Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering

Bachelor's Project

Autonomous Collision Avoidance on Crossroads

Ondřej Jelínek

Supervisor: Ing. Jiří Vokřínek, Ph.D.

Study Programme: Otevřená informatika, Bakalářský

Field of Study: Informatika a počítačové vědy

May 25, 2012

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Ondřej J e l í n e k

Study programme: Open Informatics

Specialisation: Computer and Information Science

Title of Bachelor Project: Autonomous Collision Avoidance on Crossroads

Guidelines:

1. Study the domain of autonomous collision avoidance.
2. Create agent-based model of vehicles and crossroads.
3. Design and implement algorithms for autonomous collision avoidance.
4. Experimentally evaluate implemented algorithms.

Bibliography/Sources:

[1] Matteo Vasirani: Vehicle-centric coordination for urban road traffic management. A market-based multiagent approach. Ph.D. dissertation, Dept. Arq. Tec. Comp., Juan Carlos Univ., Madrid, 2009.
[2] David Sislak, Premysl Volf, Michal Pechoucek: Agent-Based Cooperative Decentralized Airplane Collision Avoidance. IEEE Transactions on Intelligent Transportation Systems. 2011, vol. 12, p. 36-46. ISSN 1524-9050.
[3] Premysl Volf, David Sislak, Michal Pechoucek, Magdalena Prokopova: Convergence of Peer-to-Peer Collision Avoidance among Unmanned Aerial Vehicles. In 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2007). 2007. ISBN 9780769530277.
[4] Pavel Janovský: Cooperative Colision Avoidance of Road Vehicles. Bakalářská práce FEL, 2011.

Bachelor Project Supervisor: Ing. Jiří Vokřínek, Ph.D.

Valid until: the end of the winter semester of academic year 2012/2013

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE Fakulta elektrotechnická

prof. Ing. Vladimír Mařík, DrSc.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 9, 2012

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Ondřej J e l í n e k

Studijní program: Otevřená informatika (bakalářský)

Obor: Informatika a počítačové vědy

Název tématu: Autonomní vyhýbání vozidel na křižovatce

Pokyny pro vypracování:

1. Seznamte se s problematikou autonomního vyhýbání.
2. Vytvořte agentový model jízdy vozidel křižovatkou.
3. Navrhněte a implementujte algoritmy automního vyhýbání.
4. Implementované algoritmy experimentálně ověřte a porovnejte.

Seznam odborné literatury:

[1] Matteo Vasirani: Vehicle-centric coordination for urban road traffic management. A market-based multiagent approach. Ph.D. dissertation, Dept. Arq. Tec. Comp., Juan Carlos Univ., Madrid, 2009.
[2] David Sislak, Premysl Volf, Michal Pechoucek: Agent-Based Cooperative Decentralized Airplane Collision Avoidance. IEEE Transactions on Intelligent Transportation Systems. 2011, vol. 12, p. 36-46. ISSN 1524-9050.
[3] Premysl Volf, David Sislak, Michal Pechoucek, Magdalena Prokopova: Convergence of Peer-to-Peer Collision Avoidance among Unmanned Aerial Vehicles. In 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2007). 2007. ISBN 9780769530277.
[4] Pavel Janovský: Cooperative Colision Avoidance of Road Vehicles. Bakalářská práce FEL, 2011.

Vedoucí bakalářské práce: Ing. Jiří Vokřínek, Ph.D.

Platnost zadání: do konce zimního semestru 2012/2013

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE Fakulta elektrotechnická

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 9. 1. 2012

Aknowledgements

In this place, I would like to thank my supervisor *Ing. Jiří Vokřínek, Ph.D.* for his patient guidance and valuable advices.

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 24. 5. 2012 Ondřej Jelínek

Abstract

Multi-agent system approach was successfully used to simulate and control air-traffic of autonomous pilotless airplanes during the project AgentFly. Iterative peer-to-peer collision algorithm (IPPCA) was developed specially for conflict resolution of airplanes. The domain of autonomous vehicles crossing an uncontrolled intersection is quite similar to autonomous airplanes and can be modelled and controlled using multi-agent system approach too. In this thesis, I will modify and implemented the IPPCA algorithm to work with autonomous vehicles at the intersection.

Further in this thesis, another collision avoidance (CA) method is described and studied too. Proposed in the article "A Multiagent Approach to Autonomous Intersection Management" by Dresner and Stone, reservation-based algorithm can be used at the intersection, to produce conflict-free plans. Implementing this algorithm is another objective of this bachelor thesis.

Finally, I will experimentally evaluate and compare these two CA algorithms in various scenarios.

Abstrakt

Přístup multiagentních systémů byl úspěšně použit na simulaci a řízení leteckého provozu autonomních bezpilotních letadel během projektu AgentFly. Iterativní peer-to-peer algoritmus autonomního vyhýbání (IPPCA) byl vyvinut speciálně pro řešení konfliktů mezi letadly. Oblast autonomních vozidel projíždějící křižovatku bez světelné signalizace je autonomním letadlům celkem podobná a také se dá modelovat a řídit principy multiagentních systémů. V této práci upravím a naimplementuji algoritmus IPPCA tak, aby fungoval s autonomními auty na křižovatce.

Dále v této práci je popsán a prozkoumán ještě jeden algoritmus na předcházení kolizí. Navrhnut v článku "A Multiagent Approach to Autonomous Intersection Management" pány Dresnerem a Stonem, na vytvoření bezkolizních plánů může být použit rezervační algoritmus. Naimplementování toho algoritmu je další úkol této bakalářské práce.

Na závěr experimentálně ověřím a porovnám oba tyto algoritmy autonomního vyhýbání.

Contents

List of Figures

Chapter 1

Introduction

Traffic jams and car accidents arise daily all over the world. Both problems cost a significant amount of money, but what is even worse, many people lives. Fortunately there are scientists, who are trying to make this situation better. Intelligent transportation system (ITS) is the field of science, which focuses on integrating information technology with transport means and transport infrastructure. One successful direction of research in ITS are autonomous vehicles. Such vehicle usually has many cameras and sensors for its orientation in the urban environment and no human driver. Autonomous vehicles are capable of steering, waiting on traffic lights, navigating from point A to point B, etc. They are capable of everything what human drivers can do, but they do it better. Moreover, robotic drivers cannot get impatient or drunk, which is the main cause of many traffic collisions.

Autonomous cars are no more sci-fi invention only, they are real. The organization DARPA held three autonomous vehicle competition called DARPA Grand Challenge. The last year, DARPA Grand Challenge 2007, is the most important for this thesis. Commonly known as DARPA Urban Challenge, this event "required teams to build an autonomous vehicle capable of driving in traffic, performing complex maneuvers such as merging, passing, parking and negotiating intersections." [4]

Autonomy is the key quality of autonomous vehicles, as well as typical attribute of intelligent agents, entities appearing in the multi-agent systems (MAS). As a subfield of Artificial Intelligence (AI), MAS is type of decentralised control system, consisting of autonomous agents, that interact with their environment and other agents. Such approach can be more effective or the only possible approach for solving many problems.

Recent works (such as Dresner and Stone [5] and on this work based doctoral thesis of M. Vasirani [8]) have proved, that the principles of MAS can be applied on the autonomous vehicles crossing the intersection. Communicating with the intersection manager, the autonomous vehicles try to reserve their path through the intersection. Further description of this algorithm is provided in Chapter 2.

Another area, where the multi-agent system approach can be used, is the autonomous airplane collision avoidance. Located at the Faculty of Electrical Engineering, the Agent

Technology Center (ATG) is "a university research center performing fundamental and applied research in the field of agent-based computing, multi-agent systems and agent technologies" [3]. One of the projects, on which ATG researchers are working is called AgentFly. Description on the project's webpage [2] says:

> AgentFly is a multi-agent system enabling large-scale simulation of civilian and unmanned air traffic. The system integrates advanced flight path planning, decentralized collision avoidance with highly detailed models of the airplanes and the environment.

For the AgentFly project, two collision avoidance algorithms were developed. One of them is the iterative peer-to-peer collision avoidance algorithm (IPPCA), which is based on pair-wise negotiation between airplanes and is described by Volf et al. in [9]. One of the objectives of this thesis is to study and then modify this algorithm to work with autonomous vehicles at the intersection.

Further in this chapter, in Section 1.1 you will find main objectives of this bachelor thesis and in Section 1.2 the structure of the thesis is shown.

1.1 Objective

First objective of this thesis is to sufficiently describe and explain the theory needed for understanding the rest of the thesis. This means mainly the domain of intelligent agents, trajectory planning and autonomous collision avoidance.

Second objective consists of two minor tasks. First is to design and implement algorithms for autonomous collision avoidance. I will mainly focus on the reservation-based collision avoidance algorithm from [5] and the iterative peer-to-peer collision avoidance algorithm developed by the ATG center. And to be able to do that, I need to create agent-based model of vehicles and crossroads, which is the second part of second objective.

Finally, the most important objective is to experimentally prove, that previously mentioned theories and algorithms in this thesis truly works in (simulated) practice.

1.2 Structure of the thesis

The thesis is structured in the following way:

1. Chapter 1 introduces the draft of autonomous collision avoidance, reveals objectives and show structure of this thesis.

2. Chapter 2 contains the state of the art, including the key concepts of the intelligent agents, trajectory planning and collision avoidance algorithms.

3. Chapter 3 contains all implementation-related informations. First, the agent-based simulator and its components (information representation, equations of motion, simulation loop), then the implementation of the collision avoidance algorithms are described.

4. Chapter 4 concerns evaluation of above mentioned algorithms. This includes setup of tests, data from them and discussion of the results.

5. Last chapter, Chapter 5, summarizes what has been done in this thesis, what objectives have been accomplished and what are the consequences of that.

Chapter 2

State of the art

In this chapter, necessary theory from various science fields is explained, so the reader would understand the rest of the thesis. The chapter is divided in three sections: In Section 2.1, the concept of autonomous intelligent agent and its properties are described. Section 2.2 presents various principles of planning in different environments. In last section, Section 2.3, basic description of collision avoidance methods along with dividing them into cooperative and non-cooperative methods is provided.

2.1 Agents

Modern approach to Artificial Intelligence (AI) is based on agents [7]. An agent is an entity which observes its environment through sensors and acts upon that environment using actuators. You can see simple reflex agent in Figure 2.1.

A robot (with specific task) can be perfect example of an agent. Its sensors can be camera, laser range finder or microphone. Robot's actuators would be typically motors and brakes. A human could also be considered to be agent, with eyes, ears, tongue as sensors and with arms, legs, mouth as his or her actuators. An agent could have no physical body whatsoever. A software agent receives keystrokes, file descriptors and network packets as sensory inputs and acts by displaying on the screen, writing in the files and sending packets.

An agent can exists on its own, but usually it isn't capable of achieving a goal only by itself, so it has to communicate and negotiate with other agents in that environment. Than the system is called multi-agent system.

Agents usually have some kind of memory. In agent's context is this memory called knowledge base (KB). Every intelligent agent should have at least these two properties - it should be rational and autonomous.

Rationality In general, a rational agent is entity that "acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome."[7] That means, the agent is capable of goal directed behavior.

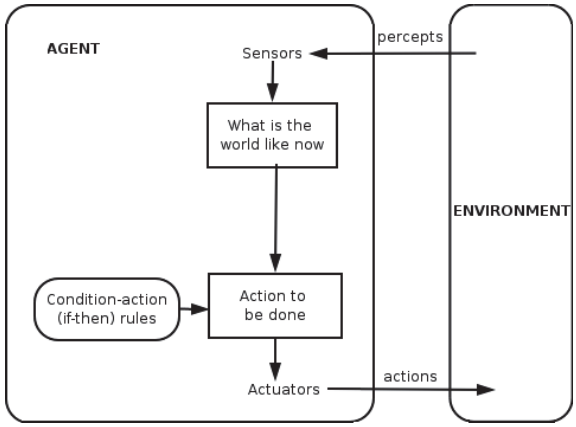


Figure 2.1: Simple reflex agent [1]

Autonomy An autonomous agent should rely more on its own percepts rather than on the prior knowledge of its designer. It should have some initial knowledge, but after sufficiently long time of staying in and learning from the environment, it can be completely independent of its prior knowledge.

Control The control in multiagent system is usually decentralised, which means that there is no central entity, which gathers all information from every agents and then decides what each agent should do.

2.2 Planning

The planning task in general is "the task of coming up with sequence of actions that will achieve a goal" [7]. But for different groups of people, this term means slightly different things.

In robotics the terms motion planning and trajectory planning are often used. An example of classical motion planning is the Piano Mover's Problem. Imagine you have an exact (virtual) model of house with a piano in it. The task is to find what actions should be executed to move the piano to another room without hitting anything.

In Artificial Intelligence, planning in discrete environment and discrete set of states is more common. This type of planning is often called problem solving. Examples of such problems are solving the Rubic's cube or achieve task that is modelled discretely, such as building stack of blocks.

Plans can be made to every environment. In some environments the planning is easier, in other it is very difficult to come up with a feasible plan, not to say optimal. Further on we will presume that the environment is fully observable, deterministic, finite and static (change only happen when the agent acts).

Plan refinement After new plan is made, there is another option, what to do with plan, else then executing the plan. Plan can be used for refinement [6]. A planner accepts the plan as input and determines a new plan that is hopefully an improvement. The new plan may take more problem aspects into account, or it may be more efficient. Refinement may be applied repeatedly, to produce a sequence of possibly improved plans, until the final one is executed.

The refinement approach has been used for decades in robotics in motion planning. Consider, for example, moving an indoor mobile robot. The first plan produces a collision-free path through the building. The second plan transforms the route into one that satisfies differential constraints based on wheel motions. The third plan considers how to move the robot along the path at various speeds while satisfying momentum considerations. The fourth plan incorporates feedback to ensure that the robot stays as close as possible to the planned path in case of unpredictable behaviour.

2.2.1 Discrete planning

Problems in discrete environment are considered to be the simplest, because their state-space is finite, or at least countably infinite. Therefore, characterization of these problems need neither geometric models nor differential equations.

Formulation of discrete feasible planning as in [6] :

1. A non-empty state space X, which is a finite or countably infinite set of states.
2. For each state $x \in X$, a finite action space $U(x)$.
3. A state transition function f that produces a state $f(x, u) \in X$ for every state $x \in X$ and action $u \in U(x)$. The state transition equation is derived from f as $x' = f(x, u)$.
4. An initial state $x_I \in X$.
5. A goal set $X_G \subset X$.

One example of discrete planning could be moving on the two-dimensional (2D) grid, where state is given by coordinates (i, j) and action is moving from one cell horizontally or vertically to the next cell (up,down, left,right).

General forward search A general template for forward search (as seen in [6]) can be seen in Algorithm 1. All alive states (state that have been encountered, but possibly have unvisited next states) are stored in priority queue Q. The sorting order (queue's priority function) is what differs between various search algorithms.

This algorithm says only whether plan to goal state exists or not, but the algorithm can be easily improved to give us such plan (if we each time associate x' with its parent x, we can then trace the actions back to the initial state).

Algorithm 1 A template for forward search, using priority queue Q

```
1: Q.Insert(x_I) and mark x_I as visited
2: while Q not empty do
3:    x ← Q.GetFirst()
4:    if x ∈ X_G then
5:       return SUCCESS
6:    end if
7:    for all u ∈ U(x) do
8:       x' ← f(x, u)
9:       if x' not visited then
10:         Mark x' as visited
11:         Q.Insert(x')
12:      else
13:         Resolve duplicate x'
14:      end if
15:   end for
16: end while
17: return FAILURE
```

Breadth first search In breadth first search, the priority queue Q is Last-In-First-Out (LIFO) queue. That results in such behaviour, that the graph is searched level by level. That means that the entire graph up to depth n is searched, before the algorithm proceeds to search level $n + 1$.

Depth first search Depth first search, as the title suggests, search preferably into depth, rather than into breadth. The algorithm keep expanding the first child node of the search tree that appears and thus, going deeper and deeper until a goal node is found, or until it hits a node that has no children. Then the search backtracks, returning one level up a goes into second child node. In the Algorithm 1, the priority queue Q is a First-In First-Out (FIFO) structure - a stack.

2.2.2 Planning in continuous spaces

Planning in continuous spaces is in robotics called motion planning. Motion planning frequently refer to motions of a robot in a 2D or 3D world that contains obstacles. For the purposes of planning it is important to define the state space. The term configuration space is vital in motion planning.

Configuration space The state space for motion planning is a set of possible transformations that could be applied to the robot. This is called a configuration space and is often shortened to $C - space$. The dimension of the configuration space corresponds to the number of degrees of freedom of the robot. Using the configuration space, motion planning

can be viewed as a kind of search in a high-dimensional configuration space that contains implicitly represented obstacles. Another term, obstacle region, $C_{obs} \subseteq C$, is subset of configuration space, in which the robot would be in collision with some obstacle.

A central theme throughout motion planning is to transform the continuous model into a discrete one. Due to this transformation, many algorithms from discrete planning are embedded in motion planning algorithms. One such transformation, called Incremental Sampling and Searching will be now presented.

Incremental Sampling and Searching The sampling-based planning algorithms presented in [6] and in here are strikingly similar to the family of discrete search algorithms. The main difference lies in step 3 below, in which applying an action, u, is replaced by generating a path segment, τ_s. Second difference is that the search graph, G, is undirected, with edges that represent paths, as opposed to a directed graph in which edges represent actions.

Most sampling-based planning algorithms follow this template [6]:

1. **Initialization:** Let $G(V, E)$ represent an undirected search graph, for which V contains at least one vertex and E contains no edges. Typically, V contains q_I , q_G, or both. In general, other points in C_{free} may be included.

2. **Vertex Selection Method (VSM):** Choose a vertex $q_{cur} \in V$ for expansion.

3. **Local Planning Method (LPM):** For some $q_{new} \in C_{free}$ that may or may not be represented by a vertex in V , attempt to construct a path $\tau_s : [0, 1] \rightarrow C_{free}$ such that $\tau(0) = q_{cur}$ and $\tau(1) = q_{new}$. τ_s must be checked to ensure that it does not cause a collision. If this step fails to produce a collision-free path segment, then go to step 2.

4. **Insert an Edge in the Graph:** Insert τ_s into E, as an edge from q_{cur} to q_{new}. If q_{new} is not already in V , then it is inserted.

5. **Check for a Solution:** Determine whether G encodes a solution path. As in the discrete case, if there is a single search tree, then this is trivial; otherwise, it can become complicated and expensive.

6. **Return to Step 2:** Iterate unless a solution has been found or some termination condition is satisfied, in which case the algorithm reports failure.

The role of the vertex selection method (VSM) is quite similar to that of the priority queue, Q, in the discrete space planning algorithms. Local planning method (LPM) computes a collision-free path segment that can be added to the graph. It is called local because the path segment is often simple (e.g., the shortest path) and travels a short distance.

Adapting Discrete Search Algorithms One of the most comfortable and straightforward ways to make sampling-based planning algorithms is to define a grid over C and conduct a discrete search. The grid does not have to have the same resolution on every axis.

The search graph $G(V, E)$ (the grid) is revealed "on the fly" such as in the discrete search. When expanding a node, its k-neighbours are generated. If generated vertex is in C_{obs}, do not add it into G. If q_I and q_G do not coincide with grid, add them to G and connect them to the nearest vertices.

2.2.3 Trajectory planning

For complete understanding of the term trajectory planning, it is important to know what a phase space is.

Phase space In robotics, most problems involve differential constraints that arise from the kinematics and dynamics of a robot. When the planner considers only kinematics of the robot, then the differential model can be expressed as constraint on the set of allowable velocities at each point in C. This results in first-order differential equations because only velocities are constrained.

However taking into account constraints on higher order derivatives seems substantially more complicated. This results into introduction of a phase space, which has more dimensions than the original C-space. Thus, there is a trade-off because the dimension is increased; however, it is widely accepted that increasing the dimension of the space is often easier than dealing with higher order derivatives.

Trajectory planning According to LaValle [6], the term trajectory planning has been used for decades in robotics to refer mainly to the problem of determining both a path and velocity function for a robot arm (e.g., PUMA 560). This corresponds to finding a path in the phase space X in which $x \in X$ is defined as $x = (q, \dot{q})$. Most frequently the problem is solved using the refinement approach mentioned at the beginning of Section 2.2 by first computing a path through C_{free}. For each configuration q along the path, a velocity $\dot{q}$ must be computed that satisfies the differential constraints.

Decoupled Planning Approach A typical decoupled approach involves four modules:

1. Use a motion planning algorithm to find a collision-free path $\tau : [0, 1] \rightarrow C_{free}$.

2. Transform τ into a new path τ' so that velocity constraints on C (if there are any) are satisfied. This might, for example, ensure that there no sharp turns on path and so a car can actually follow the path. At the very least, some path-smoothing is needed in most circumstances.

3. Compute a timing function $\sigma : [0, t_F] \rightarrow [0, 1]$ for τ' so that $\tau' \circ \sigma$ is a time-parameterized path through C_{free} with the following requirement. The state trajectory $\tilde{x}$

must satisfy $\dot{x} = f(x(t), u(t))$ and $u(t) \in U(x(t))$ for all time, until u_T is applied at time t_F.

4. Design a feedback plan (or feedback control law) $\pi : X \to U$ that tracks $\tilde{x}$. The plan should attempt to minimize the error between the desired state and the measured state during execution.

2.3 Collision avoidance

In environment, which has only obstacles, an agent should contain some collision avoidance (CA) component. Moreover in the multi-agent environment, this component has to take other agents into account. It can try to communicate and negotiate with them or at least try to predict their behaviour.

There are two types of multi-agent collision avoidance methods - cooperative and non-cooperative.

2.3.1 Cooperative methods

Cooperative methods, as title suggests it, are based on cooperation between individual agents. Typically, some kind of communication between agents is required to share all important information (such as sensor's data, future plans) with other agents.

There are many cooperative CA methods and algorithms, for example reservation-based CA algorithm from [8], auction-based methods (used e.g. in [8]) or the iterative peer-to-peer CA algorithm developed by ATG for CA of autonomous airplanes and described in [9].

2.3.2 Reservation-based algorithm

Reservation-based CA algorithm was proposed by Dresner and Stone in [5]. The algorithm presumes that there are two kind of agents - agent driver and intersection manager. The driver agent is an agent, who is driving driverless vehicle. Intersection manager is stationary agent, that manages the space of an intersection and handles the reservation requests from driver agents.

Description When agent driver approaches an intersection, it contacts its intersection manager and sends it a reservation request that contain all important info (arrival time, arrival speed, but also vehicle's characteristics such as maximal acceleration). The intersection manager processes the request, simulates the vehicle's passage through intersection and if there are no collisions, it accepts the reservation. But if there is any collision, the intersection manager will reject the reservation (preferably with brief justification of rejection) but may at least try to send a counter-offer back.

It can happen that the agent driver won't be able to arrive at the intersection on specified time (due to i.e. weather change or unexpected obstacle). The reservation system allows the agent driver to change the reservation or cancel it completely.

In the evening and in the night, when there is almost small traffic at the intersection, the interseciton manager might be able to handle the reservation requests, without them stacking. However during the day, when the traffic is high, the traffic manager has to store the reservation requests in a reservation set. In which order (called the intersection manager policy) should this set be iterated?

Choosing the right policy In the work by Dresner and Stone [5] the intersection manager policy was simple first-come-first-served policy (FCFS). M. Vasirani compared in his doctoral thesis [8] the FCFS policy with 4 universally stable policies, namely longest-in-system (LIS), shortest-in-system (SIS), farthest-to-go (FTG) and nearest-to-source (NTS). The LIS policy gives priority to the request of the vehicle which joined the system earliest. The SIS policy gives priority to the request of the vehicle which joined the system latest. The FTG policy gives priority to the request of the vehicle which still has to traverse the longest path until reaching its destination. The NTS policy gives priority to the request of the vehicle which is closest to its origin, i.e., which has traversed the least of its whole route.

In Vasirani's work, all the policies have comparable results as opposed to the traffic lights performance. He suggest using the FCFS policy since it does not require any additional information from the agent driver.

2.3.3 Auction-based algorithm

Auction-based algorithm for vehicle collision avoidance at the intersection is described in Vasirani's work. [8]. The auctioned good is the use of the space inside the intersection at a given time. According to the Dresner and Stone's model [5], an intersection is modelled as a discrete matrix of space slots. The set of items that a bidder can bid for is the set of space slots with given time-step. Due to the nature of the items, a bidder is only interested in bundles of items (rather than one individual space slot). Thus, the items must necessarily be allocated by a combinatorial auction. Agents send their bids along with the reservation requests, and the winner is determined according to a Vickrey (combinatorial) auction rules.

2.3.4 Iterative peer-to-peer conflict-resolution algorithm

Iterative peer-to-peer conflict-resolution algorithm was developed by ATG center originally for airplane-collision avoidance and is described in [9]:

> Algorithm starts with selecting the soonest conflict. It generates a set of possible flight trajectories for each airplane using predefined manoeuvres (e.g.

turn left, turn right, turn up, turn down, speed up, slow down). This manoeuvres are constructed to avoid the conflict. Each generated flight trajectory is tested for conflict with all other airplanes. If a collision with any airplane is found that would happen sooner than the currently solved one, such trajectory is removed from the generated set. Afterwards, sets for both airplanes are combined to create all possible pairs of trajectories (Cartesian product of these sets). Each pair is checked whether mutual collision persists. If there are some valid pairs, the best one is selected. Such selection can be based for example on sum of the utility value of each trajectory to reach maximum social welfare. If there is not any valid pair, the manoeuvres are applied again to generate wider range of trajectories. Newly generated and checked flight plans are added to the previously generated set. This process is repeated until a valid pair of generated trajectories is found and applied. When single collision is solved, the next soonest collision is selected. Iterations continue until all collisions are solved.

2.3.5 Non-cooperative methods

In non-cooperative methods (also called competitive), agents don't cooperate and are often called self-interested agents. They usually don't communicate with each other, either because the other agent cannot or is unwilling to communicate. The agents must rely only on data form their sensors. Prediction of other agents actions is often used as a tool to enable longer planning ahead.

Chapter 3

Implementation

This chapter describes (in three sections) implementation of various things - the agents, the algorithms and more.

In Section 3.1 the reader will find out, how I made the trajectory planning and that it is divided in three parts. Section 3.2 is devoted to the implementation of collision avoidance algorithms. Last but not least, Section 3.3 covers all details about crossroad simulator, which I made to be able to evaluate the CA algorithms.

In this chapter and this thesis, terminology (such as space slot) from Vasirani's work [8] is used.

3.1 Trajectory planner

As can be seen on Figure 3.1, the crossroad plan is divided into three logical parts. Part A corresponds with the time and place before the middle of the crossroad, part B covers the middle and finally part C is the part of plan, where vehicle has the intersection behind and rides out of the crossroad area.

3.2 Collision avoidance algorithms

Completing one of the objectives of this thesis, two collisions avoidance algorithms were implemented - the reservation-based algorithm (proposed by Dresner and Stone in [5]) and the iterative per-to-peer collision avoidance algorithm (IIPCA) described by Volf et al. in [9].

3.2.1 Reservation-based algorithm

Reservation-based algorithm belongs to the cooperative methods of collision avoidance, because this method requires communication between agents. A nice quality of this algorithm is, that it has got pretty straightforward implementation.

Algorithm 2 Reservation-based algorithm

```
1: get plans from all driver agents
2: pB ← planPartB()
3: for all maneuver m such that planPartA(m) is possible do
4:    pA ← planPartA(m)
5:    if pA has any conflict then
6:       continue with next m
7:    end if
8:    pB2 ← shift time of pB to start after pA
9:    if pB2 has any conflict then
10:      continue with next m
11:   end if
12:   for all maneuver n such that planPartC(n) is possible do
13:      pC ← planPartC(n)
14:      if pC has any conflict then
15:         continue with next n
16:      end if
17:   end for
18:   if pA, pB2, pC has no conflict then
19:      break
20:   end if
21:   if pA, pB2, pC exists then
22:      p ← concatenatePlans(pA, pB2, pC)
23:   else
24:      print "plan not found"
25:   end if
26: end for
27: save p into knowledge base
28: execute p
```

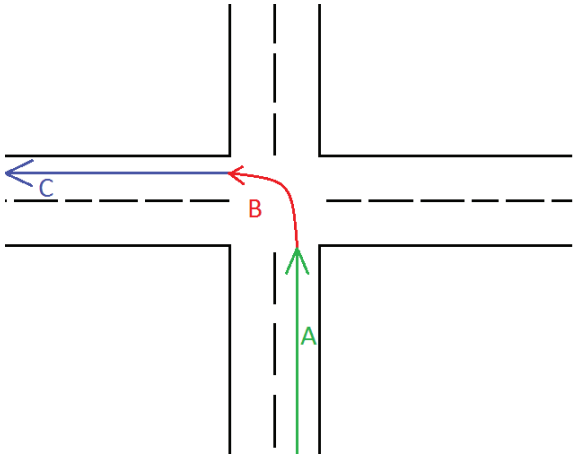


Figure 3.1: Three parts of crossroad

Reservation item The crossroad and near area is modelled as discrete matrix of space slots. One vehicle at one time can occupy one or typically more space-slots. The item, which is stored in the reservation, is set of such space slots along with the current time-step.

Description of the algorithm When a vehicle (with agent driver) is approaching the intersection, it asks all agent drivers near the intersection to send it their plans. The approaching agent driver saves these plans to its knowledge base (KB). Then the agent tries to plan its path through the crossroad. For every (future) time-step, it checks its (future) set of occupied space slots with the plans of other vehicles in its KB. If there would be a collision, agent is not allowed to execute this plan and it tries to replan its path.

After the agent driver has successfully made its plan, it saves the plan in its KB, to be able to provide it to who may ask for it. Finally the agent can execute the computed plan.

Description of the algorithm in the pseudo-code is shown in Algorithm 2.

In the reservation-based method of collision avoidance, all plans are final, meaning that once the plan is successfully made (it is conflict-free), it does not change.

My implementation My implementation of the reservation-based CA method works well, when the traffic is relatively sparse. The most "extreme" maneuver of a vehicle is just slowing down to low speed. However, because I did not completely finish the algorithm, the reservation-based planner cannot stop a vehicle, so it reports failure, when there are really many vehicles in the intersection area.

3.2.2 Iterative peer-to-peer collision avoidance algorithm

Iterative peer-to-peer collision avoidance algorithm (IPPCA) was developed in the ATG center and described in [9] by Volf et al.

Efficiency discussion It was successfully used for collision avoidance of autonomous airplanes. Because the airplanes plan and fly in three dimensional space, the algorithm is relatively successful and it takes only few iteration to find conflict-free flight plans. However, in the domain of intersection, the vehicles operates only in two-dimensional space and has less maneuvers to do. Hereby, I expect higher number of iterations, when applied on vehicles at intersection in comparison to the usage in the airplane domain.

In spite of my effort, I unfortunately did not manage to finish the implementation of IPPCA. It was due to lack of time and big complexity of the implementation task.

3.3 Crossroad simulator

Agents In the domain of intersections, agents are autonomous (computer) drivers called driver agents. Any agent can observe its and every other vehicle's position in the crossroad. It can also get information about other driver agents plans. Available agent's actions are steering, acceleration or deceleration the vehicle. Agent can also negotiate with other agents about their plans.

Representation of the world Position of each vehicle has 2D continuous representation - x and y coordinates are stored as floats with double precision. Then the coordinates can be rounded for visualisation and collision checking. Time is discretized in small steps - ticks Δt of value $\Delta t \sim 0.02$.

Equations of motion These equations are used to compute new position and new velocity of vehicle after one simulation tick. In following equations t denotes time, v denotes velocity (vector) and a is acceleration (vector) of vehicle.

Computation of new position s:

$$s = s_0 + vt + \frac{1}{2}at^2 \tag{3.1}$$

where s_0 is initial position (distance) of vehicle, t denotes time, v denotes velocity (vector) and a is acceleration (vector) of vehicle.

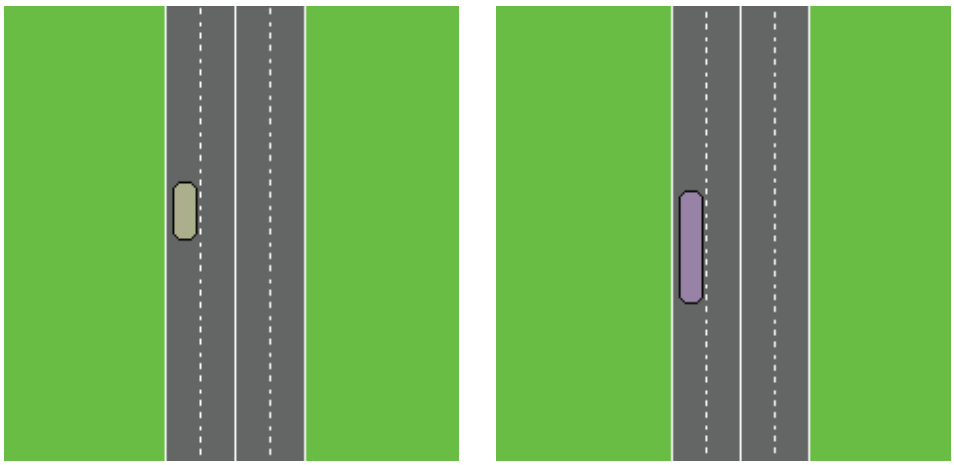
Computation of new velocity v:

$$v = v_0 + at \tag{3.2}$$

where v_0 is initial velocity (vector) of vehicle, t denotes time, v denotes velocity (vector) and a is acceleration (vector) of vehicle.

Collision checking For collision checking the float coordinates are first rounded and then checked cell-by-cell for every pair of vehicles.

The method of sampling is used to compute all occupied cells of the vehicle. Vehicle can occupy more or less cells depending on the current angle between the vehicle velocity and x axis.



(a) (b)

Figure 3.2: Graphical representation of (a) a car and (b) a bus

Vehicles In the simulator, there are two different types of vehicles - a car (shown on Figure 3.2a), which occupies two cells, when in horizontal or vertical position and a bus (on Figure 3.2b), which is four cells long and has lower maximal acceleration.

Simulation loop A simulation loop is the most important piece of code in every simulation. In this crossroad simulator, the simulation loop looks like this:

If the scenario is over, end the simulation, otherwise generate new *Vehicle* (and it's *AgentDriver*), if it's the right time (if scenario/random generator tells it). Then for every *AgentDriver* in the simulation do:

1. Check if the *AgentDriver* hasn't reached it's goal. If it has, remove it from the simulation.
2. Get new *Action* from *AgentDriver* and update corresponding *VehicleModel*.
3. Check if the *VehicleModel* is still on crossroad/map. If isn't, remove it from the simulation.

Look for all *VehicleModel*s out of the road and/or in collisions. Update (redraw) GUI. Sleep the simulation *Thread* for a while, then repeat.

More mathematical description of the simulation loop can be found in Algorithm 3.

Algorithm 3 Simulation loop

```
1:  for scenario is not over do
2:    if newGeneratedNumber = 0 then
3:      generate new Vehicle
4:      generate it's new AgentDriver
5:    end if
6:    for all AgentDriver agentDriver in agentDriverList do
7:      if agentDriver reached it's goal then
8:        remove agentDriver from simulation
9:      end if
10:     Action action = agentDriver.react(vehicleModel)
11:     vehicleModel.updateModel(action)
12:     if vehicleModel is not on map then
13:       remove vehicleModel from simulation
14:     end if
15:   end for
16:   look for all VehicleModels out of the road and in collisions
17:   update GUI
18:   sleep simulation Thread for 50 ms
19: end for
```

Chapter 4

Evaluation of algorithms

In spite of my effort, I unfortunately did not manage to evaluate both of collision avoidance algorithms in time.

I so far have evaluated my implementation of the reservation-based algorithm. It works well, but only when the traffic is relatively sparse. It reports failure, when there are too many vehicles in the intersection.

As I have already mentioned, I did not finish the implementation of IPPCA. It was due to lack of time and big complexity of the implementation task. This implies that I could not evaluate the IPPCA algorithm.

But I expect, that the iterative peer-to-peer collision avoidance algorithm will perform the collision avoidance better (meaning it would had bigger throughput in the intersection) than the reservation-based algorithm, thanks to its increased communication and the willingness to make a compromise between agent's own and other agents goals.

Chapter 5

Conclusions

In this bachelor thesis, I studied and described the domain of autonomous collision avoidance at the intersection. It is obvious, that the autonomous vehicles passing through the crossroad can be treated as an multi-agent system. Every intelligent agent in multi-agented system should have these two properties - it should be rational and autonomous. Communication between agents is also crucial, to be able to apply sophisticated algorithms and to come closer to the optimal throughput in intersection. Autonomous vehicles, when made by wise designer, satisfies all these requirements. More information can be found in Chapters 2 and 3.

Second objective of this thesis was to create agent-based model of vehicles and crossroads. I created a crossroad simulator to be able to study and evaluate the collision avoidance algorithms. This model was thoroughly described in Chapter 3.

The objective of implementing collision avoidance algorithm was unfortunately not met in its full scope. I implemented the reservation-based algorithm, but it still has some potential flaws. My version of algorithm works well, but only when the traffic is relatively sparse. It reports failure, when there are too many vehicles at the intersection. I did not finish the implementation of IPPCA. It was due to lack of time and big complexity of the implementation task.

The intended evaluation of these collision avoidance algorithms using various scenarios is missing as well. But I expect, that the iterative peer-to-peer collision avoidance algorithm will perform the collision avoidance better (meaning it would had bigger throughput in the intersection) than the reservation-based algorithm, thanks to its increased communication and the willingness to make a compromise between agent's own and other agents goals.

Bibliography

[1] ATMARAM, U. *IntelligentAgent-SimpleReflex.png* [online]. 2012. [cit. 10. 5. 2012]. Available from: <http://en.wikipedia.org/wiki/File:IntelligentAgent-SimpleReflex.png>.

[2] CENTER, A. T. *AgentFly - Projects - Agent Technology Center* [online]. 2012. [cit. 24. 5. 2012]. Available from: <http://agents.felk.cvut.cz/projects/agentfly/>.

[3] CENTER, A. T. *Home - Agent Technology Center* [online]. 2012. [cit. 24. 5. 2012]. Available from: <http://agents.felk.cvut.cz/>.

[4] DARPA. *DARPA Urban Challenge - Main page* [online]. 2012. [cit. 24. 5. 2012]. Available from: <http://archive.darpa.mil/grandchallenge/index.asp>.

[5] DRESNER, K. – STONE, P. A Multiagent Approach to Autonomous Intersection Management. *Journal of Artificial Intelligence Research*. March 2008, 31, s. 591–656.

[6] LAVALLE, S. M. *Planning Algorithms*. Cambridge, U.K. : Cambridge University Press, 2006. Available at http://planning.cs.uiuc.edu/.

[7] RUSSELL, S. J. – NORVIG, P. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2003.

[8] VASIRANI, M. *Vehicle-centric coordination for urban road traffic management: A market-based multiagent approach*. PhD thesis, Universidad Rey Juan Carlos, 2009.

[9] VOLF, P. et al. Convergence of Peer-to-Peer Collision Avoidance among Unmanned Aerial Vehicles. In *2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2007)*, s. 377–383, Silicon Valley, November 2007. ISBN 9780769530277.

Appendix A

Source codes

The agent-based model (called Crossroads) source codes are enclosed on the CD. The implemented collision avoidance algorithms are stored in following directory structure:

- Crossroads/src/plan/planners