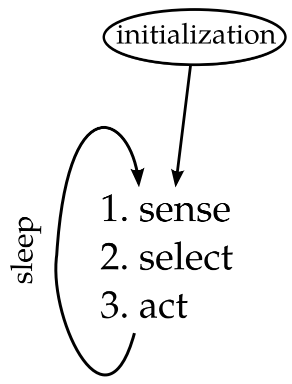


Figure 2.3: Lifetime of the reactive agent.

This cycle is still in operation at given frequency and there is pause (sleep) after each course of the cycle.

- In the sense phase the reactive agent reads data from the space.
- In the select phase the reactive agent selects the action to achieve certain goal.
- In the act phase the reactive agent writes the calculated data back into space.

The agent has two options to be woken up from sleep. It can have trigger registered in the particular block and after change of block, it is woken up. Second option is that agent has timer which notifies him after specified period of time. Courses of individual agents are not synchronized and vary from agent to agent.

Agents presented in Agent-Space architecture are reactive. The response of such agents is a pure reaction to the state of space in sense phase. These agents are stateless and internally are holding no data from the previous cycle. Abilities of such agents are very poor when they are isolated. On the other hand we can combine these agents and reach complex behavior.

2.2.2 Advantages and Disadvantages

Agent-Space architecture brings both advantages and disadvantages. In this part are described its basic features and their effect on real-time engineering.

No deadlocks

Systems which use Agent-Space architecture are absolutely resistant to deadlocks. No process can be blocked endlessly, because non-blocking message passing is established. There is guarancy of not getting stuck just for building modules, not for overall system. A robot controlled by such a system can still get stuck in its environment where it operates.

Data flow

In Agent-Space is used non-traditional way of data exchange (Figure 2.4). Neither the producer nor the consumer knows who is on the other side. They know the block only. It differs from traditional ways of data exchange, where a producer sends data to consumer, or consumer requests data from the producer (client-server). This

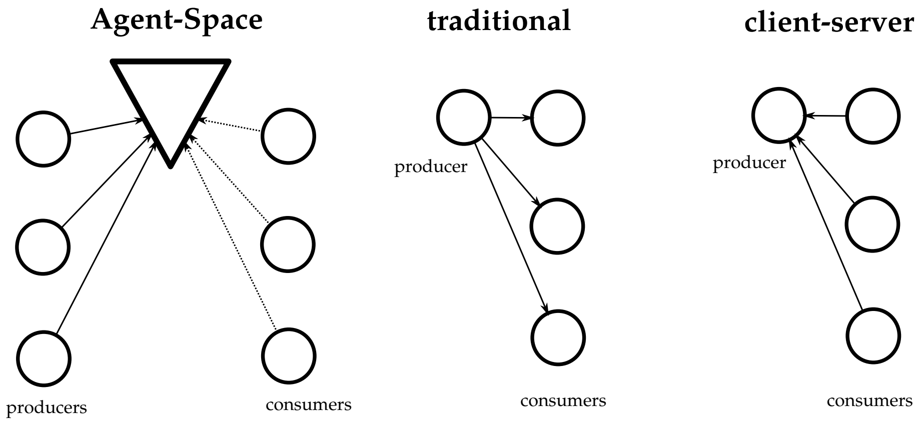


Figure 2.4: Data flow

allows better distribution of codes related to application domain into individual processes.

Recovery from errors

Special type of data flow enables restart any agent whenever it is requires, without any impact on other agents. This feature is crucial for recovery, when an agent crashes. In this situation one needs just start crashed agent again.

The system can be easily started in spite of the fact that other agents, it was communicating with, have not started yet.

Decentralization

There is no scheduling program which specifies whether an agent has to act and when. Agents are completely autonomous. Their cooperation is not defined explicitly in their code, but emerges from interaction among agents.

This architecture is suitable for development of decentralized systems, since there are no agents with exclusive position. Such as "control agents" or "upper agents". However it is possible to create central control system using this architecture, but it is not natural. The main disadvantage of the decentralization is, that we lose perfect control over the system. It makes system very hard to debug. On the other hand we are able to handle most of details locally without any overall plan.

Data inconsistency and redundancy

It is usual, that change of block has influence on many other blocks indirectly (via other agents). It takes some time to spread change of the block through system. During the propagation of change, local inconsistency takes it place. However after some time, the blocks become stable again and their data are mutually consistent.

Redundancy appears when an agent must regularly check whether the change in the source has appeared. It can execute plenty of instructions to produce none behavior.

Ability to modify

The phase of development is very important during building complex system. This architecture intends to simplify developer's job, even at expenses caused by operational disadvantage like redundancy. Ability to modify is crucial during incremental development. Each phase of incremental development can be recognized as a modification.

System can be easily extended using new agents and connecting them to already implemented ones. These new agents will read values from blocks from older version (Figure 2.5).

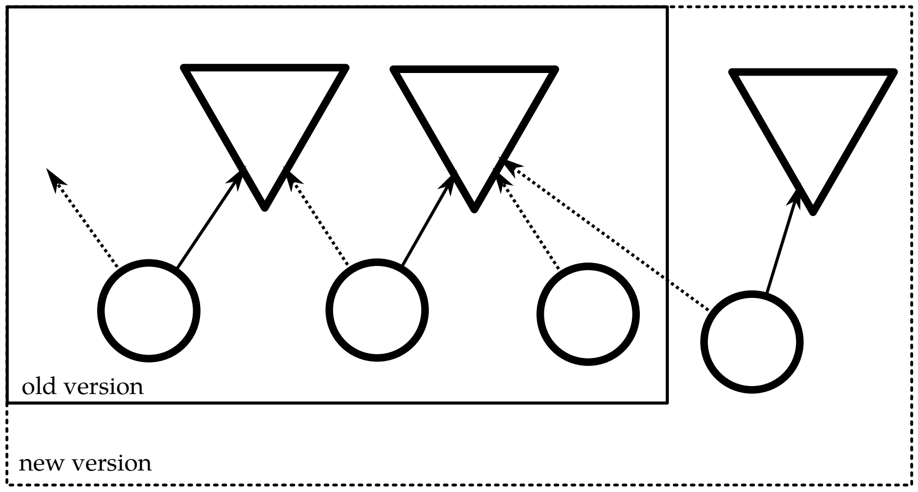


Figure 2.5: Modification by adding new agents.

2.2.3 Biological Relevancy

We chose Agent-Space architecture, because its features have similarity with living systems.

Hierarchy

In nature we can observe hierarchy between its building parts. Agent-Space architecture has ability to express hierarchy within system. It can express an agent as a multi-agent system of lower description level. This is crucial for building hierarchical systems which contain several levels. The key concept is substitution of an agent acting in space by such a group of agents (Figure 2.6). There must be agent which provides mapping between blocks in higher-level and lower-level.

Activity

It is typical for living systems, that they don't stop their activity in special or critical conditions. They still act in some way. Hence their response to such condition is not necessarily reasonable, they still act in some way. Systems based on Agent-Space architecture have the same attribute. In environment in which they were tested,

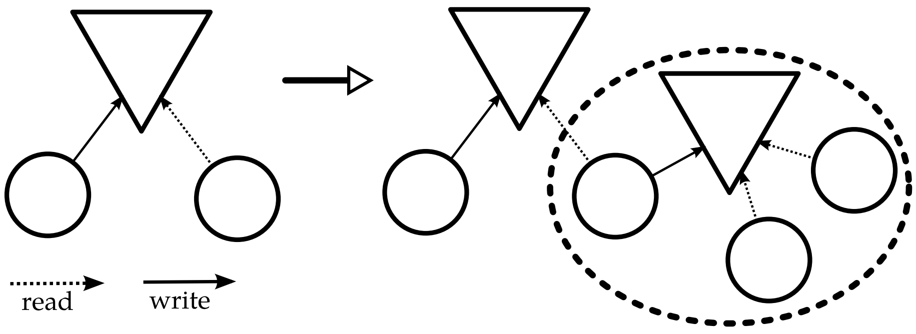


Figure 2.6: Realization of hierarchy.

they act reasonable, but in unknown environment they don't throw exception or get stuck.

Decentralization

In living systems we can observe parallelism. If we damage living system in some way, it is still active. Our hypothesis is, that in there are no central modules in living systems which wake up other modules to activity. We think that all part of system are active, but effect on behavior is suppressed for some of them. This property is very common in Agent-Space, especially when the agent rewrites data in some block with higher priority and previous data with lower priority are deleted.

Chapter 3

Methods from computer vision

In this chapter we describe the methods of computer vision that we used in our work. Our goal is to create navigation control in the indoor office environment. We are using methods which have good response in this environment. The following methods are explained in this chapter:

- detection of line segments
- detection of vanishing point
- matching between images

Most of the computer vision algorithm are taken from [Davies, 2012]. When the other source was used, it is written explicitly.

3.1 Detection of vanishing point

In the navigation control we used vanishing point for navigation in the straight corridor. The simplest definition is that a vanishing point is a point in the image plane to which parallel lines of the scene seem to converge (Figure 3.1). This point corresponds to the three-dimensional direction in space. There can be more vanishing points in the image, and also vanishing point can be out of the image plane, or even situated in infinity.

From now on, when we say that robot is adjusting its position according to the vanishing point detected in the image, we mean that it is adjusting its position according point where lines parallel with corridor meet. When the corridor is straight

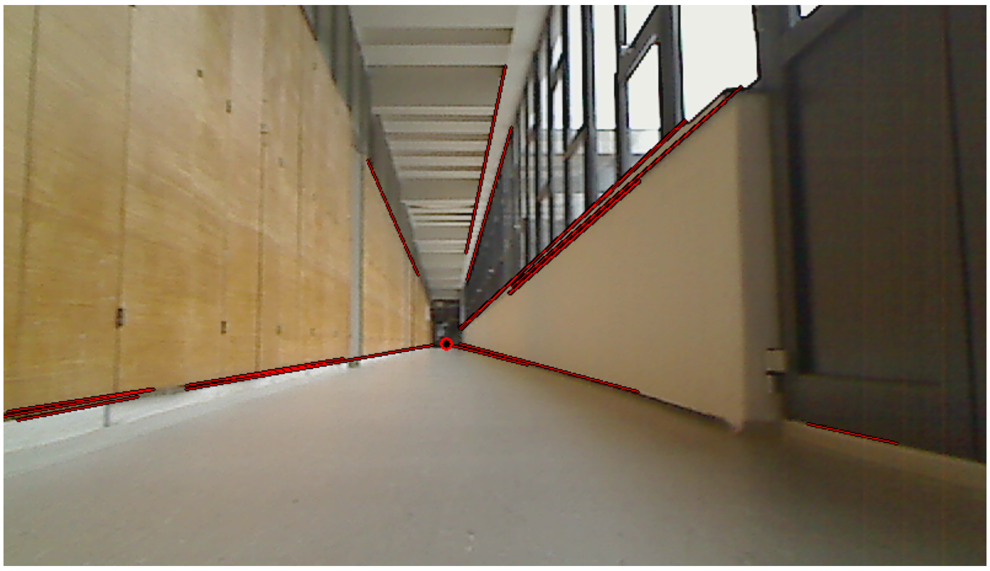


Figure 3.1: Vanishing point in straight corridor.

and the image is taken from the center of it, the vanishing point of the corridor lies in the center column of the image.

The detection of vanishing points is usually carried out in two stages. At first, all lines are located in the image. Afterwards lines passing through common points are selected. These points are interpreted as vanishing points.

3.1.1 Detection of line segments.

As we already mentioned, we are performing navigation in indoor corridors. In this environment lines and line segments are ubiquitous. Lines and line segments which we want use for the vanishing point detection are parts of edges. We use the term edge for a set of points in the image, where brightness changes sharply. These are typically organized as a set of curved line segments. Examples of edges are shown in Figure 3.2. These are edges which were used for line segment detection. We do not assume the presence of particular edge type in the images. Rather we are using general method for the edge detection. Typical edges in corridors are baseboard lines between the floor and the wall (usually thin black stripe was present there (Figure 3.2a, 3.2b, 3.2d)). Other edges which were used for line segment detection were between side cabinets and wall (Figure 3.2e).

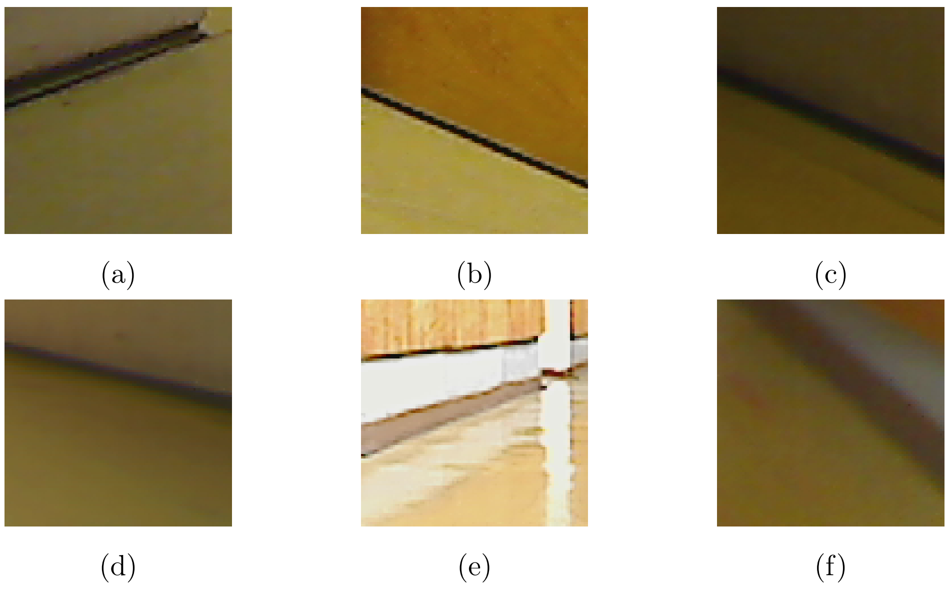


Figure 3.2: Examples of edges in corridors.

Canny edge detector

Since images acquired from camera are noised, we used Gaussian blurring to reduce noise from the image in preprocessing. Than we converted image to grayscale and applied Canny edge detector [Canny, 1986]. It involves a number of stages of processing. The Canny edge detector provides evidence for each pixel, whether it is the edge or not (Figure 3.3b). Output from Canny edge detector is used as the input for following method which finds lines segments.

Progressive Probabilistic Hough transform

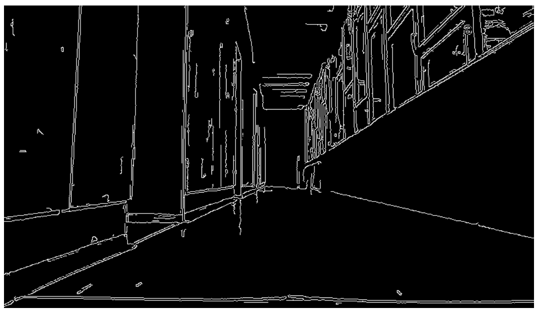
The Hough transform provides detection of lines (Figure 3.3c). The main advantage of using Hough transform is that it is tolerant of gaps in lines affected by noise in the image.

The Hough transform is not a fast algorithm for line detection in large images. It can also detects only infinite lines that pass through the whole image. For our purposes the finite line segments are better. Therefore we are using evidence from Canny edge detector as an input for PPHT [1] [Matas et al., 2000], which is the faster variant of Hough transform. Line segments detected by PPHT are shown in Figure 3.3d. Another advantage of line segments to lines is that, lines present in corridors contain usually many gaps, caused by doors, or perpendicular corridors. These line

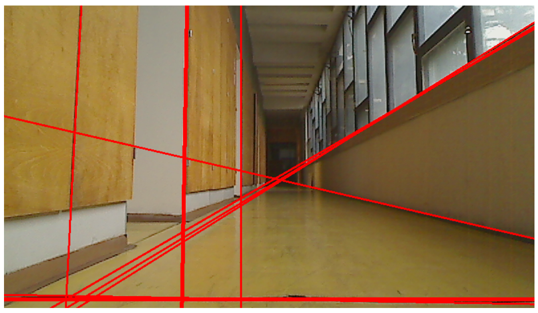
[1]Progressive Probabilistic Hough transform

segments are used for estimation of the vanishing point.

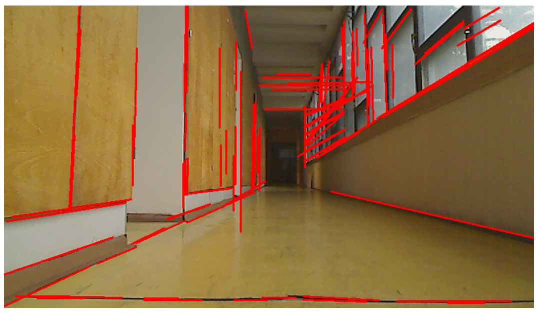
(a) Original Image



(b) Canny edge detector evidence



(c) Straight lines by Hough transform



(d) Line segments by PPHT

Figure 3.3: Detection of line segments.

3.1.2 Estimation of the vanishing point.

After acquisition of line segments we used robust vanishing point estimator proposed in [Nieto and Salgado, 2010]. This estimator can detect multiple vanishing points, and offer trade-off between accuracy and efficiency being able to operate in real time for video sequences.

The key element of this approach is a robust scheme based on MLESAC[2] algorithm [Torr and Zisserman, 2000], which is extension to RANSAC[3] algorithm [Fischler and Bolles, 1981]. RANSAC is an iterative method to estimate parameters of a mathematical model, from a set of observed data, which contains outliers. In our case outliers are line segments, which are not meeting in vanishing point. RANSAC proceeds iteratively in the two main steps. First, minimal sample subset is randomly selected in order to generate a hypothesis of the vanishing point (2 line segments). Afterwards consensus subset of line segments is selected according

[2]Maximum Likelihood Estimator Sample and Consensus

[3]Random Sampling and Consensus

hypothesis, these are line segments which distance from hypothesis vanishing point is under error threshold δ.

These two steps are repeated until the probability of finding better consensus set is below the convergence threshold. MLESAC algorithm defines a more accurate system for estimation of consensus set.

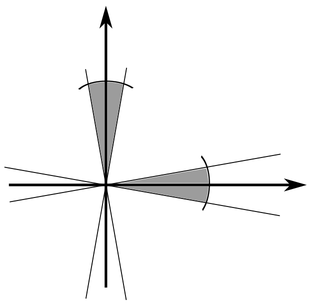
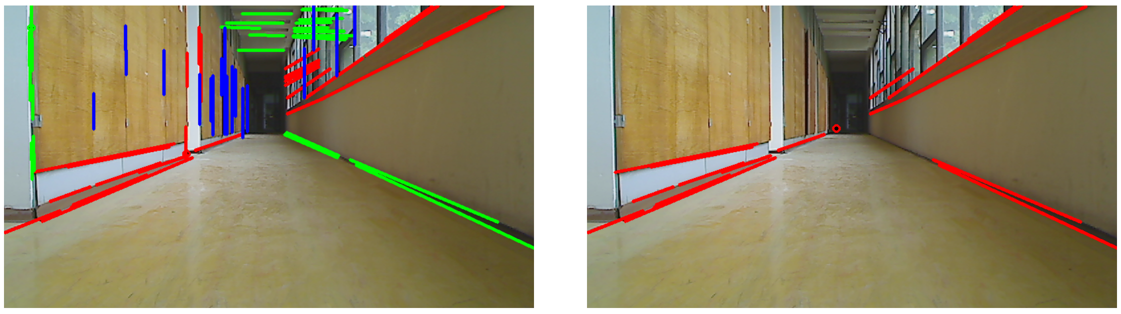


Figure 3.4: Removing of lines with predefined slope.

In order to find the vanishing point suitable for tracking we removed all horizontal and vertical line segments, which slope is in the range $[-10°, 10°] \cup [80°, 100°]$ as shown in Figure 3.4. These line segments correspond to horizontal or vertical edges of doors, windows or floor. We consider them as the outliers for estimation the vanishing point of corridor. Then we applied the algorithm for vanishing point estimation with the remaining line segments. Algorithm divides the line segments to inliers, which meets in vanishing point and outliers which are remaining line segments that were not used for vanishing point estimation.



(a) Detected line segments.

(b) Horizontal and vertical lines are removed from vanishing point estimation.

Figure 3.5: Response of algorithm for detection of vanishing point.

All detected segments are shown in Figure 3.5a. They have different colors according to which vanishing point they converge. It is output of the algorithm described above. In this case 3 vanishing points are detected. Vanishing point for blue segment lies in infinity. Vanishing point for green segments lies in the upper left corner and vanishing point for red segments is in the middle. In this case vanishing point for corridor (red) is not computed properly, because of presence of horizontal green segments.

After removal of all horizontal and vertical line segments vanishing point for corridor is found in better place and can be used for navigation (Figure 3.5b). Outliers which have not been used for the vanishing point estimation are not present in this image.

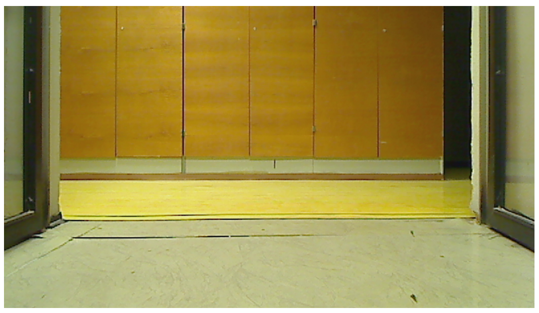
3.2 Matching between two images

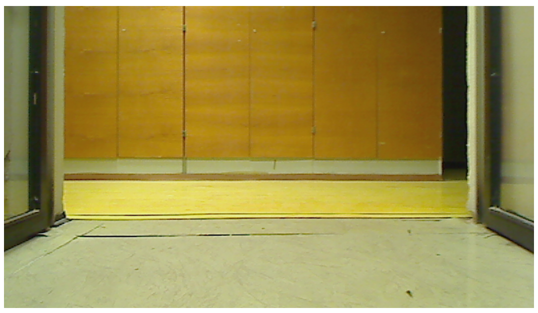
In this section we describe method, which is used, when we need to find position of one image within another. During navigation robot decides at one place to rotate by 90° to the side in order to find the vanishing point in perpendicular corridor. If it does not find it, it rotates back to position, in which it initiated rotation. Since we are not using precise motors, robot will not return to the same position after second rotation. We need to adjust its direction. We did it by image matching.

The robot scans reference image from camera, before it starts to rotate. After second rotation it scans next image and compares it with the reference image. Since camera is not moving on the robot, the robot can adjust its direction according to information extracted from these 2 images.

At first we detect feature points in both images. These points are representing distinctive locations in the image such as corners or blobs. When the images are overlapping, the same feature can be found in both images. Matching between the same features is realised using descriptors. Descriptors are usually vectors in euclidean space, which represent the area around a feature point. In the next phase, descriptors from the references image are matched to the descriptors from the second image. After matching phase we have evidence about pairs of points, which correspond (Figure 3.6c).

Matrix of projective transformation can be computed from this information. Then we can localize the reference image within the second image by transforming corners of the reference image by projective transformation. Example of the reference image and the image taken after rotations is in Figure 3.6a and 3.6b.


(a) Reference image


(b) Image taken after 2 rotations (to the right side and back).

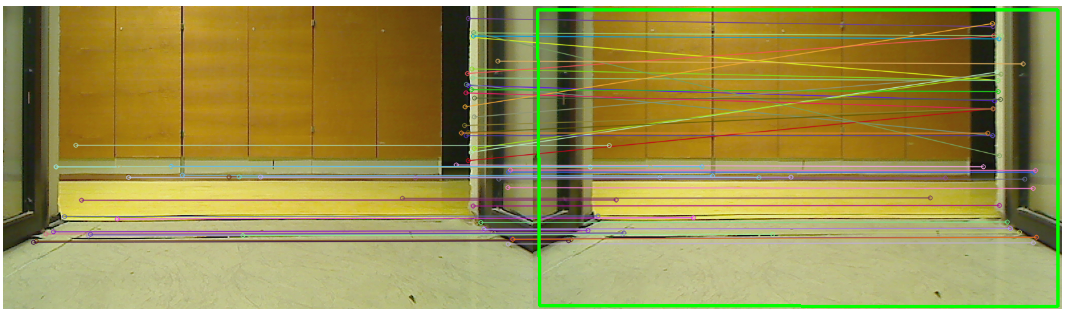

(c) Position of the reference image within the image after rotations.

Figure 3.6: Matched feature points and projective transformation.

3.2.1 Feature point detection

We are using SURF[4] detector for detecting features in the images [Bay et al., 2008]. SURF detector has property of repeatability. The repeatability expresses the reliability of a detector for finding the same physical interest points under different viewing conditions. SURF detector is scale- and rotation-invariant detector, therefore is suitable for detecting same features in the reference image and the image after rotations.

3.2.2 Feature point description and matching

The first step of the descriptors computation consists of fixing a reproducible orientation based on information from a circular region around the feature point. Then square region is aligned to the selected orientation. Finally the SURF descriptor is extracted from it.

Matching between descriptors is performed by the approximate algorithm proposed in [Muja and Lowe, 2009]. It is nearest-neighbors search among descriptors.

[4]Speeded-Up Robust Features

It uses search in randomized kd-trees. Since it is approximate algorithm (not exact), there can be false matches between descriptors of feature points, which do not correspond.

3.2.3 Homography operation

Homography, or projective transformation in 2-dimensional plane is transformation, where straight lines in image are preserved (they remain straight after transformation). It can be described by the equation $\vec{p'} = H\vec{p}$, where H is 3×3 matrix of homography, $\vec{p}$ is homogeneous coordinate of the point from reference image and $\vec{p'}$ is homogeneous coordinate of the point from the image plane transformed by homography. It can be rewritten to the form,

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

where elements of matrix H are arbitrary real numbers. Homogeneous coordinate of the point $(x, y)^T$ corresponds to triple (xZ, yZ, Z), where $Z \neq 0$. Homogeneous coordinates remains unchanged by multiplying by the common factor.

Matrix H can be computed, when at least 4 pairs of (non-collinear) points between referenced and transformed image exists. There can be also outliers, pairs of points, which were not matched correctly. We are using robust RANSAC algorithm [Fischler and Bolles, 1981] in order to find homography matrix in presence of outliers.

Once the homography matrix is found, we can find coordinates of corners of the reference image in the transformed image. Then we can decide, whether robot needs rotate to the left, or to the right in order to adjust its direction.

3.3 Limits of computer vision methods

We decided to use rather robust methods, which are not restricted to particular environment, such as presence of specific color in the image or artificial landmarks. Some of the methods, which we used for image processing has parameters which affect their performance.

The Canny edge detector contains a number of adjustable parameters, which can effect computation time and effectiveness of the algorithm. Size of the Gaussian filter can be adjusted in smoothing phase before application of the Canny edge detector. Smaller filter cause less blurring and allow detection of small, sharp line. A larger filter causes more blurring and allow detection of larger, smoother edges. The Canny edge detector uses also two thresholds for filtering and tracking edges. We used constant settings of these parameters, which worked well in testing corridors.

Image matching method is based on localising and matching feature points between images. When the reference image and the image after rotation didn't contain distinctive features, homography matrix was not computed. Therefore position of robot was not adjusted. It happened when these images were uniform, such as blank wall.

More specialized methods can be used to support general robust methods. Control architecture, which we used enables to combine several methods. Where one method fails other can succeed.

Chapter 4

Our approach to visual navigation

In this chapter we present details of navigation control. We are using Agent-Space architecture described in section 2.2 and methods from computer vision described in chapter 3. This architecture helped us overcome problems regarding combination of multiple behaviors. We implemented several agents and created hierarchy among them.

4.1 Environment

We decided to build navigation control according conditions, which are present at our university campus. There are lot of long straight corridors, which are perpendicular to each other. These corridors are connected through T-junctions and L-junctions as shown in Figure 4.1.

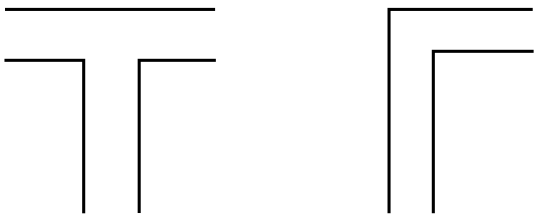


Figure 4.1: T-junction and L-junction.

4.2 Navigation control

The navigation control is split into two layers. The lower-level layer serves for navigation in straight corridors and the upper-level layer serves for navigation in junctions of corridors.

4.3 Navigation in straight corridor

For navigation in straight corridors we used presence of vanishing point in the images acquired from the camera. Vanishing point is detected in these corridors using method described in 3.1. Consequently movement of robot can be adjusted according position of vanishing point in acquired images. Algorithm which is used for this behavior is following. If the vanishing point is detected in the center column of the image, robot is moving forward. If the vanishing point is detected right from that column, robot is heading to the left wall and needs to rotate to the right until vanishing point will be again in that center column. In the same manner robot needs to rotate to the left when the vanishing point is on the left side of the center column (it is heading to the right wall). Figure 4.2 shows all three situations.

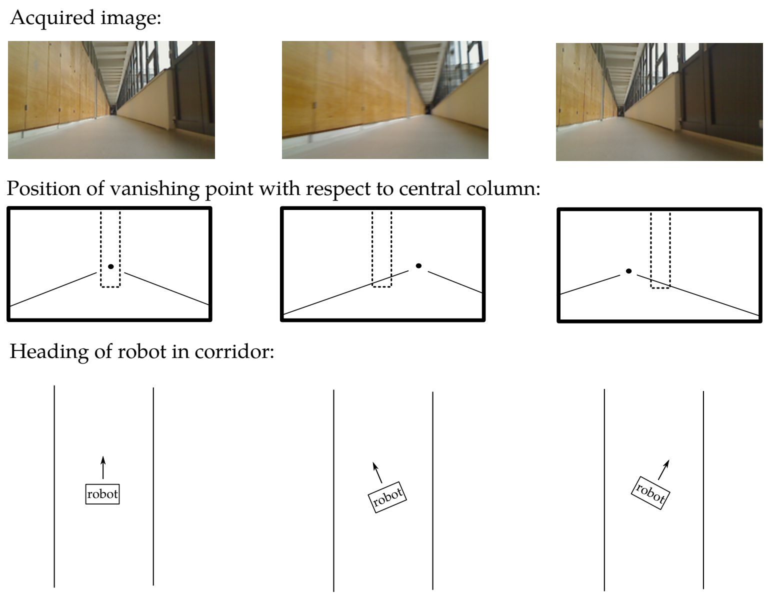

Acquired image:
Position of vanishing point with respect to central column:
Heading of robot in corridor:
robot
robot
robot

Figure 4.2: Navigation of robot in straight corridor.

In corridors, the vanishing point is located in the upper part of the image. It is in the center of the image, when the optical axis of the camera is pointing at it. Therefore we used the following dimensions for the center column. Its width is 1/10 of the image width and its height is 3/4 of the image height.

Realisation of this behavior for adjusting direction of the robot is straightforward. It is realised by three modules with the following characteristics.

- `ImageGrabber` module reads image from camera and stores it.
- `CorridorNavigation` module gets images from `ImageGrabber` module. Then it computes the coordinates of the vanishing point and stores it. According position of vanishing point in the image it generates the command to `MotorControl` module.
- `MotorControl` module gets commands from other modules and sends it directly to chassis.

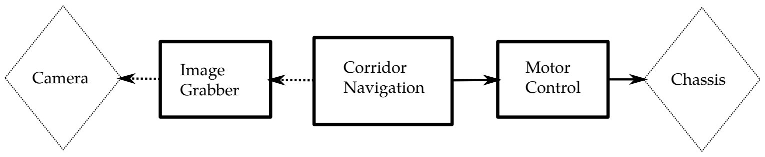


Figure 4.3: Modules of navigation in a straight corridor.

Scheme for this navigation is shown in Figure 4.3. The camera and actuators(motors) are shown as diamonds and modules are shown as rectangles. This is general scheme and can be realised in many ways.

We can easily implement it using Agent-Space architecture as shown in Figure 4.4. Modules used in general schema are substituted by groups of agents. From now on, each group of agents in schemes which corresponds to particular behavior is in the ellipse.

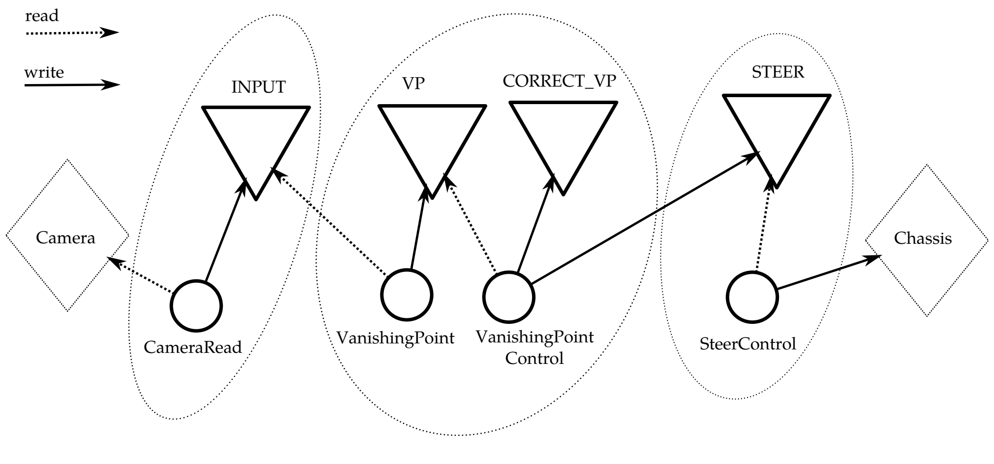


Figure 4.4: Realisation of navigation in straight corridor using Agent-Space architecture.

Agent `CameraRead` reads images from camera and stores them to block **INPUT**. Agent `VanishingPoint` reads in each course image from block **INPUT** and computes coordinates of the vanishing point. It stores them to block **VP**. Coordinates of the vanishing point are read by agent `VanishingPointControl`, which generates command for chassis according position of vanishing point in the image. It stores this command to block `STEER`. `VanishingPointControl` also stores information whether the vanishing point was in the center column into block **CORRECT_VP**. When the robot did not move forward for certain time it means that it lost the vanishing point. This information can be used by agents in the higher-level of the hierarchy for initiation another behavior.

Finally agent `SteerControl` reads commands from block **STEER** and sends them to chassis. It is called periodically by timer. If there are no data stored in that block, it sends default command to stop a movement of the robot. Commands which can be sent directly to chassis are described in section 5.1.

4.4 Navigation in junctions of corridors

We want to create behavior for movement of the robot in junctions. We consider it as the upper-level behavior comparing to a movement in a straight corridor. Generally it consists of the three phases:

1. Movement in the first corridor.
2. Entering the junction of corridors.
3. Movement in the second corridor.

4.4.1 Problem

Reasonable behavior for the second phase is stopping in the center of the second corridor and rotate until facing the end of it. Although it sounds easy it is not. We can't perform the simple operation of stopping in the center of a corridor. We don't have sensors or methods which can tell the robot information about distance from the wall into which robot is heading. Still we can create method which solves this problem. This method will work reasonably in junctions, where vanishing point can be detected in second corridor.

Traversing through junction begins with navigation in the corridor. We can use method described in the previous section. Problem arise when the robot lost the vanishing point, which was used as main source of knowledge about the corridor.

It happens before the end of the corridor, because the vision range of the robot is limited. Situation is shown in Figure 4.5.

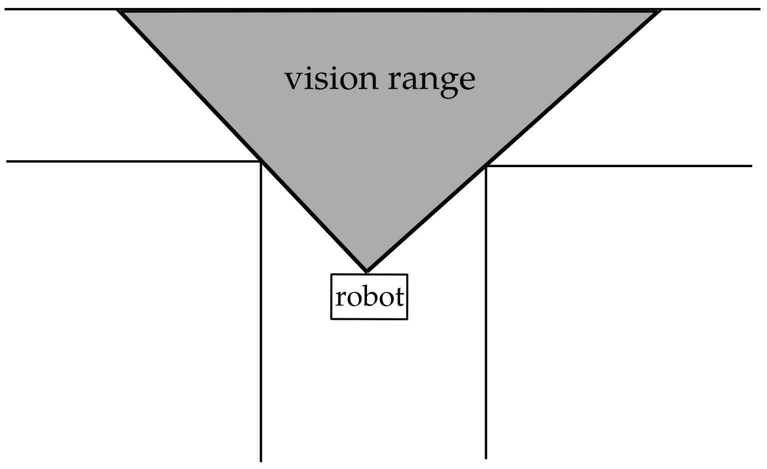


Figure 4.5: Corridor in which is robot situated is not longer visible and vanishing point can't be detected.

The robot stops after losing of vanishing point. This is the place, where we must initiate other method for finding vanishing point in next corridor.

4.4.2 Initiation of behavior

We have information about the last vanishing point, which was used for moving forward. It is stored in the block **CORRECT_VP**. In Agent-Space we can create an agent, who can observe this block and when there was no update for certain time it initiates behavior of groups of agents. Realization of this behavior in Agent-Space is shown in Figure 4.6.

Agent `VanishingPointCounter` reads from block **CORRECT_VP** information, whether detected vanishing point was used for moving forward (it lies in the center column of the image). If it was, `VanishingPointCounter` initialize counter to certain value T and writes it to block **COUNTER**. In each course it reads value from this block and decreases it by one, if the value read from **CORRECT_VP** was $false$.

If `VanishingPointCounter` reads $false$ value in T consecutive courses, value in block **COUNTER** will be 0. Afterwards it writes value $ACTIVE$ into block **DIRECTION_FINDER_ACTIVITY**. It means that in T courses robot doesn't move forward, because vanishing point was not in the center column. It initiates

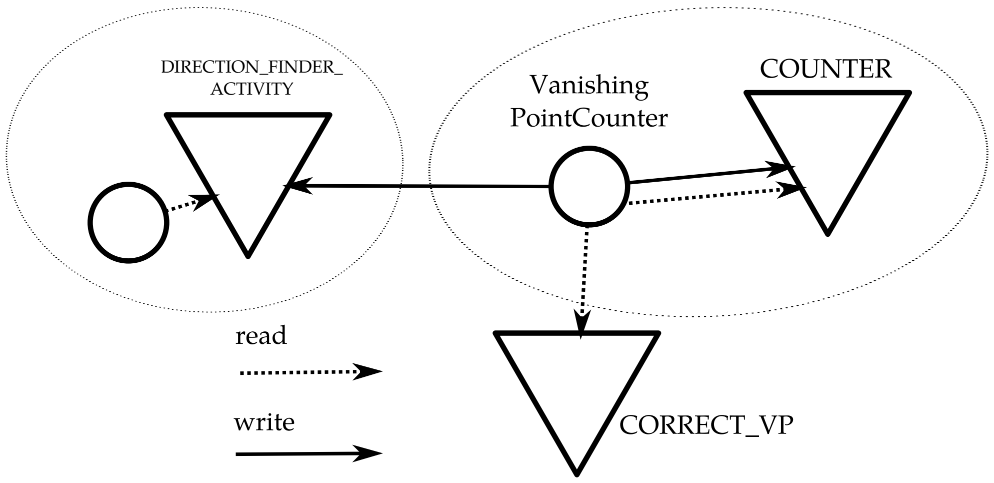


Figure 4.6: Initiation of activity by `VanishingPointCounter`.

behavior for entering the junction in order to navigate later in perpendicular corridor.

4.4.3 Method for finding vanishing point in junction

As we already said, we assume that the robot stops before the junction of corridors, because it lost the vanishing point. After losing information about it, robot needs to find next one in another corridor. Afterwards it can apply the behavior created before and move in the corridor without collision.

This behavior for navigation in straight corridor is never stopped, but commands to chassis sent by agents on lower-level of hierarchy are suppressed by the commands sent by agents on upper-level hierarchy. It is performed by writing data to block **STEER** with higher priority.

We want robot to move forward and periodically check, whether the vanishing point can be detected when it rotates to the left or right side.

First robot rotates by 90° to the right, then it checks, whether vanishing point is present in the center column. If it is, robot starts to follow it. If it is not, it will return back to starting position by rotating 90° to the left. Similarly it checks the left side by rotating 90° to the left. If there is not vanishing point present in the center column, it will return to starting position again. Otherwise it starts to follow it. After that robot moves forward for short period of time and checks both sides again. It repeats this behavior until it is inside the junction. There it checks left and right side again and select one, where vanishing point is present.

We can implement this behavior using Agent-Space architecture and combine it with with the navigation in straight corridor. Agents `DirectionFinder` and `VanishingPointChecker` are used for realisation of the behavior. This upper-layer behavior combined with lower-layer behavior for navigation in straight corridor is shown in Figure 4.7.

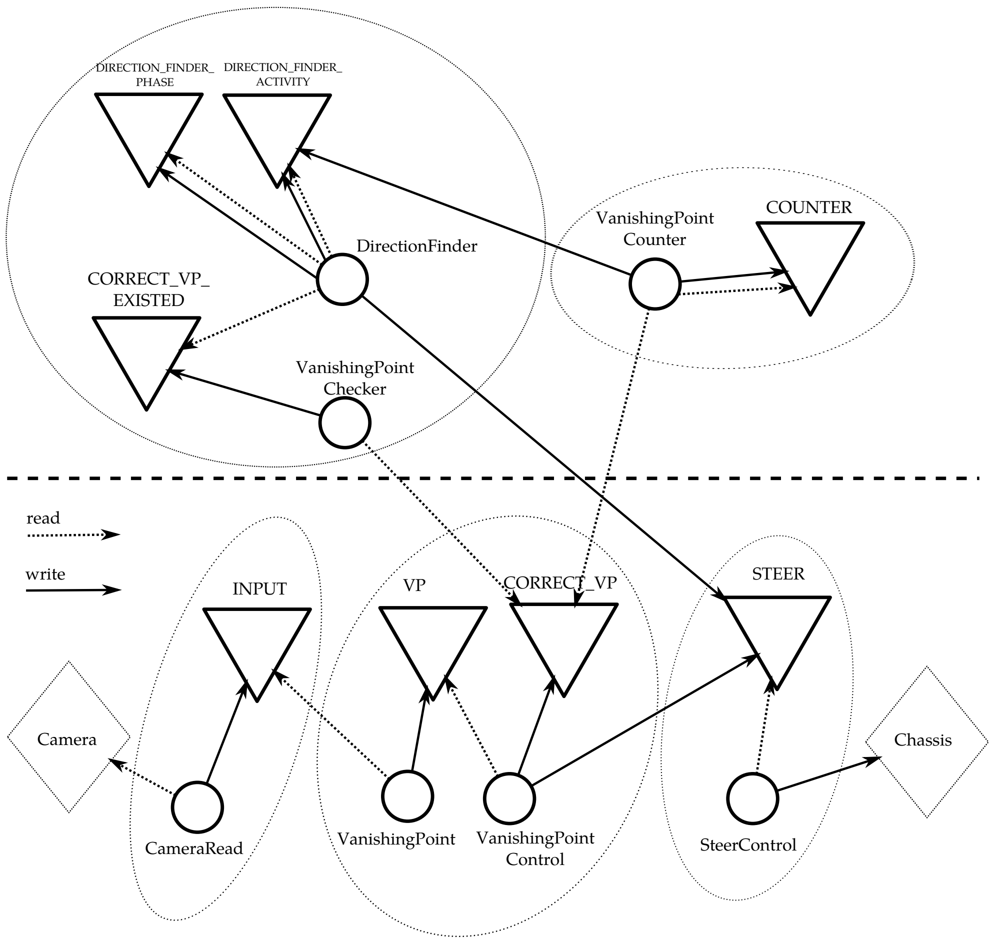


Figure 4.7: Agent-Space scheme for upper-level and lower-level behavior.

Agent `DirectionFinder`: In the each course it reads value from **DIRECTION_FINDER_ACTIVITY**. If value *ACTIVE* is stored there, it will perform its activity, otherwise it is passive and its course ends. It works in phases. In **DIRECTION_FINDER_PHASE** it stores its current phase. In the beginning of each course it reads its current phase. On the end of the course it updates its phase. One reason for storing the phases externally is requirement of the Agent-Space on agents to be pure reactive.

In phase 0 it initiates its behavior and sends command to chassis for turning right with validity 9 seconds. It is sufficient time to turn by 90° to the right side. During phase 1 robot rotates to the right. On the end of phase 1 it checks whether the block **CORRECT_VP_EXISTED** contains value *true*. It is information whether vanishing point was detected within last 3 seconds and can be used for navigation (presence in the center column). This value is updated by `VanishingPointChecker`.

If the value is *true*, `DirectionFinder` deactivate itself by writing the value *PASSIVE* into **DIRECTION_FINDER_ACTIVITY** and update its phase to phase 0. It means that vanishing point was found in perpendicular corridor and robot starts to follow it. If the vanishing point was not detected during rotating (*false* is read from **CORRECT_VP_EXISTED**), it changes its phase to phase 2 and sends command to chassis for rotating to the left.

In the phase 3 `DirectionFinder` turns left by 90°. At the end of the phase 3 it checks whether vanishing point is present. If it is, it switches to phase 0, otherwise in phase 4 it turns back. Afterwards in phase 5 it sends command for moving forward and then switches back to phase 0 for repetition of this behavior.

`DirectionFinder` deactivate itself only after phase 1, or after phase 3, when vanishing point was detected again.

Agent `VanishingPointCecker`: This Agent has trigger registered in block **VP_CORRECT**. After change in this block it is woken up from sleep and writes value *true* in block **CORRECT_VP_EXISTED** with validity of 3 seconds. This value tells `DirectionFinder`, that within 3 seconds was detected vanishing in the center column and can be used for navigation.

4.4.4 Correction of rotation

Rotating by 90° and back will not return robot to the same position as it was before rotation, since the robot is not using precise motors. We found time needed to rotate robot 90° to side by trial and error method. If we want robot to proceed forward in direction it was moving before, we need to adjust robot heading. Before the robot initiates rotation to the side it can save the reference image. Later it can compare it with the image acquired after rotation.

We can find position of the reference image within the image obtained after rotation. It is performed by detecting feature points and computing homography operation between these two images. This method is described in section 3.2.

Agent `DirectionFinder`, described in previous section needs to adjust its heading after phase 2 and phase 4, when it returns to starting position after rotations. There it wakes up groups of agents which do correction of position.

This behavior can be combined with the one we created before. We have implemented group of agents which realise the behavior for correction of heading of the robot. Realisation of it in Agent-Space is shown in Figure 4.8

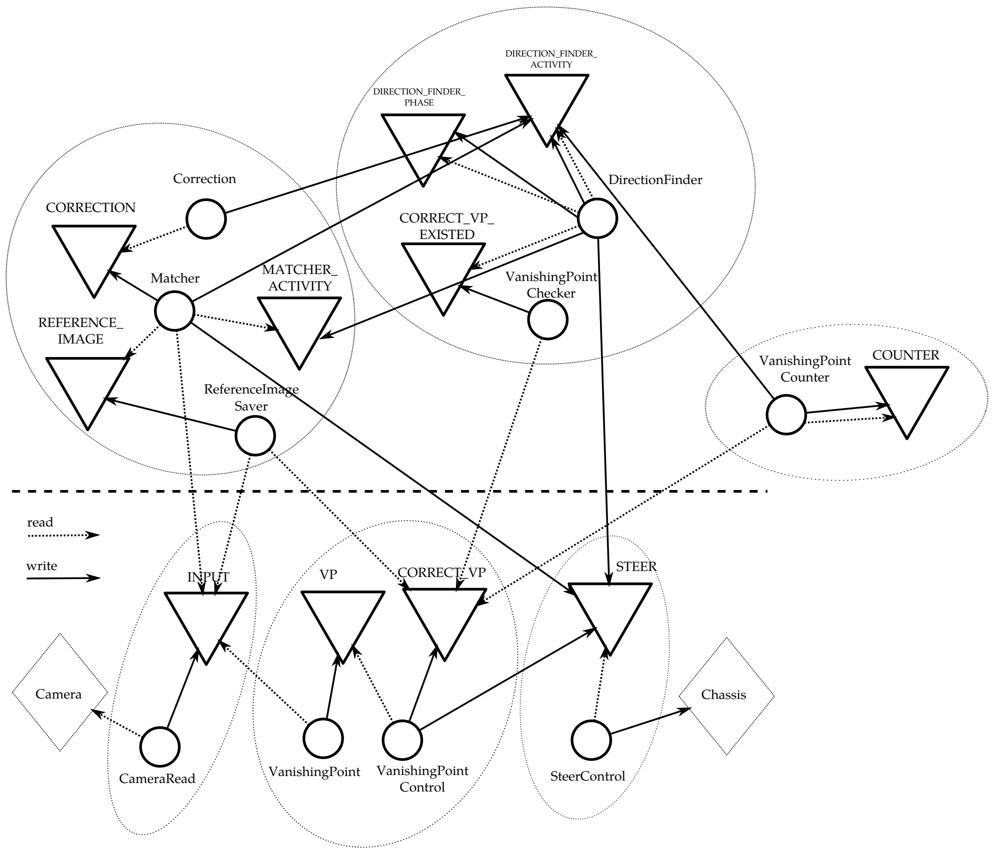


Figure 4.8: Rotation correction behavior with other behaviors in Agent-Space.

Agent `ReferenceImageSaver` : It reads value from block **CORRECT_VP** and from block **SAVE_IMAGE** (it is not shown in the scheme). Value from **SAVE_IMAGE** tells it, whether it can save any image.
When the value from **CORRECT_VP** and **SAVE_IMAGE** is *true*, it saves the current image from **INPUT** in to block **REFERENCE_IMAGE**.

Agent `Matcher` : It has trigger, which wakes him up. Its activity is controlled by value in block **MATCHER_ACTIVITY**. If the value is *PASSIVE* agent is doing nothing in the course. Otherwise it perform following activity. It reads reference image from **REFERENCE_IMAGE** and current image from **INPUT**. Then it finds the feature points in both of them and find homography matrix H among them.

If matrix H exists, `Matcher` takes image of corners of reference image and find their position in the current image. If these points lies left from the corners of current image, `Matcher` sends command to chassis for rotating right, which is valid for short period of time. If these points lies on the right side it sends command to chassis for rotating left, which is valid for short period of time. If some command for correction was sent, it stores value *true* in the block **CORRECTION** with validity of that command.

If matrix H does not exist, or images of corners are projected to degenerated shape, it sends no command to chassis for correction. If no adjustments of rotation was made, it wakes up `DirectionFinder` immediately.

Agent `Correction` : This agent is activated by `Matcher`, when it sends command to chassis. This agent is checking validity of that command and if it is not valid anymore, it wakes agent `DirectionFinder` up.

It is performed by reading value from the block **CORRECTION**. If it reads *true* from there, it starts its activity. When the value $false$ is read from there after reading *true* value in previous course, it wakes `DirectionFinder` up by writing *true* into **DIRECITON_FINDER_ACTIVITY**.

4.5 Properties of the navigation control

We built navigation control incrementally and created complex behavior. Addition of new behavior was not big problem, since not many modifications needed to be made in previous agents.

Agent-Space architecture provided sufficient tool for creating different agents behaviors as counter agent, or agent which operates in phases. We did not feel big restriction, while we were building navigation control using it. On the other hand its features were useful, especially for propagation of images between agents or combination fast modules such as vanishing point estimation and slow modules such as image matching.

The biggest advantage of using this architecture we have found in possibility to upgrade existed navigation control by adding new behavior realised by another group of agents. For instance, we can add agents which are specialized for detection of landmarks such as door or agents which can find obstacle in vision range.

Chapter 5

Implementation and results

In chapter 4, we have described structure and methods of navigation control we have developed. In this chapter we present implementation details and results which we have got using proposed approach.

5.1 Mobile robot specification

The mobile robot which we used consists of a laptop, an ordinary USB camera and a battery powered chassis. Our mobile robot is shown in Figure 5.1. The laptop which run navigation control had Intel Core i5 3210M Ivy Bridge processor and 8 GB memory.

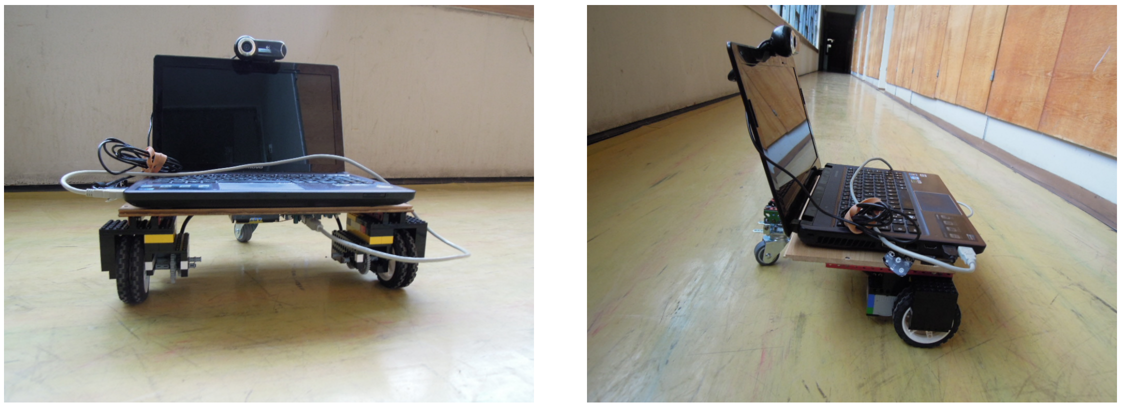


Figure 5.1: Different views on the mobile robot.

We used Logitech Webcam Pro 9000, which was connected to laptop through USB port. The resolution of the image acquired from the camera was 800×456 pixel. Optical axis of the camera was tilted to be parallel to the longitudinal corridor axis (it preserved horizontal lines during rotation).

The chassis consisted of 2 motor-driven wheels on front and 1 support rear wheel. Motors were taken from LEGO RCX set. The motors were operated with 4-channel relay. This relay is part of the chassis and initiates rotation of particular wheels into any direction at constant speed depending on the command sent to it. Table 5.1 shows direction of rotation of the wheels initiated by relay, when particular command was sent to chassis.

command	**left wheel**	**right wheel**
LEFT	backward	forward
RIGHT	forward	backward
FORWARD	forward	forward
STOP	none	none

Table 5.1: Command sent to chassis and direction of rotation of the wheels

Chassis was connected to laptop through USB port. It was operated through native linux VCP[1] driver.

5.2 Implementation details of Navigation control

We implemented navigation control in C++ language under Linux platform. It is console application which is using library with Agent-Space architecture implementation and image processing library.

In our navigation control we are using several methods from computer vision. We used open-source library OpenCV[2] which contains implementation of many methods and algorithms from domain of computer cision. We did most of the development using OpenCV 2.4.3, later we used version 2.4.4. On the end of development version 2.4.5 was released. We used its C++ interface.

Figure 5.2: OpenCV logo

For Agent-Space architecture we used open source implementation available on web [3]. We fixed bugs there and

[1]Virtual COM Port

[2]`http://opencv.willowgarage.com/`

[3]`http://www.agentspace.org/download.html`

modified it to run under Linux environment. It is library, which contains framework for further implementation of agents. Agents and space are implemented there as threads, which run parallel. The core of the library are the `Pthreads` (POSIX threads), which serve as API for creating and manipulating threads. `Pthreads` are wrapped in C++ objects for simplification of their usage.

Library with Agent-Space implementation provided just the interface for implementation. We designed and created all agents by ourselves. In addition to the agents described in chapter 4 we implemented other support agents. We created agents for reading and writing images from file. These can be used for testing with previously acquired images. We created also agent which displayed images of selected blocks on screen which was helpful during testing. These agents were easily employed within existed system. Addition of other agents in implementation is easy because the agents are independent. They can affect other agents only throuh the space.

For illustration when we need to run the navigation system with the images from file-system we need just replace agent `CameraRead` by the agent `PNGread`. This agent reads images from file and writes them to block **INPUT**. Other agents can not observe difference, that image in this block is not acquired from the camera.

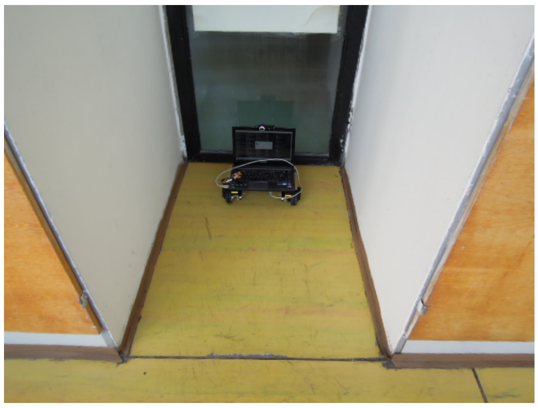
5.3 Experimental results

We did all of the experiments with the mobile robot in the corridors at university campus. Tests have been performed in the corridors illuminated by the natural light or by artificial light from the neon lamps. The corridors in which we have tested the navigation control are shown in Figure 5.3.

We used OpenCV driver for grabbing images from the camera. For now, this driver has not enabled change of camera parameters such as gain or exposure during run. Therefore we set all camera parameters before run. This resulted in worse performance of line segment detection, when robot moved in corridor to places, where light has lower intensity. It happened during testing in corridors illuminated by artificial light source. Some parts of the corridor were lighted with lower intensity than others.

Corridor A; width=1.6 m

Corridor B; width=1.6 m


Corridor C; width=1 m

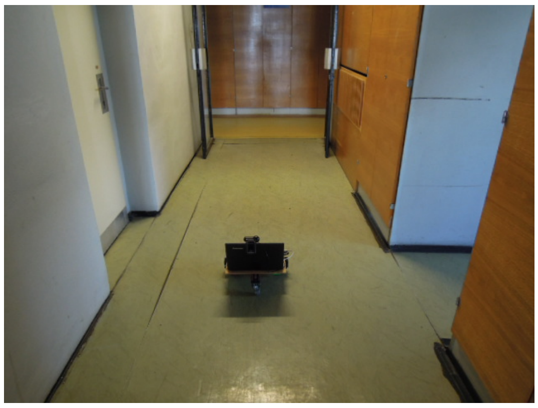

Corridor D; width=1.8 m

Figure 5.3: Corridors used for testing the navigation control with their width.

5.3.1 Straight corridors

In the straight corridor the robot used navigation method based on position of the detected vanishing point in the image. We tested mobile robot in the 2 different scenarios:

- Scenario 1: Corridor A illuminated by natural light
- Scenario 2: Corridor B illuminated by natural light

The robot followed the vanishing point in both scenarios and kept almost constant distance from the side walls. It was moving forward and adjusted its direction, when the vanishing point was not detected in the center column.

Detected vanishing points in the consequent images from the camera does not have necessarily the same coordinates. The first reason is that robot is moving and the consequent images are taken from different place. The second reason is that

PPHT (section 3.1.1) can detect different line segments in the consequent images and it affects the estimation of the vanishing point.

Detected vanishing point does not need to correspond with the only true vanishing point of the corridor. Some line segments which were used for the vanishing point estimation do not need to point in the vanishing point. This is because the vanishing point estimator does not consider them as an outliers and includes them in the computation. It results in biased position of the vanishing point. Despite of this fact the overall performance of navigation in straight corridor was solid.

We can probably say that the best method for non-colliding navigation in straight corridor without additional sensors is following the center line of the corridor, because distance from both side walls is biggest. Method which we proposed resulted in behavior where robot was following line and didn't change its direction to move in the center of the corridor. We tried to initiate navigation control from 3 different places within corridor(left, center, right) 5.4. The robot was following the line towards detected vanishing point in each case.

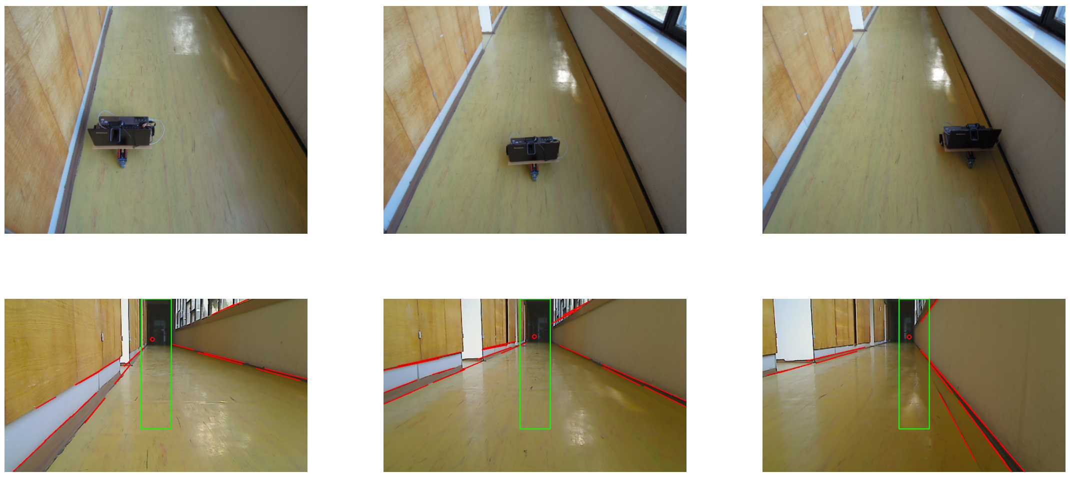


Figure 5.4: Different positions of the robot within the corridor.

5.3.2 Junctions of corridors

We have tested performance of the navigation in the T-junction of corridors in 2 different scenarios:

- Scenario 1: T-junction of corridor C and corridor B illuminated by natural light.

- Scenario 2: T-junction of corridor D and corridor B illuminated by artificial light source.

In the both scenarios, the robot first navigated using proposed method for navigation in the straight corridor until it lost the vanishing point. Afterwards, the behavior for finding a vanishing point in a perpendicular corridor was initiated.

Scenario 1

Different stages during navigation through junction of corridors are shown in Figure 5.5.

The robot passed through several stages during navigation through T-junction of corridors C and B:

1. The robot is initiated in the corridor C and detects the vanishing point in the center column.
2. The robot has lost the vanishing point and behavior for finding the vanishing point has been initiated.
3. The robot is rotated 90° to the right.
4. The robot is rotated back to the original direction.
5. The robot is rotated 90° to the left.
6. The robot is rotated back and adjusts its direction by matching between the reference image and the current image.
7. The robot advances further and haven't found the vanishing point on the right side.
8. The robot is rotated back to the original direction.
9. The robot is rotated 90° to the left. It has not found vanishing point there. It rotates back and advances further to the junction.
10. The robot is in the junction and initiates the rotation by 90° to the right.
11. The robot has found the vanishing point in the corridor B.
12. The robot follows the vanishing point in the corridor B.

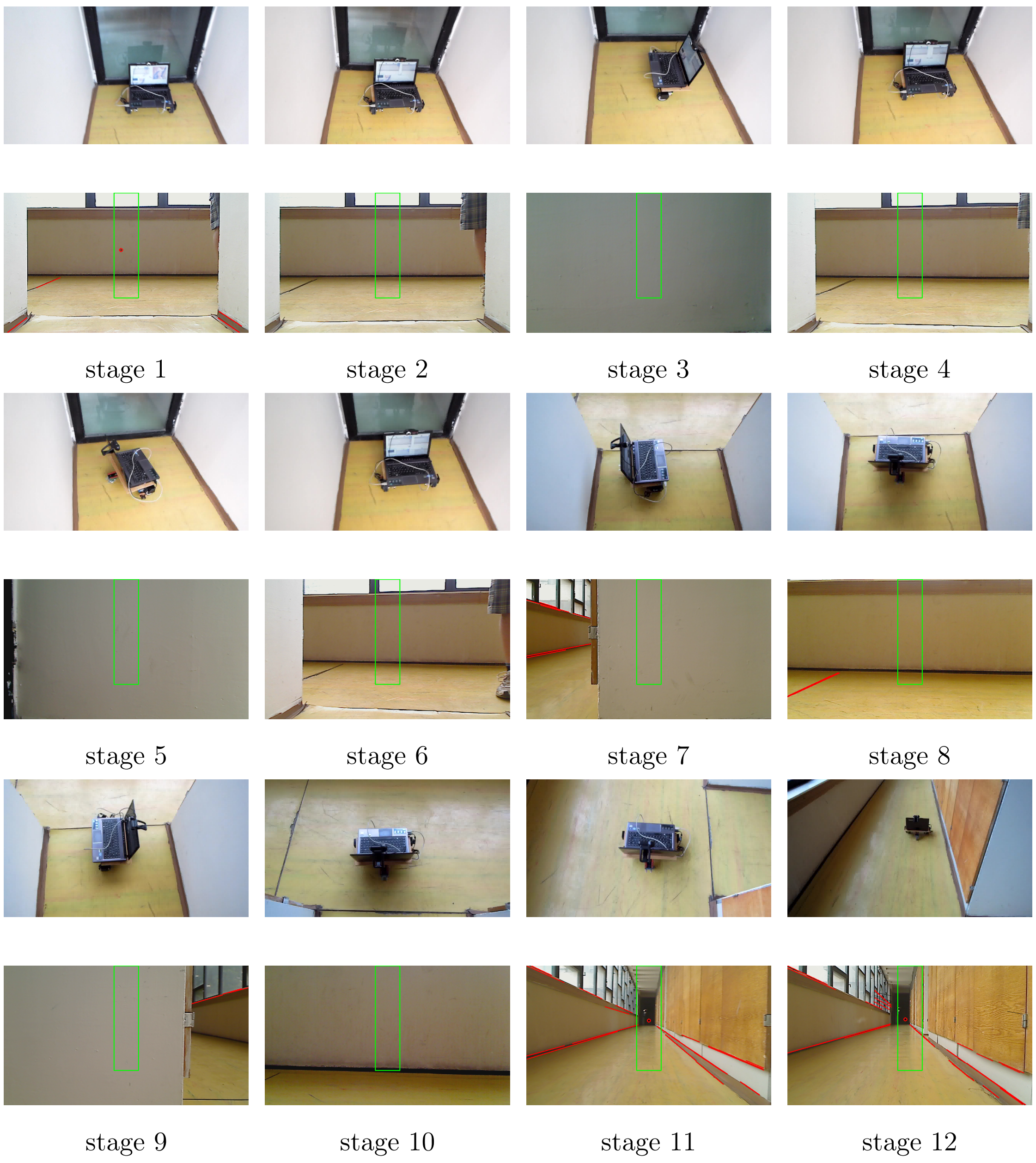


Figure 5.5: Different stages during navigation of the robot in the Scenario 1.

The robot adjusts its position at the stage 6 by matching the reference image with the current image (Figure 5.6). In this case it adjusts the direction by short rotation to the right side, since before initiation rotation in order to find vanishing point in the stage 2 it was heading to the center of the corridor. In stage 6 it is heading slightly to the left wall.

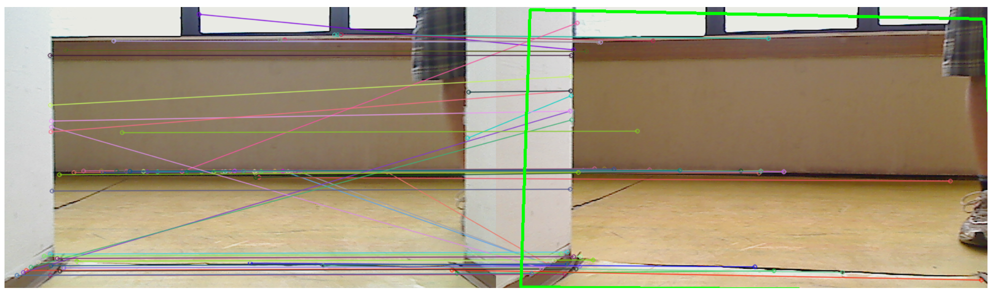


Figure 5.6: Matching between 2 images.

Scenario 2

In this scenario the robot performed navigation in the T-junction of the corridors D and B illuminated by artificial light. The robot passed through several stages (Figure 5.7).

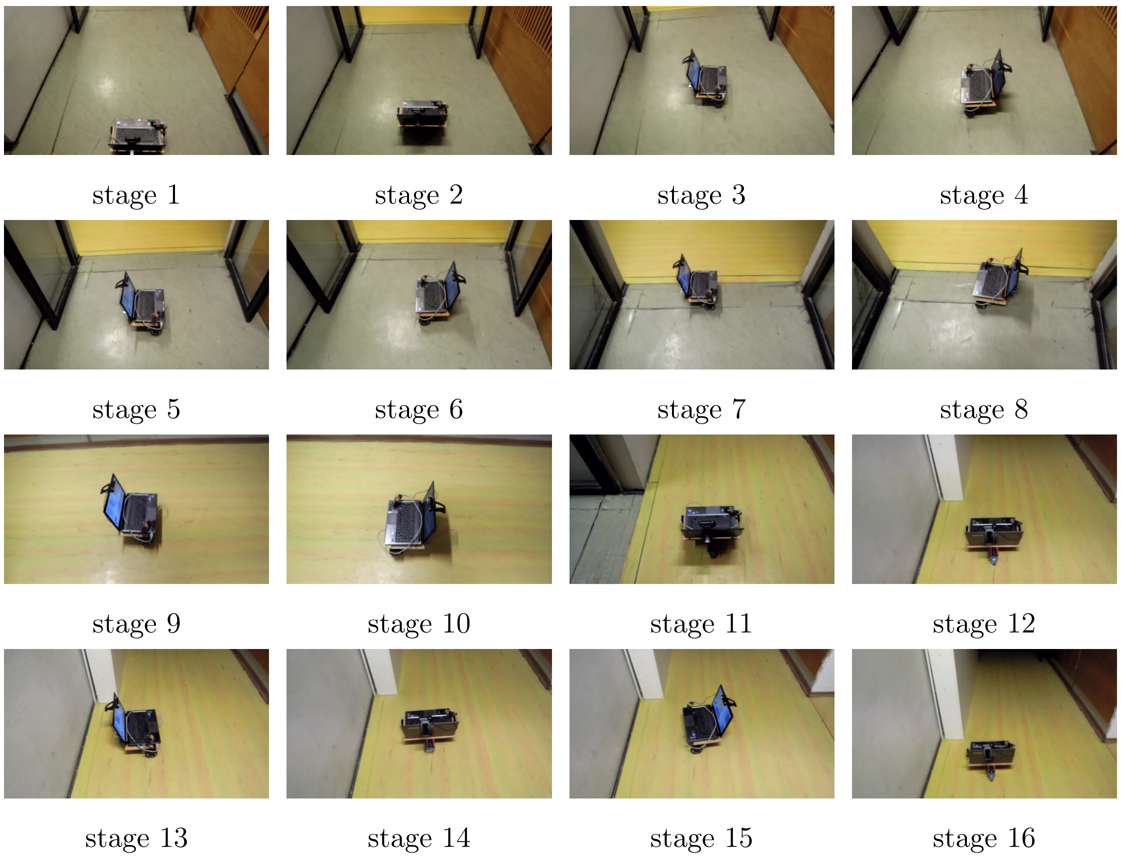


Figure 5.7: Different stages during navigation of the robot in the scenario 2.

- The robot navigates in the corridor D.
- The robot has lost vanishing point in the stage 2 in the corridor D.

- In the stages 3, 5, 7 and 9 it looks for the vanishing point on the right side.
- In the stages 4, 6, 8 and 10 it looks for the vanishing point on the left side.
- The robot has found the vanishing point in the stage 10 in the corridor B. It navigates using behavior for navigation in straight corridor.
- The robot has lost vanishing point in the stage 12 and initiates behavior for finding vanishing point.
- The robot looks for the vanishing point on the right side in the stage 13.
- The robot looks for the vanishing point on the left side in the stage 15.
- Int the stage 16, the robot continues to move forward. In the next stage it will look for the vanishing point on the right side.

The robot was already in the junction of corridors in the stage 9. It did not find the vanishing point on the right side, because the corridor B was not illuminated enough to detect line segments for the estimation of the vanishing point. On the other hand the robot found it on the left side and continued to navigate there.

5.4 Limitation of navigation control

Experiments which we performed were successful. The robot navigated through corridors without collision. Several simple behaviors realised by agents cooperated to reach complex goal. On the other hand, there are still limits in navigation control which we proposed.

The robot navigated in straight corridors without problems when it could detect line segments in acquired image. We have used constant settings of camera parameters. When the robot entered dark areas, it failed to find line segments, and to detect vanishing point. Next limit is that detection of vanishing point was sometimes biased by presence of line segments detected on sides (such as windows). Proposed method for vanishing point estimation gives the same priority to all line segments. Method which gives more priority to line segments detected between wall and floor could help.

In the navigation through junction of corridors the robot looks for vanishing point in perpendicular corridor by rotation to the right and to the left side. After rotation it adjusts its direction by image matching method proposed in section 3.2. It could fail to find the homography matrix, when there is not enough pairs of matched feature points between the images.

In this case another group of agents realising the different method for feature point detection could success and extends the previous method.

Chapter 6

Conclusion

In this work we proposed original approach to visual navigation control of mobile robot. We tried to simulate navigation control in the living organisms. In order to do that we used just visual information from camera for navigation control. Additional sensors were not used.

We used specific Agent-Space [Lúčny, 2004] architecture which has biological relevancy and is suitable for building complex system. This architecture provided us sufficient tool for building navigation control.

The mobile robot which we used for performing navigation had ordinary camera and simple actuators. Problems with combination of several behavior to complex system were solved by using the Agent-Space architecture. We used several methods from computer vision. These methods had different computational demands. Agent-Space enabled us to combine faster less accurate methods with slower precise methods, which supported the overall behavior.

Future work will be toward improvement of robustness of the proposed navigation control with respect to various environment conditions (light intensity, shadows, presence of particular objects in scene). Since the architecture which we used provides technique for combining several behaviors, we suggest add more specialized methods to proposed one. Where one general method fails, specialized method can work well. It will result in overall robustness of the system.

Our work resulted in navigation control which was tested in several corridors at our university campus. Mobile robot was capable to navigate in straight corridors and in junction of corridors.

Our contribution to the open source community is that we fixed bugs in previous implementation of Agent-Space architecture and published our code for further development[1].

[1]It can be found at `http://www.agentspace.org/NotebookNavigation`

Bibliography

[Arkin, 1989] Arkin, R. C. (1989). Motor schema-based mobile robot navigation. *I. J. Robotic Res.*, 8(4):92–112.

[Arkin, 1998] Arkin, R. C. (1998). *Behavior-Based Robotics*. MIT Press.

[Bay et al., 2008] Bay, H., Ess, A., Tuytelaars, T., and Gool, L. V. (2008). Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3):346 – 359. <ce:title>Similarity Matching in Computer Vision and Multimedia</ce:title>.

[Bonin-Font et al., 2008] Bonin-Font, F., Ortiz, A., and Oliver, G. (2008). Visual navigation for mobile robot: a survey.

[Brooks, 1986] Brooks, R. (1986). A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of*, 2(1):14–23.

[Busquets, 2003] Busquets, D. (2003). *A multiagent approach to qualitative navigation in robotics*. PhD thesis, Universitat politècnica de Catalunya.

[Busquets et al., 2003] Busquets, D., Sierra, C., and De Màntaras, R. L. (2003). A multiagent approach to qualitative landmark-based navigation. *Autonomous Robots*, 15(2):129–154.

[Canny, 1986] Canny, J. (1986). A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-8(6):679–698.

[Davies, 2012] Davies, E. R. (2012). *Computer and Machine Vision*. Academic Press. Fourth Edition: Theory, Algorithms, Practicalities.

[Fischler and Bolles, 1981] Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395.

[Innocenti et al., 2008] Innocenti, B., López, B., and Salvi, J. (2008). Integrating individual and social intelligence into module-based agents without central coordinator. In *Proceedings of the 2008 conference on STAIRS 2008: Proceedings of the Fourth Starting AI Researchers' Symposium*, pages 82–93, Amsterdam, The Netherlands, The Netherlands. IOS Press.

[Lorigo et al., 1997] Lorigo, L., Brooks, R., and Grimson, W. (1997). Visually-guided obstacle avoidance in unstructured environments. *In IEEE Conference on Intelligent Robots and Systems*, pages 373–379.

[Lúčny, 2004] Lúčny, A. (2004). Building complex systems with agent-space architecture. *Computing and Informatics*, 23:1001–1036.

[Maes, 1989] Maes, P. (1989). The dynamics of action selection. In *Proceedings of the 11th international joint conference on Artificial intelligence - Volume 2*, IJCAI'89, pages 991–997, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Matas et al., 2000] Matas, J., Galambos, C., and Kittler, J. (2000). Robust detection of lines using the progressive probabilistic hough transform. *Computer Vision and Image Understanding*, 78(1):119 – 137.

[Minsky, 1986] Minsky, M. (1986). *The Society of Mind.* Simon & Schuster, New York.

[Muja and Lowe, 2009] Muja, M. and Lowe, D. G. (2009). Fast approximate nearest neighbors with automatic algorithm configuration. In *In VISAPP International Conference on Computer Vision Theory and Applications*, pages 331–340.

[Nieto and Salgado, 2010] Nieto, M. and Salgado, L. (2010). Real-time robust estimation of vanishing points through nonlinear optimization. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 7724 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series.*

[Olajubu et al., 2011] Olajubu, E. A., Ajayi, O. A., and Aderounmu, G. A. (2011). A fuzzy logic based multi-agents controller. *Expert Syst. Appl.*, 38(5):4860–4865.

[Ono et al., 2004] Ono, Y., Uchiyama, H., and Potter, W. (2004). A mobile robot for corridor navigation: a multi-agent approach. In *Proceedings of the 42nd annual Southeast regional conference*, ACM-SE 42, pages 379–384, New York, NY, USA. ACM.

[Torr and Zisserman, 2000] Torr, P. H. S. and Zisserman, A. (2000). MLESAC: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding*, 78:138–156.

List of electronic attachments

Following data are in the enclosed DVD:

- Electronic version of thesis
- Source code of the navigation control
- Videos[2] with robot navigation in different scenarios

[2]More videos can be found at `https://www.youtube.com/results?search_query=NotebookNavigation`