"Towards the deployment of UAVs for fire surveillance"

Master Thesis

Malte Schultjan

July 17, 2012

Supervisors:
Prof. Dr. Ralf Möller
Prof. Dr. Herbert Werner

Hamburg University of Technology
Software Technology Systems
http://www.sts.tu-harburg.de/
Schwarzenbergstraße 95
21073 Hamburg
Germany

Declaration

I, Malte Schultjan, solemnly declare that I have written this master thesis independently, and that I have not made use of any aid other than those acknowledged in this master thesis. Neither this master thesis, nor any other similar work, has been previously submitted to any examination board.

Hamburg, July 17, 2012

Malte Schultjan

Abstract

The aim of this thesis is to provide the required foundation to use unmanned aerial vehicles (UAVs) for autonomous fire surveillance. In order to achieve this, the general requirements for such a system are discussed and a fitting general behaviour concept for these UAVs is introduced. This concept describes how their behaviour as autonomous agents should be divided into deliberative and reactive behaviours, thus creating a hybrid model. The basic hardware requirements are discussed, as well as the distribution of different tasks along this hardware platform. Research is reviewed in order to select and adapt algorithms that allow the establishment of a fire surveillance perimeter as well as the creation of a distributed map of the fire. This map consists of a set of points describing the fire perimeter, as well as their corresponding points in earlier measurements. These expansion observations, made during the creation of this map, can be used to predict future fire expansion for each point. Models which can be used for this prediction are developed and discussed. The two types of models proposed in this thesis are stochastical and Dynamic Bayesian Network models. The stochastical models have been implemented and tested using a simulator in order to chose the one that fits the application best as well as evaluation their performance. The algorithm that performed best in these simulations is an algorithm that includes the expansion history of itself and its direct neighboring points in order to create a Gaussian normal distribution describing the set of expansion values.

Contents

1 Introduction

The goal of this thesis is to provide a general description of the concepts and functions required to use autonomous unmanned aerial vehicles(UAVs) for fire surveillance. The focus for this description is on the theoretical and software side. Hardware will be discussed in some cases, but not as an in-depth discussion of all required hardware elements. Software concepts and algorithms required to offer the most basic functionality of UAVs, for example PID controllers, flight control and collision avoidance are not covered explicitly.

Section 1.1 describes the differences between some types of UAVs in order to convey a general understanding about the different types of aircraft that could be used, as a hardware platform, for the concepts described in later chapters. The motivation section 1.2 gives a short insight into why it makes sense to use autonomous UAVs instead of UAVs remotely controlled by pilots, surveillance planes, helicopters or satellite images.

Chapter 2, agent functionality architecture, first introduces some general design schemes for autonomous agent design. After this the assumptions made for agents, which are to be used for fire surveillance, are explained and a fitting design scheme is chosen and explained. A separation of different agent functions and their distribution between different hardware components is proposed and briefly elucidated.

The next chapter, UAV perimeter surveillance, proposes different methods for fire detection by UAVs, a necessary requirement for the next step, perimeter surveillance. Already existing perimeter surveillance algorithms are introduced in a research review section and two different additional algorithms proposed. One of those is a centralized approach that is not viable for some cases, but can be transformed into the second algorithm in order to enable decentralized remote surveillance.

Another important function is the creation of a model describing the observed fire. Chapter 4 introduces a few prerequisites required for the the following subsections. These explain how the fire perimeter can be described and the expansion of the fire can be predicted. The subsection on fire perimeter description proposes a model for fire description, reviews one method proposed in literature and offers an addition to this method. In the fire expansion estimation subsection different models that can be used to predict the expansion of a fire perimeter, described by a set of expansion points, are proposed and evaluated.

The next chapter, the prototype chapter, explains the necessary software required to make predictions based on probabilistic models and compares some of them in order to decide on the one that should be used to implement a prototype for fire prediction. The chosen software package as well as a few different inference algorithms are briefly introduced. A data generator, which can be used to create input data for the model is described together with the drawbacks of using this generator as well as

alternatives to it. The last section explains the graphical user interface and the basic design of the simulator used to evaluate the prediction algorithms for fire expansion.

Section 5.4 compares the results gained by using the simulator introduced in section 5.3 and evaluates them.

The last chapter offers a conclusion about the work done in this thesis as well as future work required.

1.1 UAVs - Unmanned aerial vehicles

In this thesis the term UAVs, unmanned aerial vehicle, is used continuously. This section offers an introduction into UAVs and their current and future use cases.

UAVs can either be controlled by a program or remotely by a pilot from the ground using either a radio or a satellite communication system. A hybrid version of these systems is possible. In this version a UAV can automatically fly to a position assigned to it via remote control. The pilot does not have to control every single step of the flight, but still controls where the UAV is going and takes control for important parts of the mission. This is similar to the autopilot in manned aerial vehicles.

The term UAV is not restricted to airplanes, but also includes zeppelins, helicopters or quadrocopters. A typical UAV airplane can be seen in figure 1.4, the UAV version of a helicopter is shown in figure 1.1 and figure 1.2 shows a quadrocopter.

Figure 1.1: The MQ-8B Fire Scout helicopter UAV [hel]

Each type of UAV offers different advantages quadrocopters for example do not require the mechanical linkages a helicopter needs to vary the rotor blade pitch angle as they spin. This makes them easier to produce and maintain.

Zeppelins can stay in the air for longer durations, if they use solar energy they can potentially stay airborne indefinitely. This is also possible for airplanes which has been shown, as a proof of concept, with the Helios UAV. Helios was destroyed in a crash in 2003 and can be seen in 1.3

The probably most well known use case for UAVs nowadays is for military surveillance or remote air-strikes. The idea of UAVs came up as early as 1915, when Nikola Tesla proposed that an unmanned airplane could be used to defend the USA. A few years later, in 1919, Elmer Sperry, the creator of

Figure 1.2: The Aeryon Scout Quadrocopter [qua]

Figure 1.3: The Helios solar power fueled UAVs

gyroscope and autopilot technology, destroyed a captured German battleship using a remotely controlled UAV, thereby offering the first proof of concept. In 1987 the first UAVs started to fulfill mission requirements, but only seven out of 105 flights passed. The first combat missions were realized in 1991 during the desert storm mission, searching for high value targets and Scud missiles in over 300 missions. [New04] The UAV used for this was a small airplane with a length of 4m and a height of 1m, the pioneer, shown in figure 1.4.

Figure 1.4: The pioneer UAV [pio]

In 1998 the german military "Bundeswehr" used UAVs, which are sometimes called drones, for surveillance in the Kosovo-war.

Since these first mission UAVs have been used more and more, not only for surveillance, but also for active combat mission. The first time a UAV was used to eliminate humans was in 2002 in Yemen, when a Reaper drone controlled by the CIAs Special Activities Division, destroyed an SUV, killing 6 men, including one local al-Qaeda leader.

Nowadays, especially under US President Barack Obama, UAV strikes are a regular alternative to typical military strikes. The number of UAVs owned by the US military between 2003 and 2009 increased from 256 to 1479, with an increase of over 100% nearly every year. In their UAV road-map the US military states that they want to further increase the number of UAVs constantly [Arm].

Between 2004 and 2012 there have been 330 UAV air-strikes in Pakistan alone, 278 under the Obama administration. The total reported number of killed humans is somewhere between 2,479 and 3,180 with between 482 and 832 civilian casualties according to the bureau of investigative journalism, a non-profit organisation for investigative journalism that won the Amnesty International Digital Awards two times in a row [Dro].

But UAVs also have civilian use-cases. They can be used for livestock monitoring, pipeline security

as well as private property security or geomagnetic surveys in search of oil or minerals. Other use cases include search and rescue missions, as well as the delivery of goods into areas that are hard to reach.

In the case of a tsunami or a flood, for example, where roads are blocked and humans trapped on top of houses or hills, the UAVs could find them and supply them with water cleansing chemicals as well as medication until they can be rescued.

Another use-case, the surveillance of wildfires will be discussed in this thesis in the hope that the civilian usage of UAV will increase and overtake the military sector as main scope of application.

1.2 Motivation

Although wildfires are not a significant problem in Germany as of yet, there are many countries that struggle with the danger posed by them, one being the United States of America.

Studies suggest, that there will be an increasing number of wildfires in the US due to climate change [M.D99]. But even today the damage done by them is already enormous. In 2011 there have been 74,126 wildfires in the US which destroyed 8,711,367 acres of land in the process [Cen]. The total cost of fire prevention, fire fighting and damage caused by fires has been $362 billion in 2008, which equals 2.5% of the US gross domestic product [JRH].

A big part of these costs are due to human losses, which include lives lost as well as medical treatment of injuries. Faster and more accurate information about forest fires does not only enable firefighters to fight fire more efficiently, but it also helps to mitigate damage to humans.

While this is written there are over a dozen active wildfires in Colorado alone, resulting in at least two casualties. At this point only 15% of the wildfires are under control.

An efficient rescue and control effort during a disaster event requires accurate and fast information about the environment. The more up-to-date information is, the better disaster managers are able to plan and realize rescue plans and prevent further damage from happening. In the past, information has been gathered by eye witnesses, agents of the involved crisis management department or airplanes. Airplanes are able to acquire the best overview data of the situation.Their usage, however, is expensive, might take time to organize and is not always possible due to danger to the pilots. This problem could be solved by using unmanned air vehicles (UAVs) to gather information.

Over the course of time prices for UAVs have dropped considerably. There are even UAVs available as toys for private consumers nowadays, while they were mostly used by the military or police before. This development makes it feasible to use a swarm of UAVs for perimeter surveillance, for example the expansion of fires or oil spills. In addition, it also allows to observe and map chemical, biological or nuclear contamination. But, because a swarm consists of a large number of UAVs, it is not practicable to let a human pilot control every single one of them. The more autonomous a swarm of UAVs can work, the more human disaster control managers can focus on fighting the tragedy.

1.2.1 Use case

The system this thesis describes is intended to enable the deployment of UAVs for fire surveillance. The course of events for this would start with the discovery of wildfires, either through reports by

humans, satellite images or warnings from sensor networks. In order to observe the fire and get as fresh information as possible about it a swarm of UAVs is sent into into the general direction of the fire in fire search mode. As soon as they find it they establish a perimeter around the fire. The algorithm for this distributes the agents around the fire in a way that minimizes the latency of information about the whole fire perimeter, which is done by evenly distributing the agents around the fire. A map of the fire expansion is created that consists of a number of points describing the fire as close as possible. The expansion speed of the fire is observed and the UAVs create a stochastical model that allows them to infer future expansion speed. If this speed is higher than a preset parameter or if the fire moves into the direction of a village or city the crisis manager can be informed that he has to act. If a fire does not expand in a threatening way it is not necessary to extinguish it right away, information about the expansion can be relayed to the crisis manager in regular intervals. Several teams of UAVs could be send out to find and observe all wildfires in an area, thereby creating a map of them with each fire weighted by its importance. This artificial intelligence helps the crisis manager to focus on the most important fire.

2 Agent Functionality Architecture

When designing an agent or a system of agents it is important to decide about the level of deliberation and planning involved in the decision process of each agent. Over the last decade most agents could be classified, to a certain degree, as one of two types: Deliberative or Reactive.

The first approach, in literature often called traditional, is the deliberative approach. This agent type has a hierarchical control system consisting of three main parts, a sensing system, the planning system and an execution system [Gat98]. This paradigm is also known as Sense-Plan-Act (SPA). Information is gathered by the sensors and used to modulate a model of the world around the agent. Based on this information the planning system calculates different possible actions and their expected outcome, evaluates these choices and decides for the one with the highest probability of serving the agents purpose. While the computed steps are being executed, the sensors gather new information leading to a changed model of the world and possibly a new evaluation of the actions. This kind of approach has first been described by Schwartz and Shair in 1983 [SJ83] and has since been refined and described in various additional papers and books. SPA is a top-down approach to AI creation, because the big picture is always in the center of the deliberation process. It can be used to combine multiple goals and constrains in a complex environment [HR08]. It has, however, three main drawbacks. First of all it is difficult to create a sufficiently accurate model of the world. The agent only knows what it can infer from its sensor data in addition to the model it has been programmed with. Because this model can get indefinitely bigger and more complex and the agent has to calculate the expected outcome of all his possible actions, intensive reasoning is required. Therefore delays in task execution, for example navigation are to be expected and high computing power is required to develop a plan as fast as possible. While this is feasible for virtual agents, because they don't have to move in reality, robots, for example UAVs, have to be able to transport the computing equipment as well as enough energy to supply it with power. Last but not least, if one of the modules of this hierarchical control engine does not function properly the whole system might fail, resulting, for a UAV, in a crash and a lost agent.

These problems, in particular the necessity to get a quick reaction from robots, in order to be able to create animal- or human-like machines, lead to the development of reactive, or behaviour based agents. Behaviour based agents were first proposed by Brooks in 1986 [R86]. The actions of these agents are solely based on the sensor input. In contrast to the deliberative approach there is no planning process involved. The goal of the agent is reached by a combination of different behaviour layers, each implementing one part of the different behaviours of the agent. This could for example be collision avoidance or the urge to recharge its battery if the power gets low. This approach does not focus on the main goal of the agent, but just describes different reactions to various events, making it a bottom-up approach. There are two main types of behaviour based architectures. One proposed by Brooks,

subsumption architecture and another one, proposed by Arkin in 1989 [RC89], the motor schema.

In the subsumption architecture all different layers receive the input of a specified set of sensors. When this sensor input matches certain criteria an output is triggered. This output, in turn, triggers a specific action. The different layers have different priorities. Every time more than one output is triggered, the output with the highest priority suppresses all other outputs. This means the highest priority output that is active defines the complete behavior of the agent. The highest priority in this architecture should always be given to the action that requires the fastest reaction time. This could be, for example, obstacle avoidance. When an agent is close to harming itself or others or moving into a situation from which it might not be able to escape later on, the most important task is avoiding this situation, even if it does not directly serve the main goal. Although there has been a large number of subsumption based robots that have been tested and put into service, this approach has one problem. Because there is always only one active behaviour, some of the lower level behaviours may be constantly suppressed by others and therefore never reach their goal. This can endanger the success of the overall goal the agents designer had in mind, because there is no planning involved that keeps this goal in the focus.

The motor schema uses input from all different behaviour layers to compute one common output. To achieve this, every behaviour layer computes an output vector based on the sensor inputs. These vectors are then added to compute one common behaviour vector. The decision how to add these vectors is purely the designers choice, he can therefore add more importance to some behaviours in order to create a system that reaches its goal.

One of the main advantages of reactive or behaviour based control is the quick reaction time. Because there is no planning involved, inputs can more or less directly be transformed into outputs, therefore enabling robots to reach, for example, the reaction speed of animals. In addition to that no complete model of the environment is needed, because, whenever information is missing for one behaviour layer in order to compute an output, another one takes over. For the same reason single modules of the behaviour can fail without the whole system failing. If one behaviour telling the agent to modify its surroundings stops functioning, this part of the task is not fulfilled correctly anymore, but other behaviours, like collision avoidance, still work and the agent is still able to fulfill other missions. The main problems with this design are coordination and high level tasks. To reach a high level goal some coordination between all behaviour layers is necessary, both subsumption architecture and motor schema offer basic solutions for this, but it is still the designers job to create a coordination between the different behaviour levels that reaches the main goal in the end.

Deliberative and reactive agent designs are completely different in their approaches, both with different advantages and disadvantages. They can be seen as two different extremes of agent designs. The reactive approach is more efficient in complex, uncontrolled and constantly changing environments, where the sensor input at any given moment is enough to complete the task. In these situations plans made by the planning layer in deliberative agents would have to be changed constantly and would not work efficiently. In controlled environments, for example the production line of manufacturers, a plan made by an agent can, in most cases, be executed without any changes and therefore can deliver higher efficiency.

Since some agent systems need fast reaction time and adaptability as well as advanced planning, to guarantee that multiple or complex goals are reached in optimal time, a combined approach was required. If there is advance knowledge about the world the agent acts in, an addition of deliberation to a purely reactive system can improve its performance. In 1998 such a hybrid approach was described by Arkin [RC98]. The deliberation layer can, for example, make changes to the different behaviour modules of the reactive layer based on gathered knowledge about the world and the effects of actions executed by the reactive layer. The main problem with this approach is the interface between the deliberative and reactive layers. This has been the focus of many research projects over the last years. In [PB09] the addition of machine learning to the deliberation layers understanding of the reactive layers is suggested to optimise the cooperation of the two layers automatically and thereby simplifying the creation process of agent control system, because the optimal design of the different layers does not have to be known in advance.

2.1 Assumptions

In order to devise a fitting architecture which enables UAVs to autonomously find fires and patrol fire perimeters a number of assumptions have to be made.

The different wildfires do not stay in one position, but can move, extend or retract depending on different factors. Therefore they can not have a fixed start- or endpoint for the perimeter.

During the fire patrol mode UAVs can only communicate with each other when they are in close range to each other. This can be either when UAVs travelling into opposing directions pass each other or when UAVs patrolling adjacent perimeter sections meet on their endpoints. This is dependent on the patrol algorithms they are in. The possible patrol algorithms will be described in chapter 3.

UAVs require enough fuel to find a fire and then patrol around it without constantly having to refuel. The more the UAVs have to refuel and the further away the base station where they have to refuel is, the less efficient the fire surveillance gets. The minimum fuel requirement for the agents is the amount required to reach the fire, fly around it once and fly back to the base station again.

2.2 UAV Agent Design

Figure 2.1 shows the general design of the UAV agents. In this section I will describe the different parts of the agent and how they work together.

All decisions of the agent are based on its sensor input. This comprises information from the camera, the gyroscope sensor, GPS information, as well as information about the position of other UAVs, if the UAV is in formation control mode. The information for formation control is gained by using a communication protocol between all agents in the formation. This protocol ensures that every agent knows its distance from one or more leader agents in X, Y and Z direction.

The sensor inputs are processed in the processing and distribution units and then distributed or provided to the fire estimation, decision making and flight control modules. This process is subdivided into two modules, because it should be executed on separate hardware, which is explained later.

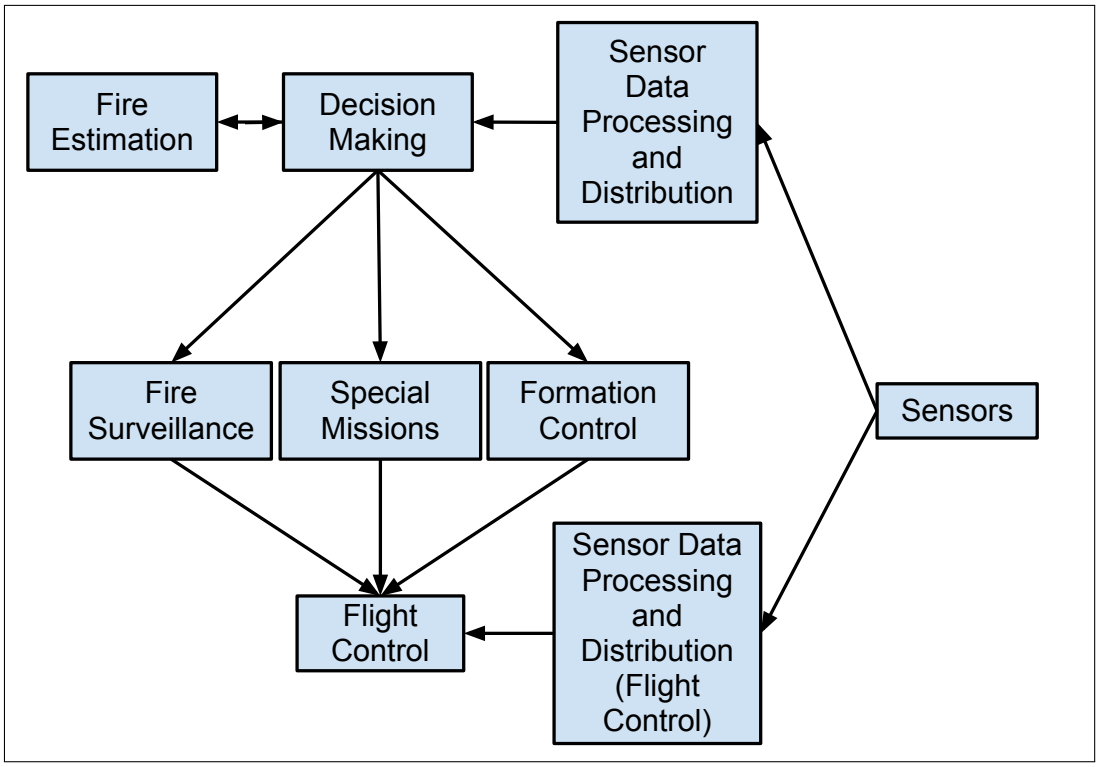

Figure 2.1: General design of the UAV Agents

The information from the gyroscope sensor, the GPS sensor and information about the position of nearby UAVs is prepared for the flight control module. Information from the camera that is useful for collision avoidance is computed and provided as well.

Sensor data from the camera is processed to get information about the current distance to the fire perimeter. This information together with all other sensor data is transferred to the decision making unit. The decision making unit bases its decisions on this data, as well as the information from the fire estimation module.

If for example all UAVs are still in communication range and there is no fire in the direct vicinity, the UAV assumes that it is in fire search mode, it searches for the next fire either based on a general direction it has been sent off to or randomly. For this mode all UAVs still stay in communication range of each other and the formation control module is needed to insure the best formation for the fire search and the constant availability of communication. For this the formation control gains access to GPS information as well as the estimated position of the other UAVs. It computes an output vector that describes the direction the UAV should fly into and sends it to the flight control module.

If a fire has been found the fire surveillance mode is started. For this mode the fire surveillance module gets information about the position of the fire in comparison to its own position from the processed camera data. With this information it can compute a course that will take it along the current part of the fire perimeter and send this information to the flight control module. In this mode information for the fire estimation can be collected from the camera and GPS sensor.

The special mission module gets input information from the decision making module as well as all

sensor data that it might need to fulfill its special mission. These missions could be for example the search for humans, for which an infrared camera would be useful. Important information may need to be relayed to the base station, in this case the special missions module plots the best course to the nearest base station or communication uplink. Depending on the special mission the UAV computes a direction vector and sends it to the flight control module.

If there is a problem with sensors malfunctions, low battery charge level or the flight control model is not able to function within specified parameters, for example because of too much wind or damaged rotators, the decision making module can decide to either abandon the mission for repairs or ignore the damage or battery level, if the current mission is considered more important and still reachable.

The flight control module is the module implementing the most basic UAV functionality. It includes, for the example of a quadrocopter, the proportional-integral-derivative controllers (PID) that control the rotors to keep the UAV in an upright position, even with wind, and allow for simple fly-into-direction commands. In addition to this it implements the functionality for collision as well as fire avoidance. These functions have a higher priority than flying into the right direction. So whenever the UAV gets a fly-to command it tries to fly into this direction, but if its way is in any way obstructed it avoids collision and flies around to a position that is as close as possible to the original target destination. This module also includes a simple fly-home command that can be executed if there has been no command from the fire surveillance, special mission or formation control module for a preset period of time and the decision making module is, therefore, assumed to be unresponsive. Before flying home the flight control module has the ability to restart the decision making module in an attempt to fix the problem.

2.2.1 Deliberative and reactive functions

All the functions described in the previous section can be divided into reactive or deliberative functions in order to create a hybrid agent system. This allows to use the advantages of both designs.

The critical functions of the UAV, meaning everything that keeps it from crashing are highly time dependent and should therefore be implemented as a reactive system. This comprises the PID controllers as well as collision avoidance. Collision avoidance, in the case of fire surveillance, also includes avoiding flying directly over fires, because the updraft and heat created by these can lead to malfunctions. These functions have the highest priority. The next important functions are not essential to avoid the destruction of the UAV, but they are required in order to provide necessary basic functionality for higher levels. These functions include keeping a fixed position or direction with influencing factors like wind as well as the fly-to command. The fly-home command has the lowest priority, only if there is no command from the deliberative layers and restarting the system has not worked it is executed automatically. It can, however, be directly invoked by the deliberative layer with a higher priority.

The basis of the fire surveillance algorithm, detecting the fire and adjusting the flight path accordingly is also time dependent and can be implemented as a reactive function, its priority, however, is not as high as the critical functions. It is more important to avoid a crash then to keep the appropriate distance from the fire. A more detailed description of the surveillance algorithms and the fire detection can be found in chapter 3.

The formation control algorithms can also be implemented as reactive functions. They comprise either keeping a fixed offset from the leader, or a fixed behaviour whenever they meet another UAV. The first formation control is required when the UAVs are in fire search mode, the second one when the UAVs have established a perimeter around a wildfire.

All other behaviour, such as deciding which information should be relayed to the base station, leaving the surveillance perimeter in order to fulfill a special mission or deciding how many UAVs are required to observe a fire are higher functionality and should be implemented as deliberative functions.

2.2.2 Physical separation

The decision making process and the fire expansion prediction as well as the image processing and formation control, special missions and fire surveillance require high computing power. Since these computations are not as critical as collision avoidance and flight control they should not interfere. The best way to do this is to have different hardware for both reactive and deliberative systems. In addition Digital Signal Processors can be used to process the image stream data from the camera to reach an even higher level of modularisation. With these different hardware modules the flight control will even work if the decision making hardware module has a malfunction and allow the UAV to safely return home. Depending on how important reliability is in comparison with weight and price the UAV could even have two flight control modules to add extra redundancy and by this achieve a higher reliability.

Figure 2.2 shows how the different functions are distributed between different hardware components.

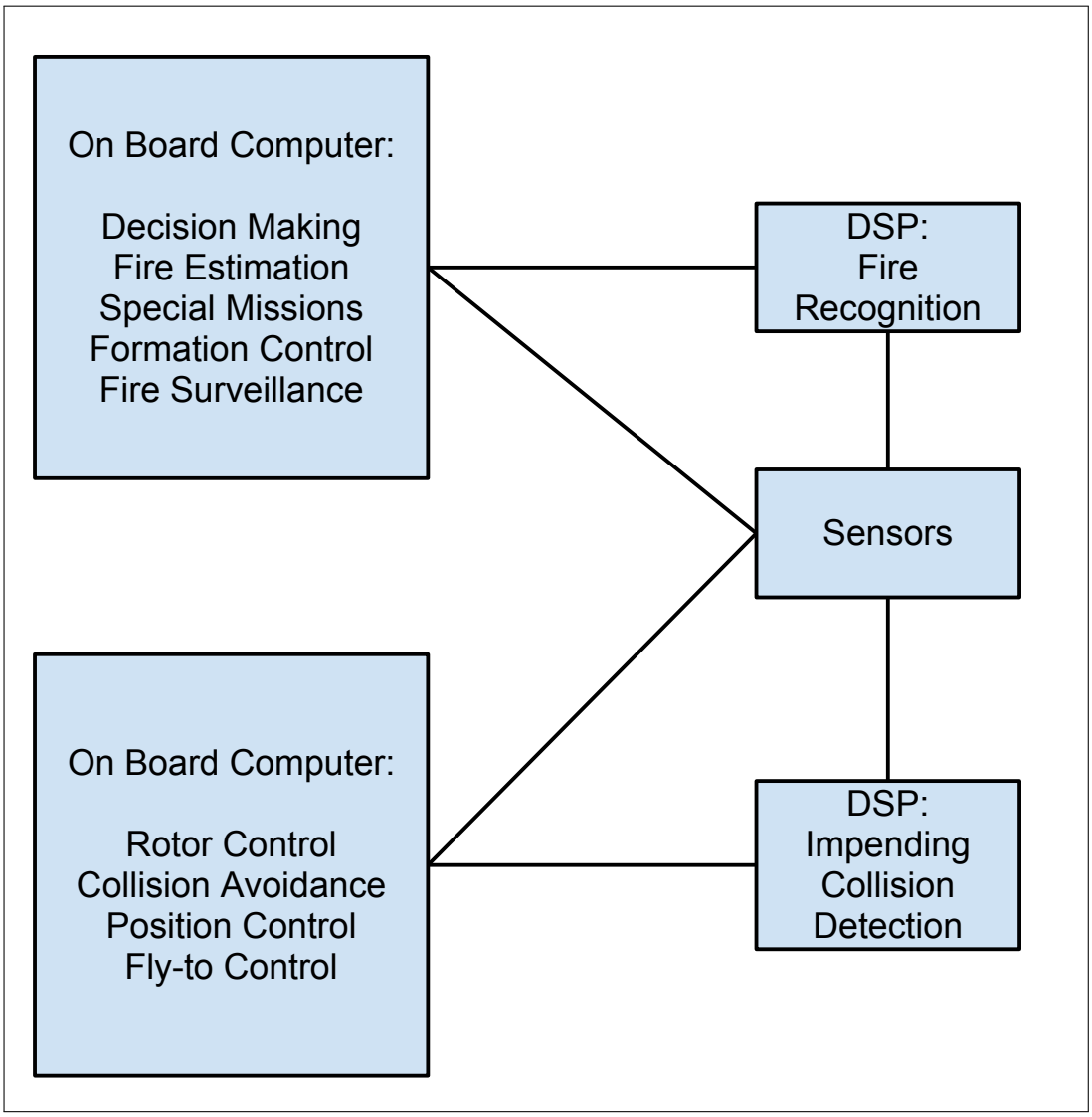

Figure 2.2: Functions divided by different hardware components

3 UAV perimeter surveillance

3.1 Fire detection

In order to find wildfires and then establish a perimeter around them, the UAVs need to be able to sense fire in their vicinity. This can be done using different methods including smoke detectors, infrared cameras or normal cameras. Detection using smoke detectors is possible, but not reliable for the surveillance mode, because the smoke is affected by wind and therefore not a good indicator for the perimeter of the fire. It can, however, help during the fire search mode. Smoke can be detected even if the source of the fire is hidden, for example by mountains. [ABU03] describes a method which allows to detect smoke up to a distance of 2.5km. This is further than the typical range of around 750 meters for infrared cameras and could therefore help to increase the area covered by one UAV in search mode. This method uses Lidar, light detection and ranging, to detect the smoke as well as the current distance to it, additional information that is not available directly from infrared images.

The most straight forward method is to use an infrared camera, this method has the advantage that fires can easily be detected because of the heat it radiates. Only very simple image recognition techniques are necessary to detect the area of the fire in an image. The UAV is able to use this information to adjust its trajectory, either into the direction of the fire, if it is in fire search mode, or along the perimeter of the fire, if it is in fire surveillance mode.

For the fire search mode the UAV would just adjust its trajectory in a way that the video, produced by a camera recording in flight direction, would show the fire in the middle of the image. In order to fly counterclockwise around a fire the UAV would adjust its course to keep the fire in the left half of the image, while the right side shows regular forest area, or, for an infrared camera, a cooler area. This positioning should be based on several pictures or an estimation made by using the video stream in order to minimize steering of the UAV. Figure 3.1 gives an idea about how the image produced by the UAVs camera should look for such a surveillance flight.

The shown image is not taken from directly above the fire, but with an offset, meaning that the camera is not pointing directly into the flight direction of the UAV, but to the left of it. This is required in reality because UAVs can not fly directly above the fire without being affected by the updraft created by the fire. This simple method of fire recognition allows the UAV to fly in a save distance to the fire while still being able to adjust its course accordingly. The method of keeping the fire in the left half of the screen also acts as a smoothing function for the fire perimeter. There could be, for example, a small bay in the fire perimeter as shown in figure 3.2. If the UAV would only see the fire directly next to it, it would first fly a curve to the left and shortly after another to the right. For small bays however this is not required, because an upper bound for the fire perimeter is more important than an absolutely precise measurement. The intensity of this smoothing function can be adjusted by deciding how far

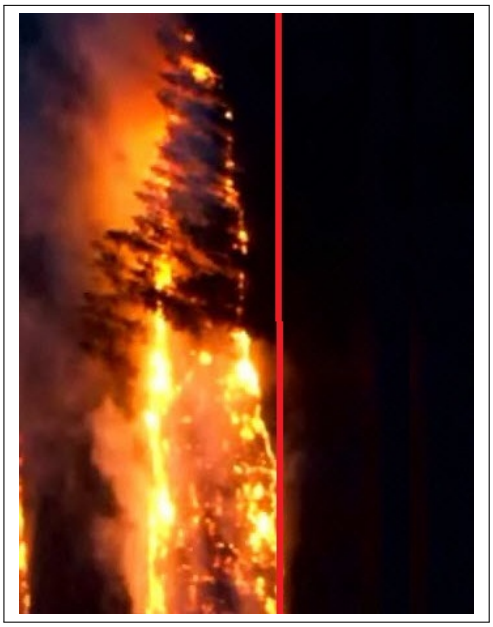

Figure 3.1: Wildfire with camera position for perimeter surveillance trajectory

into the UAVs trajectory the UAV is recording. This can be achieved by changing the cameras angle from the horizontal position into the direction of the earth.

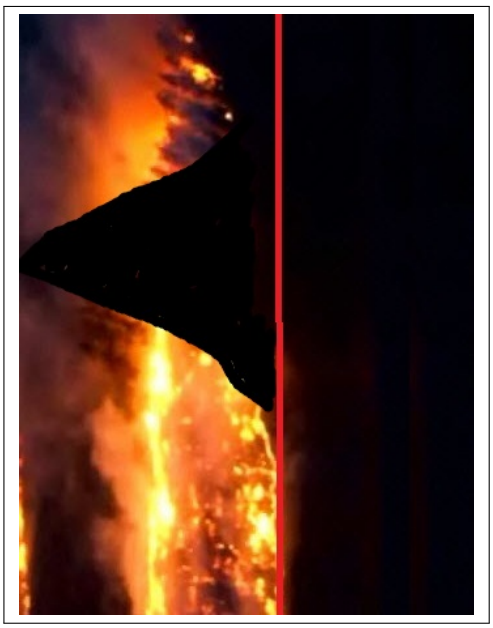

Figure 3.2: Bay in the fire

The method using infrared cameras could also be executed by using normal cameras, which have the advantage of being cheaper. Fire detection using a normal camera is not as easy as using an infrared camera. There are, however, different image processing techniques that can be used to detect fire in either a video stream or in single images. One example can be found in [MLB06], where the temporal variation of fire intensity is used to identify fire in videos. Additional information on fire recognition in images as well as in videos can be found in [Nga12], a diploma thesis in the STS institute.

3.1.1 Sensor Model

The input data for the prediction model consists only of position data produced by the GPS module. GPS technology, especially when used in a wide, open area, for example the air above a forest is already very precise and robust against errors. This accuracy can even be improved by combining the GPS module with an Inertial Measurement Unit. There are several error models that can be used to improve the accuracy and error robustness of this combination. The usual models use a 6-state error model that includes three random walks for gyroscopes and another three for the accelerometers. This model can be improved by adding three scale factors for the gyroscopes as well as the accelerometers and adding a precise gravity model as well as signal de-noising [al.05]. A comparison of these models including implementation and test flights can be found in [HS]. The 12-state model results in accuracy for the position of 5cm and for the velocity of 3cm/s, which is more than enough for the expansion prediction.

The GPS/INS (Inertial Navigation System) position, however, only measures the position of the UAV, not the fire perimeter. The accuracy of the GPS/INS measurement as an estimation for the fire perimeter is therefore based on the accuracy with which the UAV can fly along the perimeter of the fire. For this estimation the real distance from the fire is not as important as the variation between passes. If the fire does not expand, but the UAV flies along the perimeter in varying distances to the fire, the predictions made by the prediction algorithms are affected. These algorithms can handle noisy data, however the accuracy of the predictions is decreased with an increasing noise factor. The distribution describing the expansion will have a higher standard deviation.

The positioning of the UAV depends on the accuracy of the fire detection described in 3.1 and has to be tested and possibly improved if the typical distance to the fire is too random.

3.2 Perimeter surveillance - Research Review

In the last years, the research area of UAVs and autonomy of UAVs has received significant attention. Solutions have been proposed for a decentralized solution in perimeter surveillance.

In [DWCM06] , a solution for an algorithm is proposed that assumes that the perimeter could be mathematically transformed into a circle, but also works if it can be transformed into a line. Two important performance indicators of a perimeter surveillance algorithm are latency of the information and its refresh rate. Therefore the algorithm minimizes the latency for information from the furthest point of the perimeter and maximizes the update frequency for a given number of UAVs. Additionally, it is able to cope with changing perimeters, changing numbers of UAVs, allows for role changes between UAVs that meet to enable refuelling and is distributed. Their algorithm achieves the low latency configuration by sharing the travelled distance between neighbouring UAVs and computing a new meeting point that allows both to travel the same distance until their next meeting, therefore optimizing the efficiency of the UAV that travelled the smaller distance before. However, their algorithm has two major drawbacks. While achieving the low latency configuration in finite time it does so by converging to it step by step. After the first two UAVs met they agree on a new meeting point equalizing their travelled distance, however, as soon as one of them meets the next neighbour this value is very probable to

change, which in turn requires the first pair to adjust its meeting point again. Tests showed that their algorithm converges in around 7T normalized time, where T denotes the time a UAV needs to traverse the entire perimeter once. This is by itself not yet optimal in comparison to other algorithms that will be introduced later. In addition this time increases with an increasing number of UAVs resulting in a bad scalability. The second drawback is that their algorithm requires UAVs to loiter at certain points along the perimeter when waiting for their neighbours which results in wasted time and fuel of the loitering UAV.

[DKH07] describes an algorithm that is decentralized, converges to optimal behaviour in finite time, explicitly accounts for communication range limitations and works with changing parameters. Changing parameters in this case could be changing numbers of UAVs or dynamic perimeter length. Their solution assumes a perimeter with two end points while expanding fires, oil spills or toxic contamination should usually be observed on all edges and therefore they usually have a perimeter that forms a closed loop. This can be handled, however, by considering this loop a linear perimeter with both end points at the same point. The minimum latency configuration is reached by having each UAV store the number of UAVs and the length of the perimeter to the right and left of it. This value is updated by communications between UAVs that meet and enforces that the UAVs are ordered after the information has been propagated throughout the perimeter. The UAV that first arrives at the end of the perimeter is UAV Nr. N, N is still unknown at this point and has no right neighbor. When it encounters the first UAV on its way back along the perimeter in sets its left neighbor count to 1 while the UAV it met sets its right neighbor count to 1 and both UAVs change direction again. As soon as one of the UAVs reaches the starting point of the perimeter it knows that it has no left neighbor and knows the exact number of right neighbors as well as the whole perimeter length. With this information it can compute the exact length of its own perimeter and transmit that information as soon as it rendezvous with its right neighbor again. This algorithm can be seen in Algorithm 3.1, the meanings of the variables are as follows:

- P_{R_i} is the length of the perimeter to the right of agent i
- P_{L_i} is the length of the perimeter to the left of agent i
- P is the complete perimeter length
- N_{R_i} is the number of agents to the right of agent i
- N_{L_i} is the number of agents to the left of agent i
- N is the complete count of agents
- n is the relative position of the agent in the formation
- S_i are the endpoints of the agents perimeter section, formatted as {left,right}

After 3T the low latency configuration is reached, 1T for the first UAV to reach the perimeter end, 1T for the information to travel back to the starting point and 1T for the now complete information

Algorithm 3.1 Decentralized perimeter surveillance - Perspective of Agent i

if agent i(left) rendezvous with neighbor j(right) **then**
Update perimeter length and team size:
$P_{R_i} = P_{R_j}$
$P_{L_j} = P_{L_i}$
$N_{R_i} = N_{R_{j+1}}$
$N_{L_j} = N_{L_i}$
Calculate team size $N = N_{R_i} + N_{L_i}$
Calculate perimeter length $P = P_{R_i} + P_{L_i}$
Calculate relative index $n = N_{L_i} + 1$
Calculate segment endpoints $S_i = \{\lfloor n - \frac{1}{2}(-1)^n \rfloor \frac{P}{N}, \lfloor n + \frac{1}{2}(-1)^n \rfloor \frac{P}{N}\}$
Communicate S_i to neighbor j and receive S_j
Travel with neighbor j to shared border p
Set direction to monitor own segment
else if reached left perimeter endpoint **then**
Reset perimeter length to the left $P_{L_i} = 0$
Reset team size to the left $N_{L_i} = 0$
Reverse direction
else if reached right perimeter endpoint **then**
Reset perimeter length to the right $P_{R_i} = 0$
Reset team size to the right $N_{R_i} = 0$
Reverse direction
else
Continue in current direction
end if

to travel back along the perimeter. This time is smaller than 2T if the number of UAVs is known in advance or if the latest starting point for a UAV is known to be smaller than 1T.

This algorithm requires a fixed endpoint for both sides of the perimeter. Since a circle around a fire, the assumed perimeter in this thesis, does not have a fixed starting or endpoint this point needs to be defined. Because the fire is moving, the chosen point should not be a location, but rather move with the fire. To manage this all agents are initialized when they start their surveillance. After finding a fire the group of UAVs decides on which agent starts into the right direction and which into the left as well as which agents start first. The agents can update their team size to the left and right accordingly. An agent that starts to fly around the perimeter clockwise can add the number of agents that have already started, as well as the agents that are supposed to start counterclockwise, to his count of agents right from it. The number of agents left to it is the number of agents that are still supposed to start flying clockwise. The same is done for agents starting into the counterclockwise. Whenever agents meet their agent counts are compared. The agent with the lower count of agents left of him is assumed to be closer to the left endpoint and the agent with the lower count of agents right of him closer to the right perimeter endpoint. If agents meet which have a difference between their left counts $|N_{L_i} - N_{L_j}|$ and their right counts $|N_{R_i} - N_{R_j}|$ that is higher than 50% of the complete number of agents N the point where those two agents met is assumed to be the new perimeter endpoint. This choice guarantees that the agents decide on a fixed endpoint unless more than 50% of the agents are lost during 1T. If a lower

number would be chosen, agents might assume perimeter endpoints whenever single UAVs are lost at an arbitrary location around the perimeter, because this would result in a difference between left and right counts that is higher than 1. If more than 50% of neighboring agents are lost at a location around the perimeter that is not the perimeter endpoint, this will result in a second perimeter endpoint being created. However, this creation would not break the surveillance algorithm completely but only result in two teams of UAVs patrolling different sides of the same fire. If there is an uneven number of UAVs, however this algorithm would not result in a stable endpoint, because the UAVs at the end of the perimeter would never arrive at the same time, therefore this point and with it all other points would constantly shift. Figure 3.3 illustrates the problem for three UAVs.

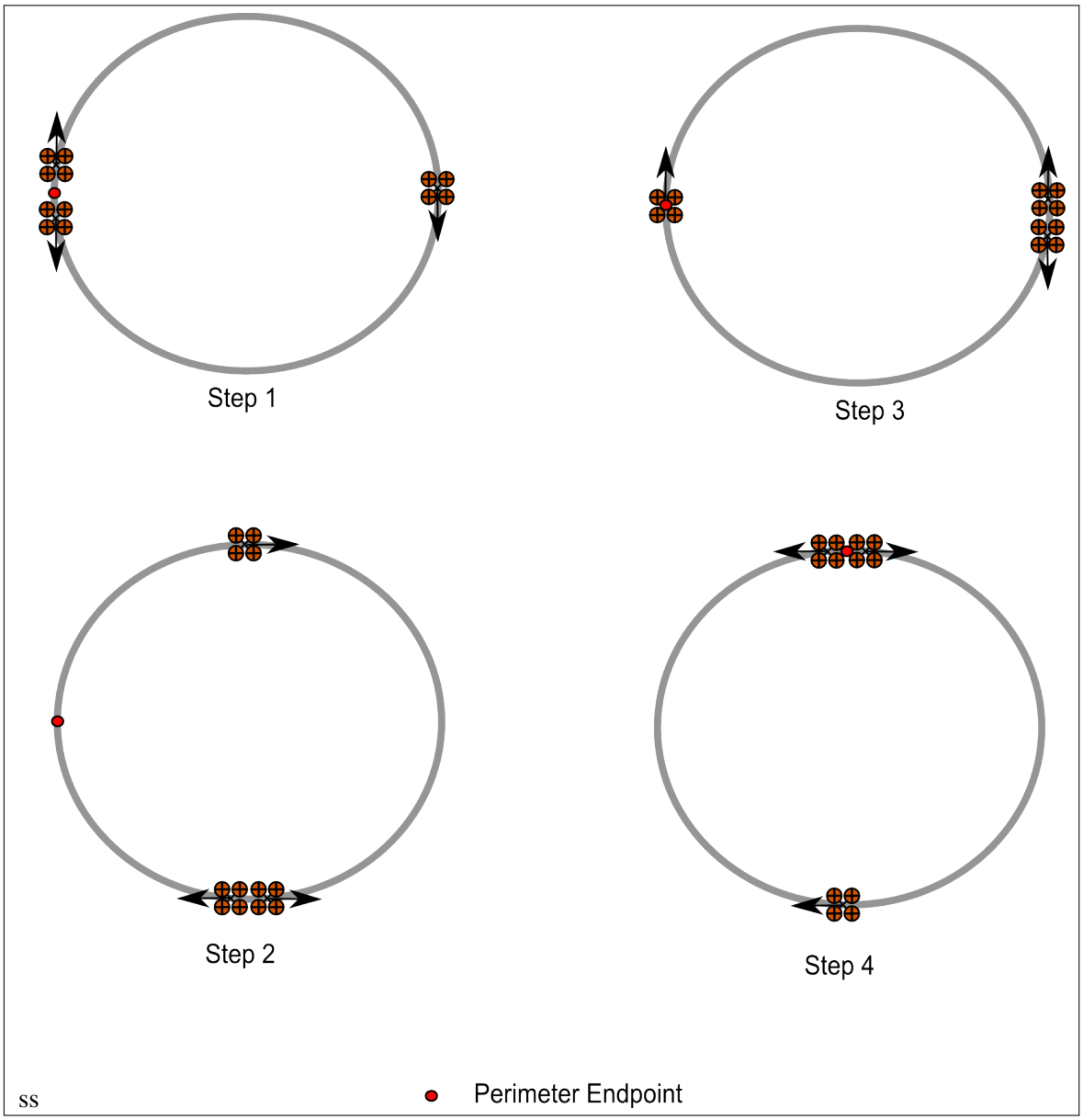

Figure 3.3: Problem with the perimeter endpoint for an uneven number of UAVs

At step one a new perimeter endpoint has just been set and the agents that met changed their directions. At step two one of the agents meets its neighbor, but the other does not, because both agents that

set the perimeter endpoint in the beginning share the same neighbor. In step three the second agent meets the neighbor and changes its direction again. When it meets the other endpoint agent again in step four they do not meet at the endpoint, but instead one quarter of the perimeter away from the starting position. They select this position as the new perimeter endpoint and the same procedure begins again.

This will always be the problem when an uneven number of UAVs tries to share a perimeter that is a circle in even parts, if the algorithm requires the UAVs to meet. The only way to make this algorithm viable is having a fixed perimeter endpoint, because the fire will move, however, this point may either lie inside of the fire or far away from the real fire perimeter after a while.

A similar problem is true for the first algorithm, it works well with an even number of UAVs, but the system breaks for an odd number.

To overcome this problem a virtual perimeter endpoint could be introduced by using a stationary UAV. This UAV could move with the fire, but would not patrol along the fire perimeter. Instead it would act as reference point for all other UAVs. The main drawback for this solution is the possibility that this UAV is lost. If this happens, however, the UAVs that were formerly next to the endpoint would meet and realize that the reference endpoint is lost. The would then decide on one of them to become the new virtual perimeter endpoint.

For both solutions only the goal of perimeter surveillance is considered, while in reality it might hold advantages for one or two UAVs to stop collecting information on the perimeter and instead follow a different, more important goal. Observing a person in a danger zone can be such a goal, or, additionally, even warning that person and providing it with important information. As a second example, it could be more important to supply medicine or water purification pills to endangered and trapped humans during a flood. A different goal could be observing a second fire that is moving in a direction different from the previously observed fire and should be monitored closely. Moreover, information about those different goals could be important enough for one drone to stop its monitoring task and directly return to the base station to relay the information as soon as possible.

In many situations, it is a advantageous if the drone decides whether another goal is more important or not. The reason for this is that the drone will not always be in communication range with a base station where the decision making could be handled. When a drone decides to terminate its monitoring tasks for a more important goal this must not endanger the overall observation goal. Therefore, a sensible coordination solution for multi-agent surveillance with communication restrictions should be decentralized and able to adapt to different numbers of agents.

3.3 Centralized Approach

Over the last years a quite a few fire surveillance algorithms have been proposed, all along the line of the two shortly presented in the last section. The important performance features for such an algorithm are: Information latency, Information refresh rate, Robustness (loss of agents) as well as Adaptiveness to changing perimeter.

The optimal low latency configuration, in regard to this thesis, is defined as the configuration where

the age of information, about a certain point of the perimeter, as soon as it arrives at the perimeter starting point, is exactly the time a UAV needs to fly along the perimeter to the closest end. To minimize that time only information of UAVs travelling into the direction of the closer end of the perimeter is considered fresh information. The lowest possible freshness of information, or latency ϕ for each point around the perimeter, with UAVs flying into opposing directions around it, is described by

$$\phi(x) = \frac{x}{v} \text{ for } 0 \leq x \leq P/2 \tag{3.1}$$

and

$$\phi(x) = \frac{P-x}{v} \text{ for } P/2 \leq x \leq P \tag{3.2}$$

where P is the complete perimeter length, x is the current point on the perimeter and v is the speed of a UAV.

The best possible latency from the middle of the perimeter is therefore

$$\phi\left(\frac{P}{2}\right) = \frac{P}{2v} \tag{3.3}$$

The overall latency profile of this setup can be computed by integrating over the freshness of information along all points of the perimeter, which gives:

$$\int_0^P \phi(x)\,dx = \frac{0.25P^2}{v} \tag{3.4}$$

In addition to these requirements, the optimal low latency configuration is required to have the optimal reachable refresh rate for information manageable with the available number of UAVs. To reach this optimal refresh rate agents should be equally spaced along the perimeter, resulting in a refresh rate of 2T/N, when the refresh rate is taken in regard to pairs of UAVs arriving back at the base station.

Most of the existing algorithms assume a base at the perimeter starting point which is used to recharge the UAVs and gather the collected information, or they ignore the problem of limited fuel or battery as well as the necessity to transmit the collected information to the crisis managers altogether. They aim to distribute agents along the perimeter evenly in a decentralized manner as well as minimizing the time it takes until such a low latency configuration is reached. Every agent along this perimeter has his section to patrol and flies back and forth along it once the perimeter is established. If an agent needs to recharge he can change roles with neighbouring agents and fly back to the base station. This solution offers little benefit over a centralized solution where all drones are started from the base station in pairs of two flying along the perimeter in opposing directions.

In order to establish a perimeter with this centralized algorithm, at first a minimum perimeter length is assumed. This estimation can be based on the number of agents, for every agent add a fixed length to the perimeter; satellite images, based on the age of the image and a fire expansion model add the assumed expansion to the perimeter length visible on the image; or simply on an educated guesses by

the crisis managers. Based on this estimation agents are started every

$$\frac{2P}{v} \times \frac{1}{\text{Number of UAVs}} \tag{3.5}$$

As soon as the first pair of UAVs arrives back they can fly around the perimeter again if they have sufficient energy. At this point the length of the perimeter is known to a certain degree, the perimeter length at the time that the first pair of UAVs passed it is known, but not yet how far it expanded during that time. When the second pair of UAVs arrives at the base station a first model for the expansion of the fire can be included by comparing the different perimeter lengths of the first and second pair in addition to their difference in arrival times. Depending on the result the agents can be started directly again, that is if it fits with the computed low latency configuration. For the computation of this configuration it is assumed, that no UAVs have been lost. If they have to loiter before their next start in order to reach the optimal configuration they can be refueled at the base station, batteries can be exchanged and damaged parts replaced. When the next agents arrive the model is updated accordingly.

This setup offers at least the same latency profile as other proposed completely decentralized algorithms, because it offers the optimal profile. In cases where the UAVs loiter or have to turn around in other algorithms it offers even better results. The refresh rate of this algorithm is always 2T/N, which equals the best achievable refresh rate with the decentralized solutions.

3.3.1 Perimeter Knowledge

One advantage of the completely decentralized solution could be that the same UAV always observes the same part of the perimeter and that they therefore have superior knowledge about the territory. This drawback can be addressed by simply transmitting this information whenever two agents traveling into different directions meet. Since the decentralized solution transfers all important information about the environment from drone to drone to transmit it to the base station it is reasonable to assume that this transfer is possible without losing time in comparison to the decentralized algorithm even if the transfer takes more time than available during a simple fly-by-transfer. The agents don't have to transmit all collected information but only the assumptions the agents made and a few images to compare them to new sensor information. However, all the observed values for the inference engine may have to be exchanged between both drones, depending on the implementation. For the infer.net implementation chosen in 5.3 the computed distributions can be used as priors for the new observations, old observations do not have to be transmitted.

3.3.2 Lost UAV

A UAV in this algorithm can be assumed lost when only one from a pair arrives during a small interval. If there are spare agents another one can be started together with the one from the pair if the second agent arrives delayed he can be recharged and put into the reserve. If there are no spare agents and there is now an uneven number of agents assumed to be in the system, the agents can be started alternating into different directions with the offset of 2T/N to still get the lowest possible refresh time. The assumption that an agent might be lost or on a special mission in that case is dependent on T, the time

agents are assumed to need for one round along the perimeter based on previous data. An agent is sure to be lost when either his successor arrives at the base station or one of the agents flying in the different direction reports him as missing. If an agent is lost after he finished half of the perimeter he will be reported missing by his successor at T/2 + 2T/N, if he is lost on the first half of the perimeter and the UAV pairs are started synchronous at the starting point he will be reported missing after T/2. If they are started asynchronous at the latest at T/2 + T/N (Time to fly around half of the perimeter + time to reach the second half of the perimeter in worst case).

3.4 Remote Surveillance

In section 3.3 one main assumption has been made that will not be true for every use-case, namely the regular communication with the base station. Whenever a constant stream of information about the perimeter is required at the base station this assumption is true. However, if the UAVs have the ability to estimate the expansion of a fire on their own it would be possible, and in most cases recommendable, to only send back information when an important change in fire behaviour occurred. Using this UAV behaviour would prevent UAVs from constantly wasting time and battery power to fly back to the base station, only to report that the fire behaved as expected.

If there is no regular communication with the base station the centralized approach does not work anymore, however, a similar but decentralized approach can be implemented. The main idea behind this approach is that each agent tries to keep the same distance between himself and his predecessor and himself and his follower. To achieve this state all UAVs have to adjust their speed every now and then. Therefore they can not fly with their maximum speed all the time and have a slightly worse latency profile then the centralized solution. It is, however, still better than any solution requiring loitering.

To reach an equal distribution of agents along the perimeter every agent has to know if his distance to the agent flying in front of him is bigger or smaller than the average distance between UAVs. Whenever a UAV encounters an agent flying into the opposite direction it records the time difference between him and the agent in front of him. He then calculates the current average of distances. This can be done by having a unique identifier for each UAV and just adding all the most recent time differences and dividing them by the number of UAVs.

To handle lost UAVs this data is collected in a list. Whenever a UAV should have been encountered but another UAV is encountered instead it is deleted from the list. Whenever a new UAV is encountered it is entered into the list instead. If the number of UAVs flying in one direction is consistently bigger than the number flying into the other direction one UAV has to change its direction. To reach this goal each UAV should have a unique identifier, which can be set for example before the mission is started. This identifier should be chosen as an integer. This identifier is used to determine which UAV should change its direction. The UAV with the lowest identifier in the list of the UAVs in the direction with more UAVs is chosen to change its direction and thereby adjust difference.

The computed value for the average distance between UAVs is then transmitted to the encountered UAV together with its distance from the UAV directly in front of it. Depending on this information the UAV changes its speed, it accelerates if the distance to the UAV in front of him is greater than the

average and it slows down if it is smaller. A potential formula for controlling the behaviour can be seen in equation 3.6.

$$v(dist, distA) = v_{normal} + \gamma(dist - distA) \times v_{normal} \tag{3.6}$$

dist is the distance between a UAV and the UAV in front of it, while *distA* is the average distance. γ is a function to relate the difference in distances to a percent value in a predefined range. This means the maximum speed gain or loss is capped.

An example for such a function with a cap at $\pm$ 10% could be the one in equation 3.7.

$$\gamma(x) = \begin{cases} -0.1 & \text{for } x < -\frac{distA}{10} \\ \frac{x}{distA} & \text{for } -\frac{distA}{10} \leq x \leq \frac{distA}{10} \\ 0.1 & \text{for } x > \frac{distA}{10} \end{cases} \tag{3.7}$$

3.5 Conclusion

The centralized approach in section 3.3 , as well as the decentralized remote surveillance version of it in section 3.4, offer some advantages over the ones described in section 3.2. The main points are a faster convergence time for the centralized approach as well as no need to loiter at any point during the surveillance. There are, however, considerable drawbacks to these approaches. In both solutions the UAVs fly around the whole perimeter and ,therefore, have only old information about every part of the perimeter they are flying over. The UAVs in the approaches in section 3.2 only fly along one section of the perimeter, which gives them fresher information about this area. This will be required to make predictions about the expansion of the fire in chapter 4. The lack in perimeter knowledge can be overcome if each UAV transmits its knowledge about the perimeter to UAVs flying into a different direction than they themselves. The transmission of information requires time as well as energy, however, and might force UAVs to stop during their surveillance just to transmit all required information. If there is an uneven number of UAVs in the centralized approach it results in different update rates for the two directions of the perimeter and, therefore, in different update rates for the two sides of the perimeter to the left and the right of the base station. An uneven number in the approach described in section 3.2 results in a constantly changing position for the perimeter endpoint and would therefore never reach a stable state. It is not possible to use this algorithm for the surveillance of a circle perimeter if no fixed perimeter endpoint is available.This problem can be overcome by introducing a stationary UAV, however. The right approach has to be selected based on the importance of these factors, this can for example be influenced by the hardware used, the amount of data that can be stored and transmitted, or the battery power.

Currently the proven algorithms from the research section 3.2 are the best choice, if they are combined with the stationary UAV as virtual perimeter endpoint. Additional research is required in order to prove that the algorithm proposed in 3.4 is viable.

4 Fire Model

4.1 Prerequisites

4.1.1 Splines

The idea of splines origins from shipbuilding, where splines, long and thin pieces of wood, have been used to describe the curvature of the hull of a ship. These flexible splines were fixed at a few certain points that are required to lie on the ships hull and then formed the curvature of the hull in a way which minimizes the inner tension of the spline. The same idea is nowadays used to interpolate and approximate curves or areas. When connecting a number of points the first idea would be to either connect them directly or to find a polynomial that fits all points. If the first approach is used to describe curves it is either not very accurate or it requires a high number of description points. The second approach is prone to overfitting. If a high number of points is connected using one polynomial, this polynomial becomes very complex and oscillates heavily in between these description points. Because of this the polynomial doesn't describe additional points on the approximated curve very well. Instead a different polynomial function is used to describe the connection between pairs of two control points. To create a smooth curve between all points these functions have to fulfill certain properties. The most important property is that two neighboring curves have to be continuously differentiable in the point where they meet. For a spline of degree N the polynomials have to be $N-1$-times continuously differentiable. [ML98] Figure 4.1 shows a linear connection between 5 points, figure 4.2 shows the same connections with cubic splines.

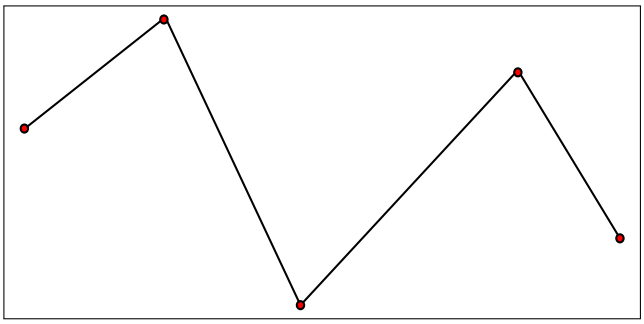

Figure 4.1: A linear connection (which could be a linear spline) between points

4.1.2 Normal Distribution

The normal distribution is one of the most used and well-known distributions. This is mainly due to the central limit theorem, which states that the sum of a large number of independent random variables, all created by the same distribution, is approximately normally distributed. The original distribution

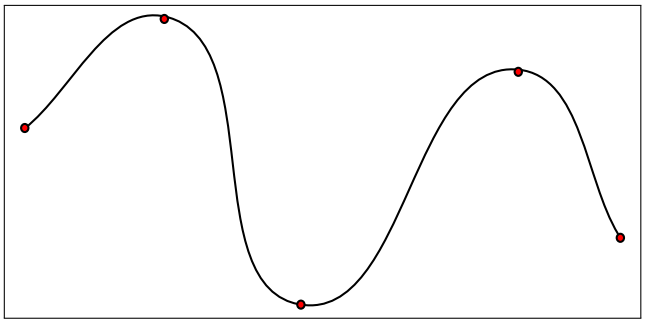

Figure 4.2: A spline of higher order connecting the same points

of the random variables does not matter. The "fuzzy" central limit theorem states that data which are influenced by many small and unrelated factors are approximately normally distributed [Weia].

Because of this property normal distributions can be used to describe for example observational error or random errors in the production of materials or parts.

The Normal distribution is described mathematically as shown in equation 4.1.

$P(x)$ is the probability density for a given value of x. The probability density describes how high the chance that the value x will occur is. If the probability density of $x = 4$ is twice as high as the probability density of $x = 3$ this means that $x = 4$ is twice as likely to be measured.

There are two parameters influencing the shape of the distribution:

- μ is the mean value of x.
- σ is the standard deviation, which describes how much the random variable varies from the mean.

The integral of the probability density distribution is 1.0 which is ensured by the factor $\frac{1}{\sigma\sqrt{2\pi}}$. With this normalization it is possible to use the integral function to directly make statements about the probability of a measurement of the random variable being in a certain range. For example if the integral from -1 to 1 is 0.6, the probability of the measurement being in this range is 60%

Figure 4.3 shows the probability density function of normal distributions with different values for μ and σ.

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2 / 2\sigma^2} \text{[Weid]} \tag{4.1}$$

4.1.3 Gamma Distribution

The Gamma distribution is related to the beta distribution, but while the beta distribution is defined only between $[0,1]$ the gamma distribution has a range of $[0,\infty]$. It is usually used in processes for which the waiting time between Poisson distributed events are relevant.

In probabilistic programming it is mainly used to define the precision parameter of a Gaussian distribution. The probability density function for a Gamma distribution is shown in 4.2.

The meaning of $P(x)$ is explained in 4.1.2 .

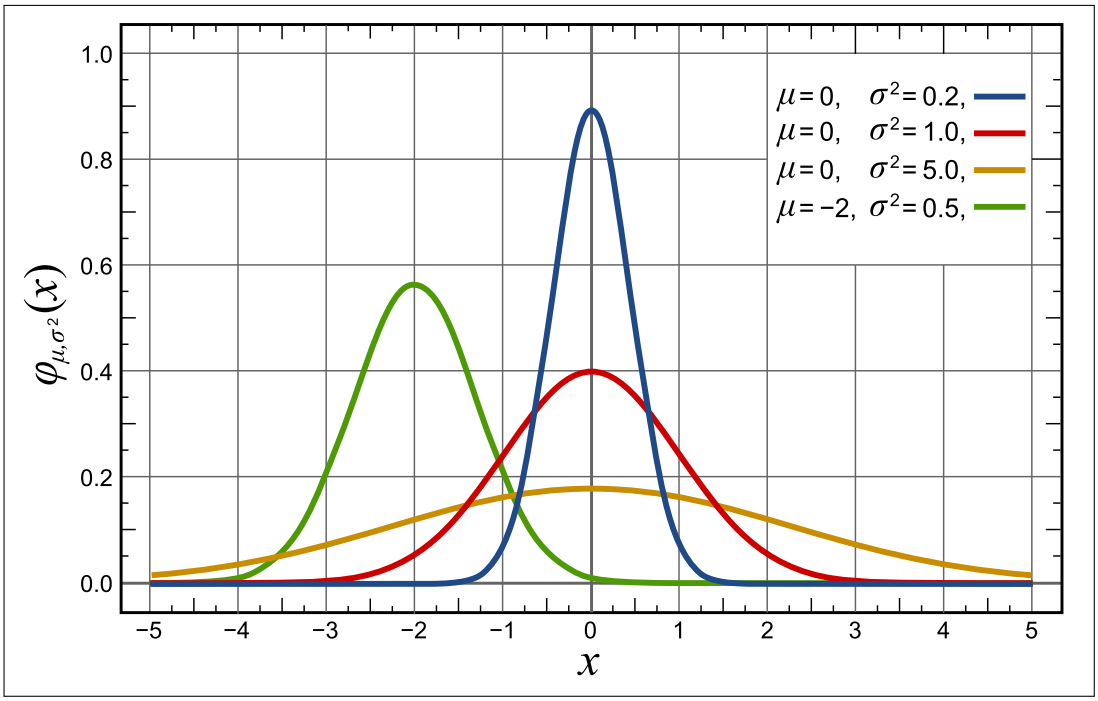

Figure 4.3: Probability density function of typical normal distributions [Comb]

Γ is a Gamma function, it is defined as

$$\Gamma(n) = (n-1)!$$

for integer values of n and

$$\Gamma(z) = \int_0^\infty e^{-t} t^{z-1} dt$$

for complex values of z.

The definition can be found for example in [Weic].

The shape and scale of the Gamma distribution are influenced by two parameters

- k, the shape parameter
- θ, the scale parameter

The scale parameter, in probability theory and statistics, is a parameter that describes how spread out the distribution is.

A shape parameter is any parameter influencing a probability distribution that is neither a scale nor a location parameter or any combination of those.

$$P(x) = x^{k-1} \frac{e^{-\frac{x}{\theta}}}{\theta^k \Gamma(k)} \text{ for } x \geq 0 \text{[Weib]} \tag{4.2}$$

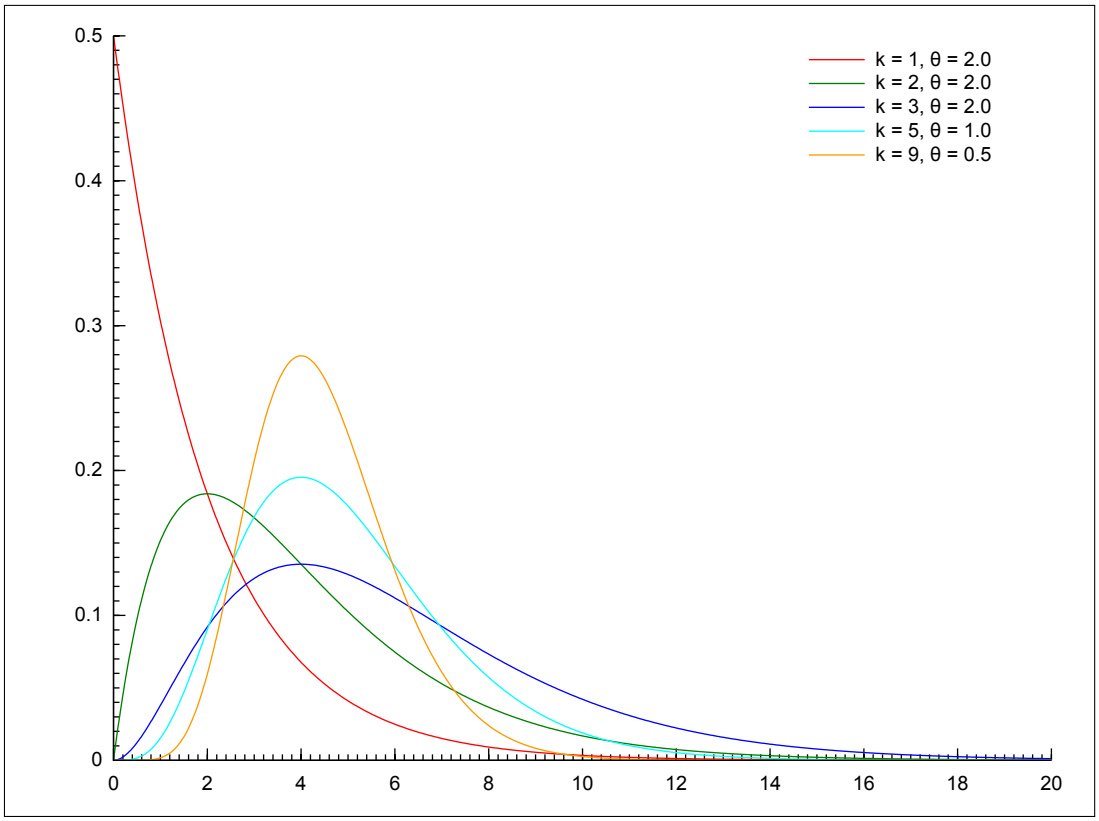

Figure 4.4: Probability density function of typical gamma distributions [Coma]

4.1.4 Hidden Markov Model

Although I will not directly use Hidden Markov Models in this chapter I feel it is still important to introduce them, as well as their drawbacks, to make clear why DBNs have been chosen instead. The model proposed in 4.3.5 is the Dynamic Bayesian Network representation of a Coupled Hidden Markov Model.

A HMM is a stochastic finite automaton, where each state generates (emits) an observation. Usually X_t denotes a hidden state at time t while Y_t denotes the matching observation. X_t could for example be the weather, while Y_t is the observation of somebody bringing an umbrella or not. The actual weather in this case is hidden, we can only make assumptions based on the umbrella usage.

The HMM has a three defining parameters:

- Initial state distribution $\pi(i) = P(X_0 = i)$
- Transition model $A(i, j) = P(X_t = j | X_{t-1} = i)$
- Observation model $B(i, k) = P(Y_t = k | X_t = i)$

The HMM transition model is usually described using a stochastic matrix. Stochastic means that each row sums up to one, which is required because the combined transition probabilities from one state can not exceed 100%. This matrix can also be represented as a state transition diagram, which makes it easier for humans to quickly get an idea about the possible state transitions. Figure 4.5 shows such

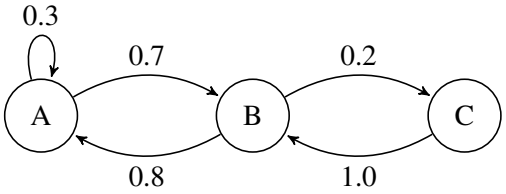

Figure 4.5: HMM state transition diagram

a diagram with the possible states A, B, C. The nodes represent states, while edges represent transition probabilities. The self-loop at state A means that there is a 30% probability of state A transitioning into state A again. State B has a 80% probability to transition into state A and a 20% probability to transition into state C.

The main problem with HMMs is their rapid growth in complexity. One example for this would be using a HMM model to track the state, meaning position, of N objects in an image sequence. Each object can be in one of k possible states. This means $X_t = (X_t^1, \ldots, X_t^N)$ can have $K = k^N$ values, because the Cartesian product of the state-spaces of each individual object has to be formed. Because of this the transition model $P(X_t|X_{t-1})$ needs $O(k^{2N})$ parameters to specify , which means it takes much more data to learn the model. In addition, inference takes exponential time, e.g., forwards-backwards takes $O(Tk^{2N})$ as well as $O(TK^N)$ in space, where T is the length of the sequence [Mur02]. Using DBNs instead helps to mitigate this problem.

4.1.5 Bayes Net

Bayes Nets or Bayesian Networks are graphical representations of the relationship between random variables. For a set of discrete random variables $[X_1, \ldots, X_N]$, with a finite number of possible values, a Bayesian Network describes the relationships between them using a directed acyclic graph. The nodes of the graph represent the different random variables, while the edges describe how they influence each other. Each node has a conditional probability distribution which describes what the probability of each possible value for that node given the values of its parent nodes in the graph is.

$$P(X_i|Pa(X_i))$$

The conditional probability distribution together with the graph encode the joint distribution $P(X_1, \ldots, X_N)$. A unique joint probability distribution over X from this graph can be factorized as

$$P(X_1, \ldots, X_N) = \prod_i P(X_i|Pa(X_i))$$

[Pea88]

4.1.6 Dynamic Bayesian Networks

While HMMs represent the state of the world using a single discrete random variable, $X_t \in 1, \ldots, K$ DBNs represent the state using a set of random variables, $X_t^{(1)}, \ldots, X_t^{(N)}$. With this distributed repre-

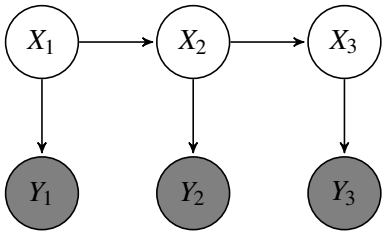

Figure 4.6: HMM as DBN

sentation $P(X_t|X_{t-1})$ can be represented in a compact way using a parameterized graph. For this each DBN is defined to be a pair of $(B_1, B_{\rightarrow})$. B_1 is the prior $P(Z_1)$ and $B_{\rightarrow}$ is a two-slice temporal Bayes net defining $P(Z_t|Z_{t-1})|$ as shown in equation 4.3.

$$P(Z_t|Z_{t-1}) = \prod_{i=1}^{N} P(Z_t^i|Pa(Z_t^i)) \tag{4.3}$$

Z_t^i are the random variables, which are usually partitioned into

- Input variables U_t
- Hidden variables X_t
- Output variables U_t

i and t express that it is the i'th node at time t. $Pa(Z_t^i)$ are the parents of Z_t^i in the Bayes net graph. None of the nodes in the first slice of the two-slice temporal Bayes net have any parameters. Every node in the second slice has an associated conditional probability distribution which defines $P(Z_t^i|Pa(Z_t^1))$ for all $t > 1$[Mur02]. Because Dynamic Bayesian Networks are basically extended Bayes nets(The extension allows to model dynamic processes) and Bayes nets are a directed acyclic graphical model, DBNs can be seen as a graphical model. Directed edges in this model represent causation, either inside one timeslice or between different timeslices. For connections between timeslices only "older" timeslices can influence "younger" ones and not the other way around.

A HMM can be represented using a DBN, an example for this is shown in figure 4.6. As usual observed nodes will be displayed shaded, while hidden nodes are clear. The graph represents two conditional independence assumptions:

- $X_{t+1} \perp X_{t-1}|X_t$, the Markov property
- $Y_t \perp Y_{t'}|X_t$ for $t' \neq t$

4.2 Fire Perimeter Description

A useful fire description should not only relay all important information about the size and the expansion rate of the fire, but also do this with as few description points as possible. This is due to two reasons. First all points have to be stored and for a big fire the amount of points that have to be stored

and transmitted can get infeasible, but more importantly, all points are used to estimate the expansion of the fire and therefore a huge number of description points increases the required computational power as well, without creating any useful additional information. The main points required to describe any perimeter are the corner points. Corners can be assumed everywhere where the turn the UAV has to fly is sharper than a preset threshold. When the UAV turns away from the fire during the turn it is a concave corner, when it turns into the direction of the fire the corner is convex. This can be seen in figure 4.7.

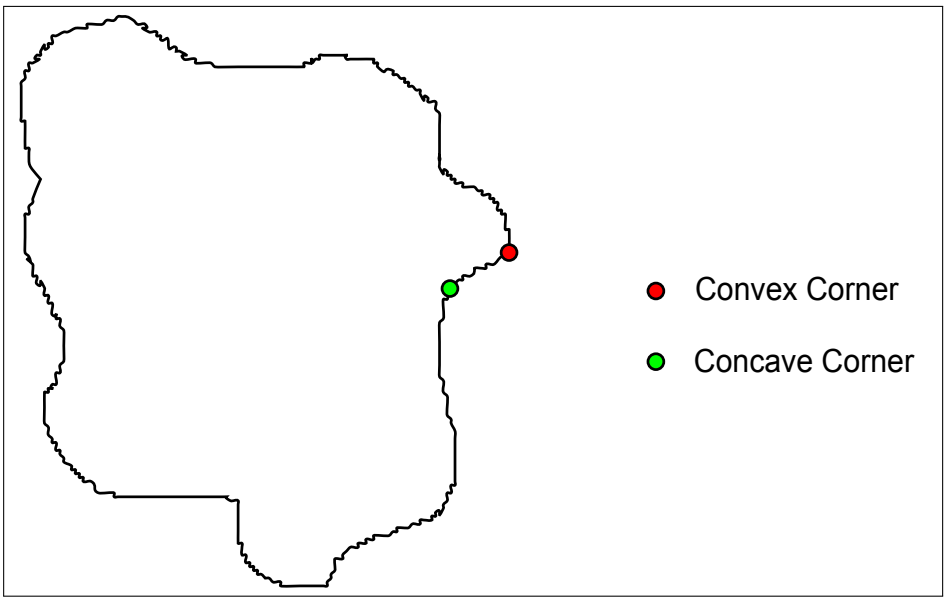

Figure 4.7: Convex and concave corners

- Points at convex corners are enough to describe an upper bound for the perimeter size
- Points at concave corners describe areas where the expansion is lower than the upper bound

This criterion alone, however, is not enough to define a single point, whenever the UAV takes a longer turn, every point during this turn would have to be recorded. Instead of doing this the point when the sharpness of the turn first surpassed the threshold is recorded as well as the point at which the sharpness falls below the threshold again. If there is either another rise in the turns sharpness another point is added. The same is done if a long distance has been travelled without recording a point, which can happen if a turn is very long. This is done to ensure that the distance between points does not become too big, which would lower the probability of correlation in the expansion parameters between two neighboring points. This will be discussed in more detail in subsection 4.3.4. Figure 4.8 shows candidates for description points for an example fire perimeter.

If there are many sharp turns in a small area the amount of points required to describe the fire using this method grows bigger than actually necessary. If this happens a few points have to be deleted. This can either be done using a very simple linear algorithm or using one involving splines.

The simple algorithm always compares three points, which are next to each other, and connects the two on the ends of that perimeters section with a straight line. If the third point lies on the side of the

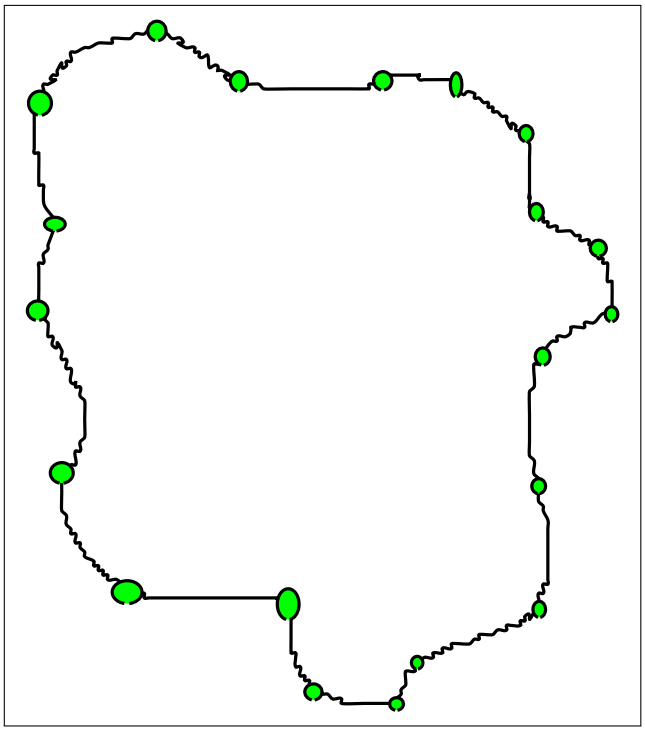

Figure 4.8: Candidate points for describing the fire

line where the fire is, it can be removed. This procedure can be continued until there are no more points that can be removed. Using this method the points describe a fire that is bigger than the actual fire, but still cover the whole area.

The second algorithm uses splines to connect all points describing the fire model. The connection of the points using splines is already required to get a fitting description of the fire with only a few data points. If the points would be connected linearly, some parts of the fire would not be inside the enveloped area for every convex corner. Using splines to connect all points allows to create a better fire description with the same amount of points. Because the UAV is flying around the perimeter while gathering description points, it can not only record the point, but also the steepness of the curve it has been flying while saving the point. Because this information is basically the derivative at the same point it can be used to find very accurate splines for the description.

The describing splines can be used to decide if all description points along the perimeter are necessary. Whenever the description using splines differs from the real position of the fire, the UAV can either change the parameters of the spline or add additional points as required. If points should be deleted the UAV can check for every point how much the splines change if a certain point is deleted. If the change is within specified parameters the point can be deleted. The same method can be used to move description points along the parameter, one point can be moved to get a more even distribution of description points. It can only be moved along the spline and only if it is possible without changing the spline too much. Subsection 4.2.2 gives additional detail on how to decide whether a spline is fitting or not. This includes the decision process on whether a point can be removed, which results in two splines being replaced by one.

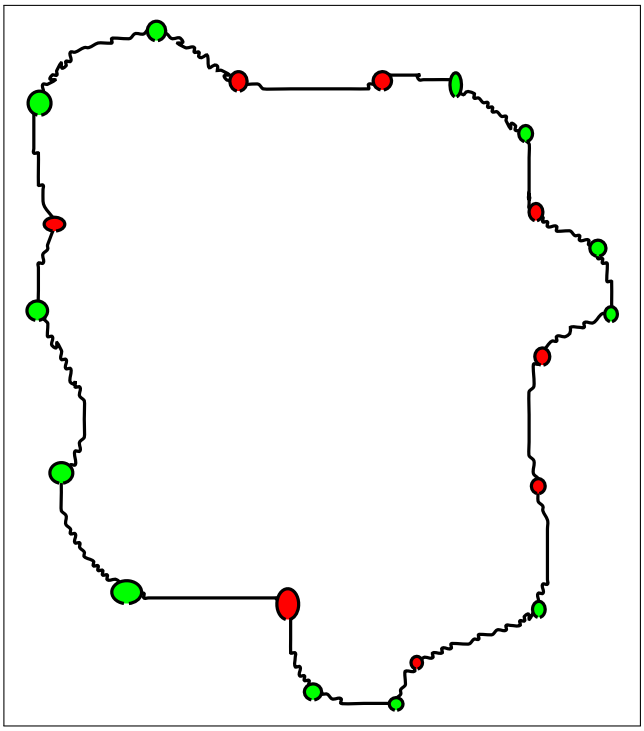

Figure 4.9: Points that can be removed if necessary (in red)

4.2.1 Boundary estimation - research review

[FB09] describes an algorithm that enables a robotic network to estimate the boundary of a perimeter using polygonal interpolation. Their solution allows for a trade-off between between equidistant distribution of points along the perimeter and the higher amount of points required to describe areas of the perimeter with a higher curvature. This is done by first defining the distance between points in a way that includes the arc-length between them as well as the curvature in between using equation 4.4.

$$D_\lambda(q_i, q_{i+1}) = \lambda D_{curvature}(q_i, q_{i+1}) + (1-\lambda) D_{arc}(q_i, q_{i+1}) \tag{4.4}$$

$D_{curvature}$ is a function describing the curvature and D_{arc} the arc-length between the two points.

λ is used to define whether the curvature or the arc-length is more important and should be $\lambda \in (0,1)$. The closer λ is to 1 the higher is the importance of the curvature. Their solution for the perimeter estimation consists of an update law for a single robot and a cooperative update law.

For single robot update law the agent collects information about its tangent and curvature along the path. The old perimeter estimation points are projected onto its new path whenever enough information for this projection is available and in the next step the position of the projected point is optimized along this path.

The projection step takes place whenever the agent crosses the line that is the perpendicular to the agents current movement vector and goes through the point about to be projected. After this the optimization step for the preceding point is executed. For this the interpolation point is moved along the path of the agents trajectory, in order to balance its distance from its neighbors. This is done by a

function that places q_{i-1}^{+}, the in a way that equation 4.5 is fulfilled.

$$D_\lambda(q_{i-1}^{+}, q_i) = \frac{1}{4} D_\lambda(q_{i-2}^{+}, q_{i-1}) + \frac{3}{4} D_\lambda(q_{i-1}^{+}, q_i) \tag{4.5}$$

Where q_i is the newest point, q_{i-1} its predecessor and q_{i-1}^{+} the optimized position of q_{i-1}.

For cooperative estimate update law each agent has its own list of the perimeter representation and updates its list separately. Every time an agent updates its interpolation points, either by using the projection or the optimization step it transmits these points to its neighbors. These agents update their interpolation points accordingly. The agents only update their own values with transmitted interpolation points from other UAVs if it is a point they are not currently processing. This means there have to be at least two points interpolation points between neighboring agents. It is assumed that every agent can communicate with its direct neighbors at any time.

4.2.2 Spline comparison

To decide whether one spline is a better approximation for curve than another there needs to be an unambiguous decision criterion. The criterion for this use-case is the distance between the two curves. The distance can be defined in different ways, some of which are not viable. It does, for example not make sense to compare the minimum distance between the curves, because it could be only one point along these curves that fits very well. Instead the distance can be defined in two viable ways.

The first one is the maximum distance between the the spline and the curve to be approximated. If the maximum distance between these two curves is minimized it can be guaranteed that no point of the real fire lies further than this distance away from the description. Therefore extreme points, where the two curves differ greatly, but only over a very small area, in the fire perimeter are handled correctly. However, since interpolation points are created on such extreme points during the boundary estimation process, such outliers are not possible.

Because of this the second method can be used, where the distance between the spline and the perimeter curve is not compared for every point on the curves. Instead the area between the two curves is used as a measure for the overall distance. This can be done by first finding the points where the two curves intersect and then computing the integral between those two curves in the way described in equation 4.6.

$$Dist = \sum_{i=1}^{n} \int_{p_i}^{p_{i+1}} \begin{pmatrix} upper \\ curve \end{pmatrix} - \begin{pmatrix} lower \\ curve \end{pmatrix} dx \tag{4.6}$$

In this equation n is the number of intersection points plus two, for the start and endpoint of the section of the curve about to be compared. p_1 is the starting point and p_n the endpoint, all points $p_2 \dots p_{n-1}$ are the points where the two curves intersect. The upper curve is the curve between two of those points, which always has the higher y value, the lower curve the one with the lower y value. There might not always be an upper or lower curve, but a left and right curve instead depending on the

alignment of the coordinate system. In this case the equation changes to the one in equation 4.7.

$$Dist = \sum_{i=1}^{n} \int_{p_i}^{p_{i+1}} \begin{pmatrix} right \\ curve \end{pmatrix} - \begin{pmatrix} left \\ curve \end{pmatrix} dy \tag{4.7}$$

In a real scenario the function describing the real fire perimeter will not be available, a discretization of the describing path is necessary in order to store it on the UAV. Therefore the UAV records temporary points along the perimeter in regular intervals, the distance between these points D_{temp} is a lot smaller than the distance between interpolation points $D_{interpol}$: $D_{temp} \ll D_{interpol}$. These points are only recorded to allow a comparison between the real fire perimeter and the interpolating spline. The points can be deleted as soon as a fitting spline has been found, thereby saving memory.

With these points instead of a function describing the fire, the distance between the spline and the fire is computed by summing up the minimum distance between every temporary point and the describing spline as shown in equation 4.8.

$$Dist = \sum_{i=1}^{n} \sqrt{\left(x_{temp_i} - x_{spline_i}\right)^2 - \left(y_{temp_i} - y_{spline_i}\right)^2} \tag{4.8}$$

x_{temp_i} and y_{temp_i} are the x and y coordinates of the temporary points, x_{spline_i} and y_{spline_i} the points on the spline which are closest to this temporary point. Since those are not known yet the first derivative of

$$\sqrt{\left(x_{temp_i} - x_{spline_i}\right)^2 - \left(y_{temp_i} - y_{spline_i}\right)^2}$$

has to be set equal to 0 and this equation solved.

When trying to find the spline that describes the section between two interpolation points accurate enough the UAV can start with a simple linear spline. If the distance between the spline and the temporary points is low enough to fulfill mission requirements, which is the case if the perimeter between those two points is similar to a straight line, the approximation process can stop. Otherwise the next spline, a quadratic spline, is computed and the evaluation is repeated. This process can be repeated with a cubic spline. If this is not enough to reach mission requirements another interpolation point should be inserted at the point between the two interpolation points that has the highest curvature.

Using the spline comparison to decide whether an interpolation point should or can be deleted involves a similar process. To start the comparison a set of three points A, B, C is required, B is the point that should be removed, A and C are its neighbours to the left and right. At first the distance between the splines connecting A and B as well as B and C and the temporary fire perimeter is computed. Then a new linear spline connecting A and C is created and the distance between this spline and the temporary fire perimeter is compared to the one of the two splines. If the distance is smaller for the single spline it is always a good choice to delete point B, if it is not smaller, but still fulfills mission requirements it can be deleted. Otherwise the process can be repeated for quadratic and cubic splines.

4.2.3 Conclusion

The algorithm for single robot updates proposed in subsection 4.2.1 can be applied in the same way for the fire perimeter description proposed in the beginning of section 4.2. Splines are used to connect the points found this way and to delete points if necessary.

The cooperative estimate law, however, needs to be adapted. [FB09] makes two assumptions that are not valid for the model proposed in this thesis.

- Agents are flying around the whole perimeter and only into one direction
- Agents can communicate with their neighbors constantly.

Depending on the surveillance algorithm that is chosen from chapter 3 changes to the algorithm are necessary.

If the algorithm in 3.3 is chosen the UAVs only patrol one section of the perimeter, if the algorithm in section 3.4 is chosen agents are flying in different directions around the perimeter, but always around the whole perimeter. In both cases they are only able to communicate with each other whenever they meet and they only meet agents flying into different directions than themselves. In order for the cooperative estimate law to work the UAVs have to transmit their interpolation points to each UAV they meet, this UAV can now use these points to update its own interpolation points in front of it. How many points in the UAVs flight direction are updated depends on how accurate and fresh each UAVs map of the whole perimeter should be. The best map can be created it if all points, up to a distance of 50% of the complete perimeter length, in flight direction, are transmitted and updated. This allows each UAV to have the latency profile shown in equation 4.9 every time it meets another agent.

$$\int_0^P \phi(x)\,dx = \frac{0.25P^2}{v} \tag{4.9}$$

.

The reason for this is that the UAV in this situation virtually has the position of a base station described in section 3.3.

The number of points to be transmitted can be reduced to reduce the time of transmission as well as the battery consumption. This reduces the accuracy of the interpolation points each agent has and thereby the overall latency profile. If, however, each UAV transmits only the points it updated since it last met another UAV, every UAV is able to make good predictions for the the sector of the perimeter it last flew over, as well as the sector it will travel along next. Each UAV can make prediction for these two sections with the lowest achievable latency profile and the best possible accuracy. The distributed information all UAVs flying into one direction have is, therefore, enough to make accurate predictions of the fire expansion around the whole perimeter. This solution should be enough for most use cases, because every agent is able to act on these predictions.

4.3 Fire Expansion Estimation

4.3.1 Why estimate fire expansion

Estimating the expansion of a fire directly on the UAVs offers a number of advantages. With the expansion of the fire known in advance the UAVs can estimate how many of them will be required to observe the fire, with some requirements for parameters, for example maximum latency or maximum perimeter sector size, that have to be met. If the number of UAVs at the fire is too low, or will be too low in the future, based on the predictions, the UAVs can inform the base station that more agents are required.

Areas that are not expanding fast or not expanding at all anymore can be observed by less UAVs than fast expending areas. This results in either less UAVs being required to observe a fire or in a better surveillance of the fast expanding parts. An image of this can be seen in figure 4.10. An uneven distribution along the perimeter can be integrated in the perimeter surveillance algorithm by allowing a UAV coming from a “red” zone, which is a fast expanding zone, to a “green” zone, which is a slowly expanding zone, to turn around without meeting its neighbor UAV. This can be done only once, the next time the UAV arrives at the same perimeter point it has to fly until it meets its neighbor, to ensure the expansion values are still the same and its neighbor UAV is still in place. To integrate this into the surveillance algorithm “red” areas can be counted as double of the area covered for the equal spacing part of the surveillance algorithm.

Based on the expansion speed, warnings can be send to the home base. When a fire is expanding faster than a preset parameter, or expanding fast into a preset direction, for example into direction of a town, one UAV can fly back and inform the base station. Based on this kind of information from different fire hot-spots, the crisis manager can make decisions about which fire to extinguish first.

4.3.2 Simple Model

The simplest model to predict fire expansion considers only the history of one fire perimeter point to make predictions for exactly the same point. Because the UAVs measure the expansion whenever they fly by the point, all expansion values are based on time steps defined by the speed of the UAV and the size of the perimeter as well as the distance of the point from the perimeter ends. At first the expansion between the previous measurements is computed by simply subtracting the predecessor of each stored position vector from itself.

$$\begin{pmatrix} ExpX \\ ExpY \end{pmatrix} = \begin{pmatrix} PosX_t \\ PosY_t \end{pmatrix} - \begin{pmatrix} PosX_{t-1} \\ PosY_{t-1} \end{pmatrix}$$

After this the expansion value is normalized by dividing through the time between the measurements at t and $t-1$. It is assumed, that the normalized expansion speed of the fire is normally distributed over a fixed number of measurements in the past. This could for example mean that only the last five measurements are considered for the prediction. To make a prediction about the expansion speed in the future, the Gaussian normal distribution fitting the fixed set of of expansion values best is computed. The mean value of this distribution is the expansion speed prediction for this perimeter point.

The same method is used for every point on the fire perimeter and the prediction is always updated by

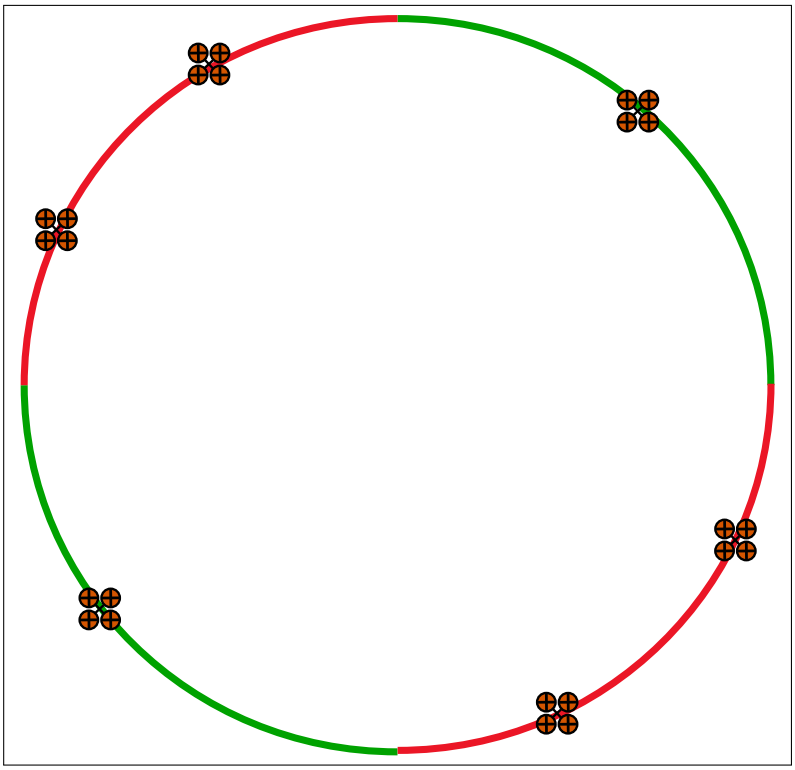

Figure 4.10: Fire perimeters with different expansion speeds

deleting expansion points that are outside the look-back scope and adding new measurements whenever they are taken. A image of this model can be seen in figure 4.11.

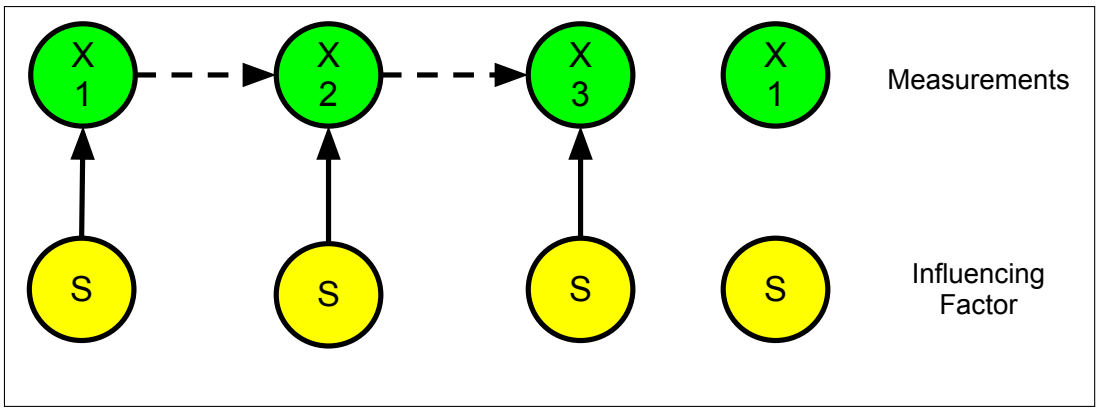

Figure 4.11: Simple model

4.3.3 Simple Model with Attenuation

In the simple model the order in which the different values have been observed does not matter. This means a new measurement influences the distribution in the same way as an old one does. Since factors might change over time and change into the same direction this should receive attention in the model. Wind for example could change its direction or increase in intensity over time. The ground in the forest will not change suddenly from dry to wet or from dense to sparse in most cases but rather change

slowly.

To include this into the model a second influencing factor can be introduced. All measurements share these influencing factors but the strength of their influence changes over time. For the oldest measurement the influence of factor S_{old} is 100% and for the newest measurement the influence of factor S_{new} is 100%. In between the influence slowly shifts from S_{old} to S_{new}. Using this model enables the inference engine to recognize the direction of change in the fire expansion parameters. If only the influencing factor S_{new} is used for predictions and there has been a change in the expansion speed S_{new} will allow for better predictions, if there has been no change in the expansion S_{new} will be the same as S_{old} and therefore the predictions will not differ from the ones made using only the simple model.

Figure4.12 shows an image of this change in the model.

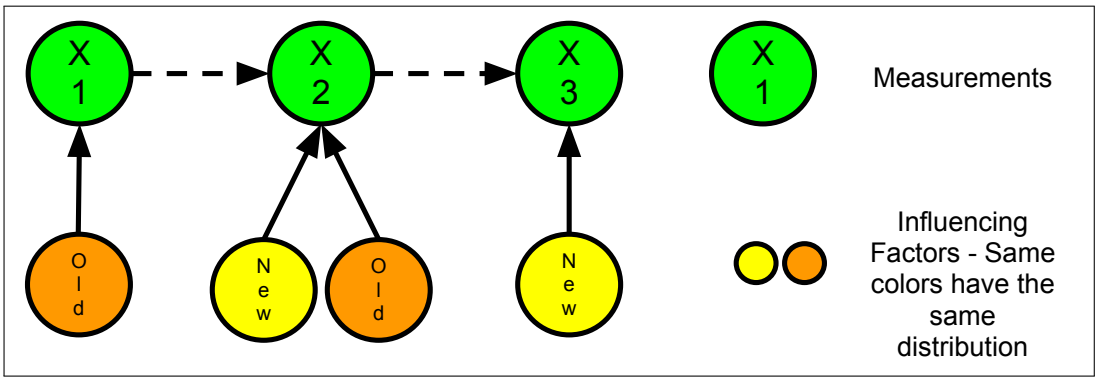

Figure 4.12: Simple model with attenuation

4.3.4 Simple Model with Neighbors

The simple model with neighbors is similar to the simple model, only this time more influencing factors are considered to make use of the correlation between the expansion of neighboring points. If two points are close enough to each other, which they usually are, because the UAVs try to find points that are close enough to each other to describe the fire, there is a correlation between the two expansion rates. One influencing factor is the wind, which can be assumed to be similar in the same area. Additional factors like steepness of the land, dryness, as well as the burning material (trees, brush, etc.) may or may not differ, but have a higher probability of being similar within close proximity.

This correlation is used by creating a model that has two factors influencing the expansion of the fire, shared factors and non-shared factors.

Because it is not important which of the many underlying factors are shared and which are not, the model combines them all into those two factors, which are assumed to be Gaussian distributed.

To make a prediction about the expansion of the fire in the future, the observed expansion for the last time steps is entered into the model shown in figure 4.13.

Based on this data the most probable distribution of shared and non-shared influencing factors is computed. These factors can not only be used to predict the expansion of a fire at one point, but also describe differences between the different parts of the fire perimeter. Based on the correlation between points close to each other the prediction algorithm can automatically be adjusted. The influence of

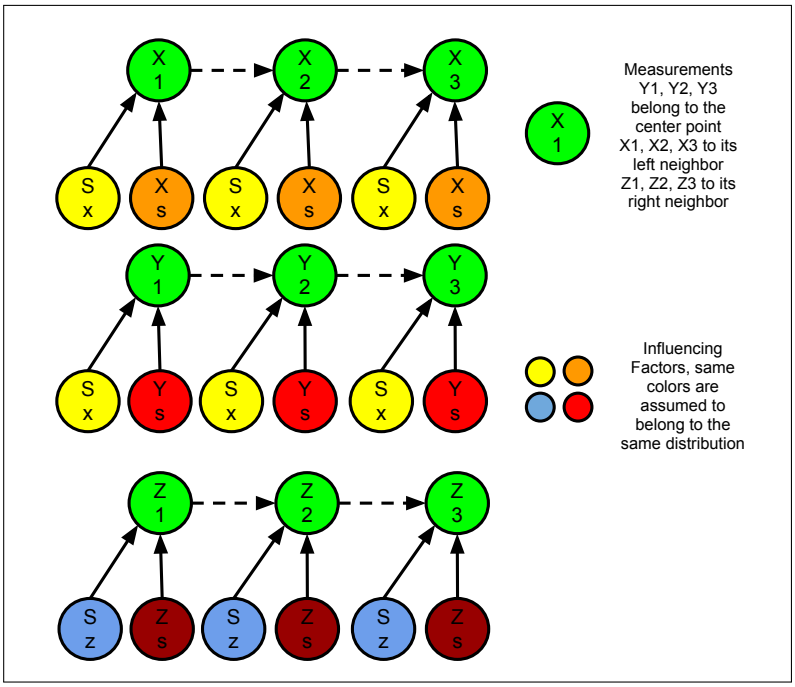

Figure 4.13: Simple model with correlation between all points

neighboring points on the fire prediction of one point can be adjusted according to the relation between the shared and non-shared factors. Additionally, the observed values for the neighboring points can also be adjusted by the difference between non-shared values, before using them to make a prediction about the fire expansion. Using this model will allow to reach the required size of observed values to make predictions faster and using more recent measurements. This allows the model to adjust to changes in factors, like changing wind direction or different burning material, more quickly. To even increase this effect this addition to the model can be combined with the model proposed in 4.3.3 resulting in three factors influencing each measurement.

Common factor with only one neighbor

While some factors, like wind, can be assumed to be shared between the center point as well as its two neighbors, others, like steepness of the terrain or burning material, can not. These factors can be used to make more accurate predictions as well, but this requires another change in the model. The center point in this new model shares one set of factors with its left neighbor and one with its right neighbor. If information about the shared factors of all three points is required this factor can be added to the model similar to the proposal in subsection 4.3.4. Figure 4.14 shows this model without attenuation or common factor between all points.

4.3.5 DBN Models

Another way to model the expansion of the fire is by using a Dynamic Bayesian Network. The UAVs don't require continuous predictions for the fire expansion for their decisions, it is enough to know whether some area of the fire perimeter is expanding slow, fast, too fast or retracting. Because of this a Dynamic Bayesian Network representation of a Hidden Markov Model can be used to make

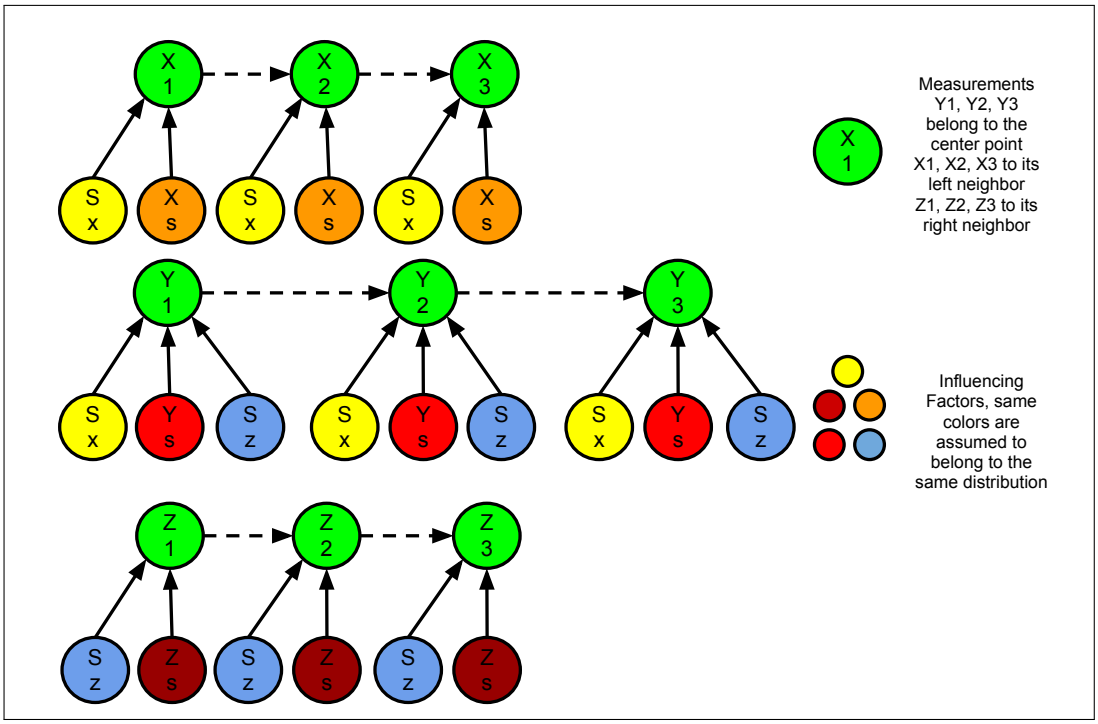

Figure 4.14: Simple model with correlation between neighbors

predictions. The hidden states in this model are the discretization of expansion values of the fire, while the measurements of GPS positions of the UAV while flying around the perimeter are used to infer the expansion. For the simplest version of this model it is assumed that the expansion of the fire at each point is only influenced by its own history. Parameter learning is used to compute the probabilities of state changes. The number of old state changes used to infer theses probabilities has to be carefully chosen. A higher number results in better predictions, but only if the influences on the fires expansion speed did not change in the meantime. If the wind direction changed in between measurements the probabilities for state changes may have changed as well, resulting in inaccurate probabilities.

The probabilities computed using parameters learning represent a model of the probable fire expansion around the perimeter in the future and can be used by the UAVs for their decision making.

Figure 4.15 shows a graphical representation of the Dynamic Bayesian Network, if a Markov 1 Model is assumed , which means that only direct predecessors influence the probability of a state change of the current state. Additionally only the last three historic measurements are used to learn the parameters.

This simple model can be extended to include the states of its direct neighbors as well. This means the probabilities for a point to change depend on the state of its direct predecessor as well as the last states of its direct neighbors.

Model including additional data

All of the models introduced up till now only use the observations made by the UAVs in order to make predictions of the fire expansion. Other factors, however, play a big role in the expansion speed and can be included in the prediction model. Figure 4.17 shows a model that uses a number of different factors

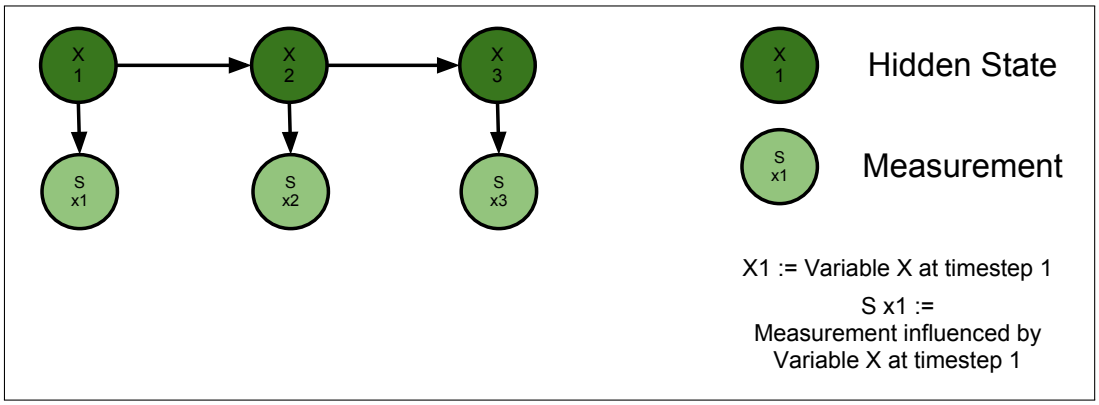

Figure 4.15: Simple Boolean Dynamic Bayesian Network model for fire prediction

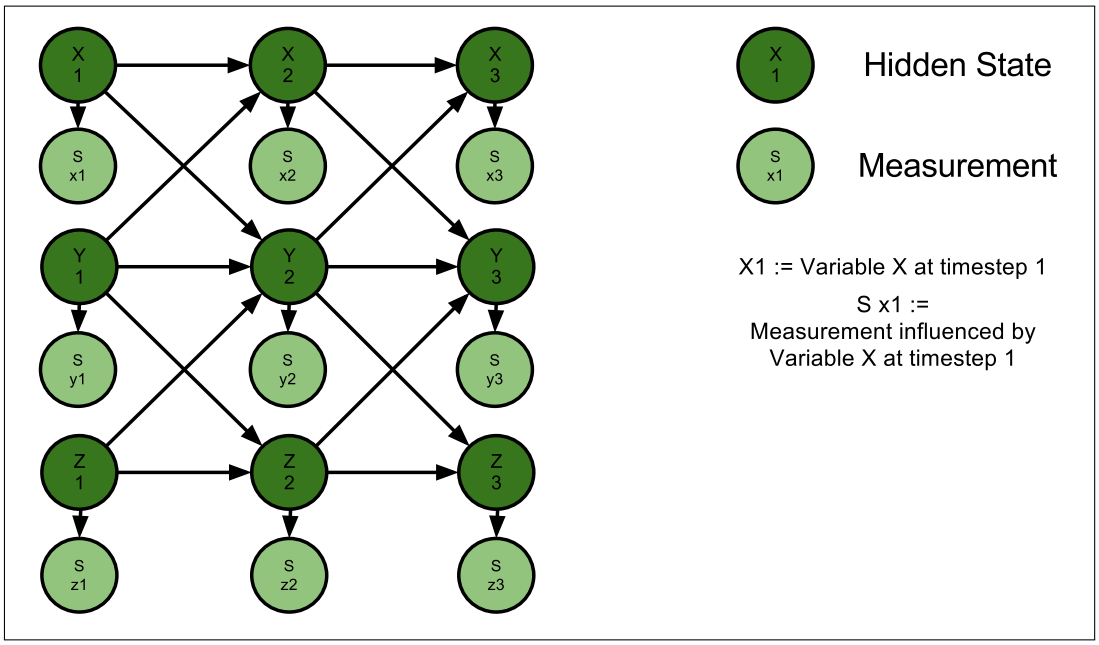

Figure 4.16: Boolean Dynamic Bayesian Network model for fire prediction including neighbors

that can be used as input for the predictions, most of these have to be loaded onto the UAV before the mission begins. If there is a map of the different kinds of trees, bushes and the general consistency of the forest its can be used as the **Burning Material** factor. The **Burning Material** has a direct influence on the expansion speed of the fire, if the wood is fast burning it will expand faster. If it burns hotter it can increase the influence that wind has on the expansion speed, because the air will be hotter and burning particles will have a higher probability to ignite forest areas that are further away. Dryness affects the fire expansion in the same way and is in turn affected by rain in the last days, lakes or rivers in the area as well as steepness of the terrain and the ability of the ground to absorb water. The wind is dependent on the surrounding area as well, if there is a hill blocking the air or a chasm compressing the air and leading it into another direction this affects the fire expansion as well. In addition the weather forecast can be used to as an influence on **Dryness** and the **Resulting Wind**, because the forecast bundles all other meteorological influences that affect the weather. The probability of strong wind if the forecast is strong wind could be used for example. As before the **Expansion Speed** can be inferred from the UAVs position measurements, additionally the **Resulting Wind** can be inferred from the drift of the UAV, if there is wind it results in the UAV having to use different speeds for its rotors in order to achieve the same direction vector. This information could be used to infer the wind direction and strength.

The image shown in figure 4.17 is only one time slice of the DBN, for the whole network each hidden variable is affected by its direct predecessor as well.

4.3.6 Conclusion

While a good prediction of the fire expansion is important for the UAVs to make sound decisions, another important factor has to be considered. The computational power on the UAVs is restricted by weight as well as energy consumption. The model described in subsection 4.11 describes a simple but solid way to predict the fire expansion, it has, however, the drawback of either a slower adjustment to changing fire expansion parameters or a less reliable prediction. This depends on the number of past measurements that are chosen to predict the expansion. A greater number of measurements offers a better reliability, but only as long as the parameters don't change. If the parameters change the prediction will be a mixture of two different expansion models, the one with the old parameters and the new one. Because of this the prediction will be less correct until no more measurements with the old parameters are considered. This problem is overcome with the addition of observed value attenuation in subsection 4.3.3. Introducing another influencing factor increases the adaptivity of the model, but it also increases the computational power required.

The DBN model described in subsection 4.3.5 offers a very good model of the probabilities of state changes for the fire expansion. It offers, however, only Boolean predictions of the fire expansion, while the crisis manager at the base station might be interested in continuous expansion values as well. In addition to that the complexity of the model increases depending on the number of different states for the expansion, especially for the more complex model including neighbors. For a model with three states (not expanding, expanding, expanding fast) the conditional probability table consists of nine

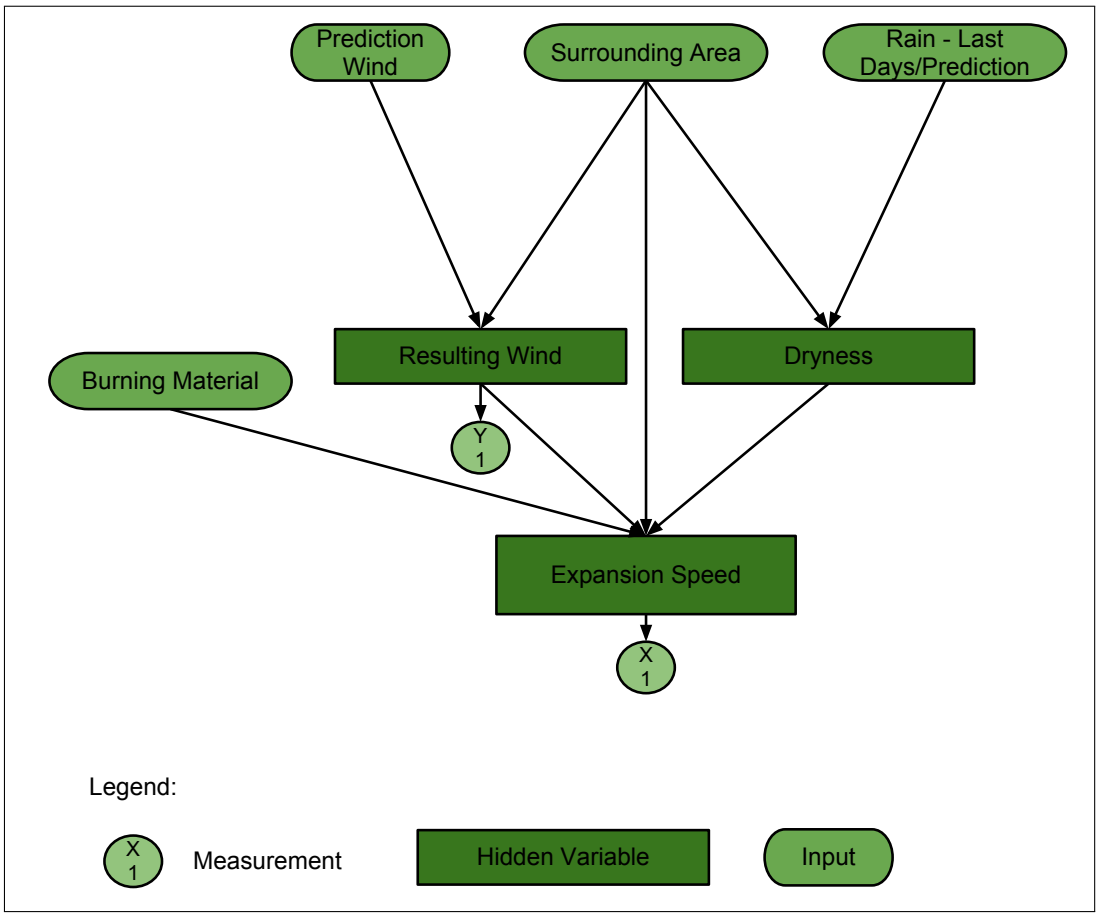

Figure 4.17: DBN prediction using additional information

entries, one for every possible current state with probabilities for every possible change. With only a few measurements of the fire this table will not become very accurate because many possible state changes will not be recorded at all and therefore have either a probability of zero or one that has been set in advance. If more measurements are taken into account this results in a longer history of the fire expansion influencing the conditional probability table. Because some parameters influencing the fire expansion are likely to change (wind, steepness, burning material, etc.) this results in less accurate probabilities as well. The problem becomes even worse for the DBN model including neighbors, because this results in 27 probabilities, three current states for each point and three possible resulting states. The problem can be overcome in parts by using the DBN Model proposed in subsection 4.3.5. While this model consists of even more conditional probability tables these can be filled in advance by using information about the different factors as well as expansion models created by observing similar fires. This model should be able to achieve very good predictions if the input data and the preset conditional probability tables fit well enough.

The model described in subsection 4.3.4 offers a way to decrease the number of measurements required of one point to make accurate predictions, because of its usage of the expansion values of neighbors. Therefore it is able to adjust to changes in parameters faster than the other models. The required computational power is similar to the one described in 4.3.3. If the model is extended using the observed value attenuation as well it further increases the required computational power while increasing its ability to adapt to changing parameters.

In the end the best model for each use-case could be either chosen by the crisis manager before starting the UAVs or by the UAVs themselves. The crisis manager could base his decisions on advance knowledge of the area in which the fire is expected, as well as predictions for changes in wind direction and intensity. The UAV could change between models based on change in expansion during its last measurements as well as the accuracy of predictions made. If the predictions a UAV makes differ from the observed values for a longer period time and in an intensity that is not inside the mission requirements it can decide to switch to a more complex model.

5 Prototype - Fire Model

5.1 Prerequisites

5.1.1 Modelling Programs

There are a lot of different programs that allow to describe dynamic Bayesian networks and support different inference algorithms as well as parameter learning. All of them have different advantages or drawbacks, they require for example additional software, which in some cases is expensive. Some are easy to integrate into other software, while others don't offer this possibility at all. There are huge differences in the available interfaces, while some offer graphical modelling others require code. For the decision which software to take two different products seemed promising and have been compared for the implementation of the different models proposed in chapter 4. The combination of Matlab and BNT and Infer.net.

BNT is the Bayes Net Toolbox, a toolbox for Matlab, which has been developed by Kevin Murphy at the University of California, Berkeley . It supports a number of different conditional probability distributions that can be used for modelling nodes, including

- Gaussian
- Multi-layer perceptron, for neural networks
- Deterministic

It supports static as well as dynamic Bayesian networks and offers many different inference algorithms for dynamic Bayesian networks:

- Exact inference
  - junction tree
  - frontier algorithm
  - forwards-backwards
  - Kalman-RTS
- Approximate inference
  - Boyen-Koller
  - factored-frontier / loopy belief propagation

The supported parameter learning methods are Batch MLE/MAP using EM as well as Sequential/-batch Bayesian parameter learning. BNT however requires MATLAB to run and can not easily be

integrated into other software without it, this is especially a problem for running it on the processor of a UAV.

Infer.net [MWGK10] is a project of the Microsoft research team in Cambridge, team members include Tom Minka and John Guiver. It offers a framework for running Bayesian inference in graphical models. It offers message-passing algorithms as well as graph algorithms and linear algebra routines. It allows for Bayesian networks as well as undirected models with both discrete and continuous variables. As inference algorithms Infer.net supports expectation propagation, variational message passing and block Gibbs sampling in its current form.

Infer.net is still in its beta phase but already offers the features required for the implementation of the rather simple prediction models proposed in chapter 4.

Its main advantages are that it offers a high computational efficiency and that it is a stand alone framework that does not rely on other software. It can easily be included into self written code, including C, Managed $C++$, F, IronPython and others, create probabilistic programs that make decisions based on the inferred values of Dynamic Bayesian Networks. Because of this Infer.net has been chosen to implement the models in this thesis.

5.1.2 Infer.net

In this section the basics about Infer.net will be discussed to allow an understanding of the source code of the fire model prototype. A more complete description including all factors, models and distributions can be found in reference [Cam11] and [Mi11] on which most of the information in this section is based.

The basic building block for probabilistic programs with Infer.net are the random variables. The random variables are implemented int the Variable<T> class. They can be either discrete or continuous and their domain type is defined by T. Discrete random variables are either of type bool or int, and therefore restricted in the number of their possible values. Continuous random variables are of the type double and can take an infinite number of values.

As with usual variables a Infer.net variable needs to be declared and defined to be used, which is shown in the following example for a continuous variable x of type double. The first point is the variable declaration and the second point the definition.

```
Variable<double> x;
x = Variable<double>.New();
```

At this point the variable is declared and defined, but does not have an associated distribution yet. To achieve this the variable must be statistically defined using the statement

```
x = Variable.Random<bool>(Dist);
```

where Dist is a previously chosen distribution.

There are different ways to execute these steps,depending on the chosen variable and distribution. The following examples show a few possibilities for declaring, defining and statistically defining variables. In many cases this is possible in one single step. To create a simple, Bernoulli distributed

variable, a Boolean variable that takes 1 as a value with probability p and 0 with probability $1-p$ with $0 < p < 1$ the following code is used:

```
Variable<bool> BernoulliVariable = Variable.Bernoulli(p);
```

Other, more complex variables, like Gaussian or Gamma distributions can be created using different describing parameters, for example **mean** and **precision** or **variance** for a Gaussian distribution or **shape** and **scale** for a Gamma distribution. Explanations for these parameters can be found in sections 4.1.2 and 4.1.3. Listing 5.1 shows two different ways to create a Gaussian distributed variable as well as a Gamma distributed variable.

Listing 5.1: Random variables with different distributions and different ways to create them

```
Variable<double> GaussianVariable1 =
Variable.GaussianFromMeanAndVariance(Mean, Variance);

Variable<double> GaussianVariable2 =
Variable.GaussianFromMeanAndPrecision(Mean, Precision);

Variable<double> GammaVariable =
Variable.GammaFromShapeAndScale(Shape, Scale);
```

To be able to use these random variables to make predictions a few more steps are necessary. First these variables have to be combined to create a model. In the case of a very simple expansion prediction algorithm for a fire, like the one described in section 4.11 a number of these variables have to be defined according to the number of historical expansion points that should be used for the prediction. These variables are assumed to be dependent from the same distribution, which is achieved by using a Gaussian variable as input for the mean and a Gamma variable as input for the precision for each of the Gaussian variables used to describe the historic data. The code to achieve this can be found in listing 5.2

Listing 5.2: Two random variables for observed expansions influenced by the same distribution

```
Variable<double> averageExpansion =
Variable.GaussianFromMeanAndPrecision(0, 0.01);

Variable<double> noise = Variable.GammaFromShapeAndScale(2.0, 0.5);

Variable<double> observedExpansion1 =
Variable.GaussianFromMeanAndPrecision(averageExpansion, noise);

Variable<double> observedExpansion2 =
Variable.GaussianFromMeanAndPrecision(averageExpansion, noise);
```

The variables **observedExpansion1** and **observedExpansion2** are both created with **averageExpansion** as mean and **noise** as precision distributions, while those two have been initialized using fixed values. The 0 as mean for **averageExpansion** assumes an average expansion of 0, while the precision of 0.01 expresses that there is a large uncertainty for this mean.

At this point the **observedExpansion** variables have only been created, but not yet filled with observed data. Whenever the UAV records a new observation of the fire expansion it has to be assigned to the variable in order to train the model. This is done using the variables **ObservedValue** property. The example shows how to assign an expansion value of 3 to the ObservedValue property of the **observedExpansion1** variable.

```
ObservedExpansion1.ObservedValue = 3;
```

For a model with only two **observedExpansion** variables the observed values would be assigned to **observedExpansion1** and **observedExpansion2** alternately.

To create a more complex model it is necessary to create variables that are influenced by two or more different distributions. This can be done, for example, by assigning a linear combination of two variables to the shape and scale parameters of the random variable that is dependent on those values. Listing 5.3 shows the necessary code for this.

Listing 5.3: Two distributions influencing one random variable

```
Variable<double> averageExpansion1 =
Variable.GaussianFromMeanAndPrecision(0 , 0.01);

Variable<double> noise1 =
Variable.GammaFromShapeAndScale(2.0, 0.5);

Variable<double> averageExpansion2 =
Variable.GaussianFromMeanAndPrecision(0 , 0.01);

Variable<double> noise2 = Variable.GammaFromShapeAndScale(2.0, 0.5);

Variable<double> observedExpansion =
Variable.GaussianFromMeanAndPrecision
(averageExpansion1 + averageExpansion2, noise1 + noise2);
```

At this point there is still no difference to a model using just one random distribution to describe how **observedExpansion** is influenced. This is due to the fact that both random distributions are only used to describe this one variable. All observed values for **observedExpansion** are used in the same way to infer the distributions **averageExpansion1** and **averageExpansion2** as well as their noise distributions. Therefore these distributions do not differ, even if different starting values for their mean and precision would have been chosen the two distributions would converge towards each other. This would lead to the same result as using only one distribution would.

To make use of the two influencing distributions there has to be another set of observed values that depends on one of the distributions, but not on the other. Using this model results in one distribution that describes the similarity of the influences for both observed variables. In the case of fire perimeter prediction this second set of observations would be the expansion values for the neighboring point.

Listing 5.4: Two observed expansion points with a shared distribution

```
Variable<double>observedExpansion1 =
Variable.GaussianFromMeanAndPrecision
(averageExpansion1 + averageExpansion2, noise1 + noise2);

Variable<double> observedExpansion2 =
Variable.GaussianFromMeanAndPrecision
(averageExpansion2 + averageExpansion3, noise2 + noise3);
```

In listing 5.4 **averageExpansion2** together with its noise distribution **noise2** is shared between **observedExpansion1** and **observedExpansion2**. Therefore, it represents the shared influences on the expansion at point 1 and 2.

With these building blocks more complex models can be created and trained using training data. To be able to make predictions, however, an instance of an inference engine has to be created and the right inference engine for the use-case has to be selected. The inference algorithm used by the inference engine can be set using the algorithm property of the inference engine object. The two steps involved can be seen in listing 5.5. The inference engine is used to compute the posterior distribution for a specified random variable according to the model's other random variables. This distribution is usually called the marginal distribution [Cam].

Listing 5.5: Creation of an instance of the inference engine and engine selection

```
\\creation of the inference engine instance
InferenceEngine engine = new InferenceEngine();

\\select different inference engines
\\Choose Expectation Propagation
engine.Algorithm = new ExpectationPropagation();
\\Choose Gibbs sampling
engine.Algorithm = new GibbsSampling();
\\Choose Variational Message Passing
engine.Algorithm = new VariationalMessagePassing();
```

The difference between the different inference engine algorithms is described in 5.1.3.

With the model defined and the right inference engine selected it is possible to make predictions using the inference engine. This can be done by creating a variable that depends on the same distribu-

tions as the observedVariables belonging to the point at which the expansion should be predicted. An example of this can be seen in listing 5.6.

Listing 5.6: Expansion prediction using the inference engine

```
Variable<double>predictedExpansion1 =
Variable.GaussianFromMeanAndPrecision
(averageExpansion1 + averageExpansion2, noise1 +  noise2);

Gaussian predictedExpansionDist =
engine.Infer<Gaussian>(predictedExpansion1);

double expansionMean = predictedExpansionDist.GetMean();
double expansionStdDev =
Math.Sqrt(predictedExpansionDist.GetVariance());
```

After this **expansionMean** is a double variable containing the mean of the predicted expansion, while **expansionStdDev** contains the standard deviation of this distribution, meaning its uncertainty.

A lot of different models can be created, filled with observation data and used for inference using these building blocks. section 5.3 explains the interface and principles of a the simulator that has been build out of these (and similar) building blocks and can be extended with additional models without deeper knowledge of the simulator itself.

5.1.3 Inference algorithms

This section only describes the general functionality of the three algorithms, as well as their advantages and disadvantages. More information can be found in the cited literature.

Expectation propagation

Expectation propagation combines assumed density filtering, an extension of the Kalman filter and loopy belief propagation, an extension of belief propagation in Bayesian networks into one new algorithm.

Loopy belief propagation propagates exact belief states and is therefore only useful for some belief networks, for example purely discrete networks. Expectation propagation approximates the belief states with expectations. Instead of an exact belief state it uses an expectation for the mean with its related uncertainty.

Assumed density filtering processes observations one by one, and calculates posteriors directly from this observation. These are used to process the new observations.

Expectation propagation extends assumed density filtering with refinements to earlier approximations, by making additional passes. Whenever new information is observed the choices made in earlier approximations are refined. Because of this expectation propagation is independent of the ordering of observed information.

Expectation propagation is a deterministic algorithm, which means that it will always give the same solution for the same initialization. In most cases expectation propagation does not calculate an exact result. This is possible only in some simple cases, for example for factor graphs that are discrete or linear Gaussian and do not contain loops. Expectation propagation is not guaranteed to converge to a solution, neither exact nor approximate. Its efficiency is not as high as variational message passing in most cases, but higher than Gibbs sampling. [Min01]

Variational message passing

Variational message passing is similar to the EM algorithm. The only difference is that it learns distributions over parameters, instead of maximum likelihood estimates. It calculates its predictions by sending messages between nodes in the model network and then updating posterior beliefs at each node using only local functions. Parents send the expectation of their sufficient statistic, a statistic that completely describes their statistical model, to their children. Children send their natural parameters to their parents. The variational message passing algorithm calculates a lower bound on the evidence and maximizes it. This can be used to detect convergence.

Like expectation propagation variational message passing is deterministic. It is guaranteed to converge and the most efficient of the supported algorithms for most cases. It offers, however no exact result.

[Win03]

Gibbs sampling

Gibbs sampling is a Marcov chain Monte Carlo algorithm used to obtain random samples from the joint probability distribution of multiple random variables. Gibbs sampling creates a Marcov chain of samples, where each sample is correlated with nearby other samples. In the simplest version each variable is visited in turn and sampled from its conditional distribution, depending on the current state of its neighbors. All other variables are not changed. The order in which the variables are visited does not matter, neither does the amount of visits as long as every variable is visited often enough.

A more efficient version is the Gibbs sampling method, where all variables are grouped into acyclic blocks. As with the single variables each block is visited in turn and its distribution conditioned on all other variables is sampled. In Infer.net you don't have to explicitly define the blocks, as default blocks are automatically created based on the model.

In order to get independent samples from the Gibbs sampling thinning can be applied. With this option chosen only every $n-$th sample from the chain is taken in order to remove correlation between nearby samples. Because the samples from the beginning of a chain are not necessarily representing the desired distribution they can be discarded, which is known as the burn-in period. Both values can be specified if using the Gibbs sampling algorithm in Infer.net.

Gibbs sampling is the only undeterministic algorithm supported by Infer.net. Undeterministic means that depending on the random seed it will compute different samples with each execution. It is guaranteed to converge, but less efficient than both deterministic algorithms.

[Gem84]

5.2 Data Generator

To test the different models for fire expansion prediction, test data is required. The optimal way to get this data would be using real UAVs flying along a fire. Since this is not possible for me as of yet, the second best option would be to get fire expansion data from a fire simulation program. There are a number of fire models and fire modelling programs using these models. These include for example Farsite [Fin98], PHOENIX [APK07], Prometheus [(CI12] and BEHAVE [PLA98]. But none of these programs offer an interface for using the generated data in other programs. The output is mainly created in pictures or different tables and is not directly usable to create input data for the fire expansion prediction models.

The data from these programs could only be used after applying, for example, image processing techniques to read the expansion data over time out of images. Because this would pose an immense additional workload it is out of the scope of this thesis. The prototype, however, should at some point be tested with either real data or data created using one of these more sophisticated models.

For general testing during the construction of the software prototype I used a self-made data generator. The data is generated based on wind direction, the fire center and randomization. If the fire center is used an equal expansion into all directions away from the fire center is assumed. This expansion is then altered by the wind direction and, in the end, randomized to simulate the difference in burning material. The formula of this model can be seen in equation 5.1

$$\begin{pmatrix} ExpansionX \\ ExpansionY \end{pmatrix} = \left(\begin{pmatrix} PointX - FireCenterX \\ PointY - FireCenterY \end{pmatrix} + \begin{pmatrix} WindX \\ WindY \end{pmatrix} \right) * RandomFactor \tag{5.1}$$

Obviously this model is only a very rough model of real fire expansion, since important influencing factors, like dryness, steepness of the area, burning material, ... are not included. The generated data can only be used to test the basic principles of the fire expansion prediction prototype. Since both parts of the program have been created by me the prediction method might be biased based on the assumptions I made for the data generation. Due to this, other tests are, as described earlier, absolutely necessary.

The data is generated in the way the UAVs would find record data if flying back and forth along part of the perimeter. This means a list is created for every observation point. Whenever the UAV would fly over this point a new entry is created.

Figure 5.1 shows the part of the data generation program where the parameters can be set and 5.2 shows an example output. In addition to this written output, lists with all values for all variables are saved.

5.2.1 Save / Load

Because of the considerable drawbacks of the self-made data generator there is an additional way to gather data for usage in the Simulator. A save and load data function can be used to either store data that has been created using the generator in an XML file or load data created using external sources. This data could be extracted from professional fire simulation programs, gathered directly using UAVs,

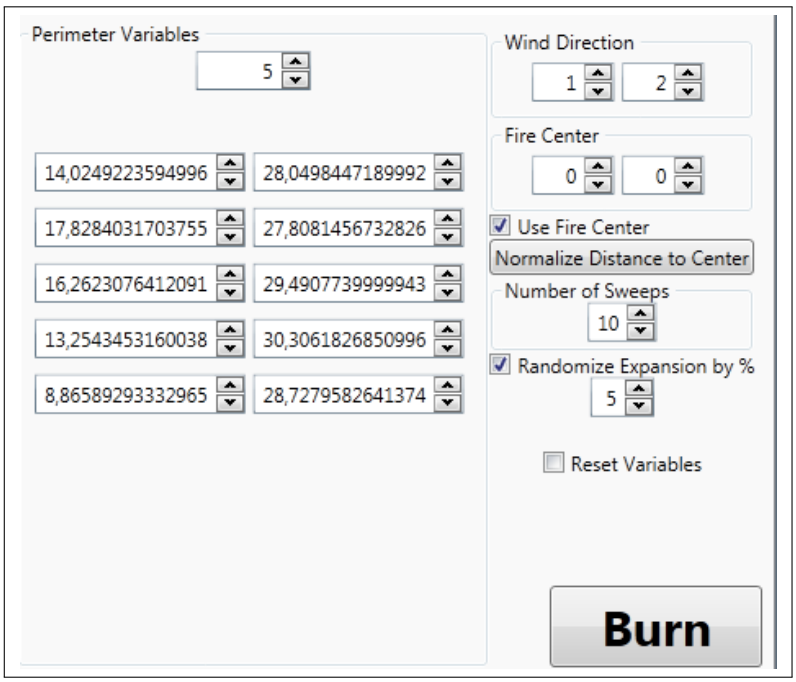

Figure 5.1: Fire expansion data generator settings

```
Forward Sweep: (1,00 ; 0,00)  (1,49 ; 1,50)  (1,03 ; 1,98)    Backward Sweep: (1,03 ; 1,98)  (1,78 ; 1,79)  (2,03 ; 1,04)
Forward Sweep: (2,03 ; 1,04)  (2,28 ; 2,30)  (2,01 ; 3,03)    Backward Sweep: (2,01 ; 3,03)  (2,66 ; 2,68)  (3,07 ; 2,03)
Forward Sweep: (3,07 ; 2,03)  (3,17 ; 3,17)  (3,04 ; 4,02)    Backward Sweep: (3,04 ; 4,02)  (3,61 ; 3,61)  (4,03 ; 3,02)
Forward Sweep: (4,03 ; 3,02)  (4,14 ; 4,11)  (4,09 ; 5,01)    Backward Sweep: (4,09 ; 5,01)  (4,61 ; 4,58)  (5,07 ; 4,00)
Forward Sweep: (5,07 ; 4,00)  (5,11 ; 5,10)  (5,05 ; 5,98)    Backward Sweep: (5,05 ; 5,98)  (5,57 ; 5,55)  (6,05 ; 4,97)
```

Figure 5.2: Example output

taken from historical fire expansion data or created by hand.

This function allows easy additional tests with the simulator to reassess the validity of the prediction algorithms as soon as better data is available.

5.3 Simulator

Once data describing points along the fire perimeter has been gathered, in a way that is comparable to a UAV flying along the perimeter and collecting data points, it can be used to test the different prediction algorithms.

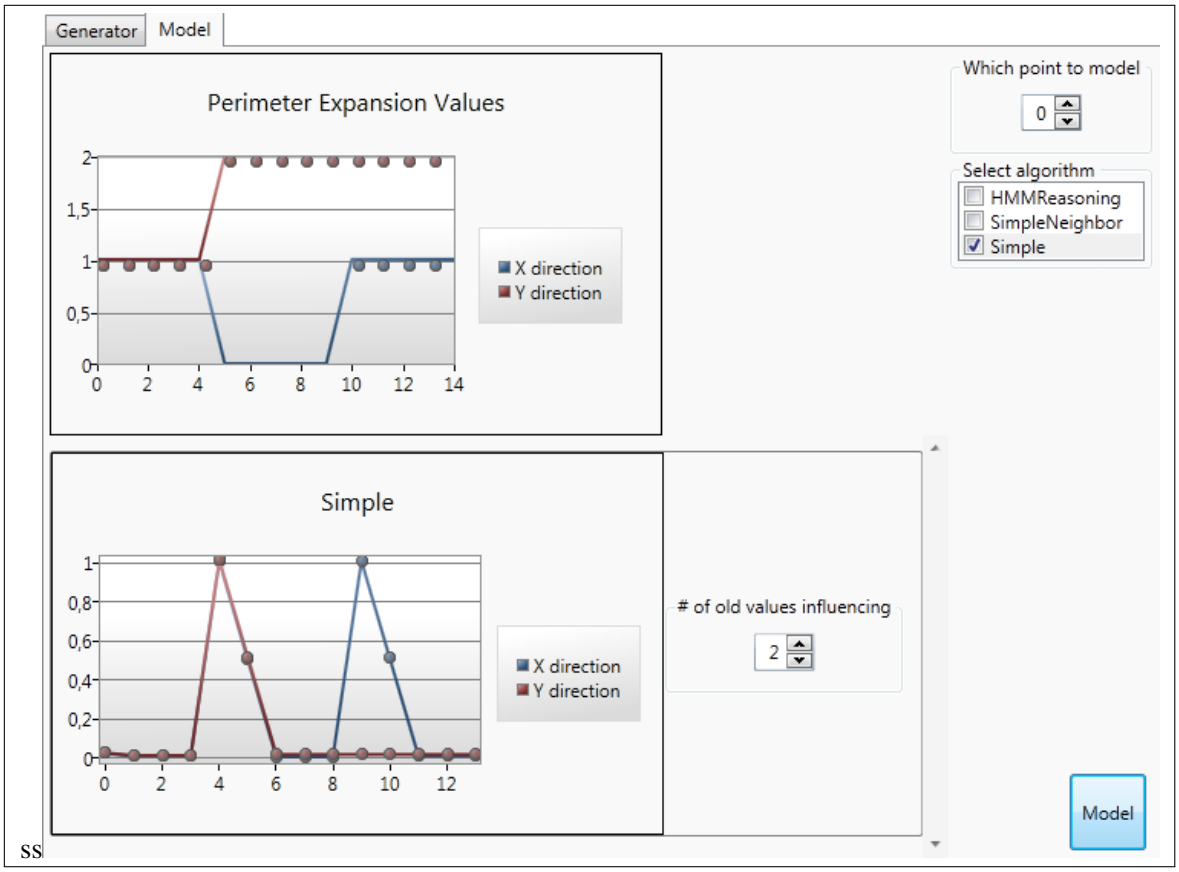

Figure 5.3: Simulator - graphical user interface

To start a comparison the point on the perimeter that should be modeled has to be chosen using the **Which point to model** selector. The next step is decide which algorithms should be used. The algorithms shown in **Select algorithm** are gathered automatically from the assembly of the prototype code. Subsection 5.3.1 describes the interface used for these algorithms. Implementation of new algorithms is possible without knowing the rest of the prototype code. Any number of algorithms can be chosen using the check box next to them. As soon as an algorithm is selected a new coordinate system with its name is created, figure 5.3 shows this for the algorithm named **Simple**. Next to the coordinate system an integer box **# of old values influencing** is added. This box can be used to decide how many old measurements of the fires expansion for one point should influence the predictions made by the algorithm.

The coordinate system named **Perimeter Expansion Values** always shows the expansion data for the point selected in **Which point to model**. The x-Axis shows the number of the measurement, the first measurement taken of the expansion at the selected point is at the left at $x = 0$, the latest measurement is at the right corner of the x-Axis, all measurements in between are ordered by the time of measurement. All values on the y-Axis correspond to the expansion at the selected point at the time step defined by the corresponding value on the x-Axis. The expansion values are split up by expansion into X and Y direction. These directions in reality would be longitude and latitude coordinates from the GPS module.

Similar to this the coordinate systems for each algorithm show the measurements on the x-Axis and the prediction error for each measurement on the y-Axis. The prediction error is the difference between the prediction that has been made by the algorithm for each measurement and the real value shown in **Perimeter Expansion Values**.

In addition to this graphical output a few numerical results are displayed. Figure 5.4 shows an example of this output.

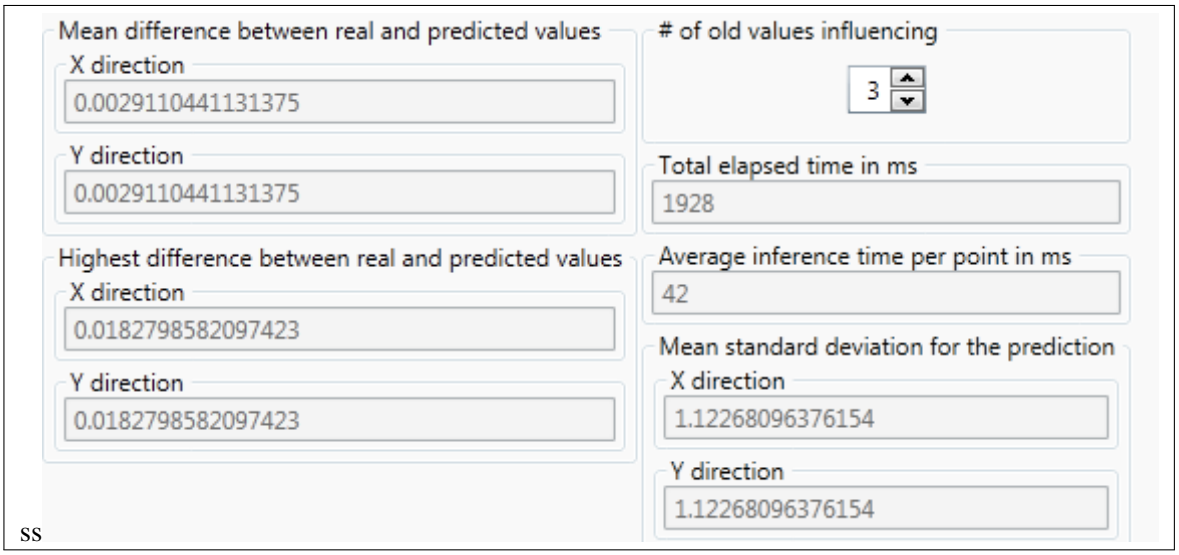

Figure 5.4: Numerical simulation output

The boxes on the left show the mean difference between real and predicted values as well as the highest difference between real and predicted values, both in X and Y direction. On the top right side of the figure two boxes show the elapsed time for the execution of the algorithm. The box titled “Total elapsed time in ms” shows the total time for execution, this includes the compilation of the statistical model as well as observation and inference for all points in the data set. “Average inference time per point in ms” only includes the time required to add one new measurement as observation to the model and infer the expansion in the next time step as well as storing that prediction in memory. The last two boxes show the mean standard deviation for the prediction in X and Y direction and offer a measurement for the certainty of the predictions made by the algorithms.

5.3.1 Implementing additional algorithms

Implementing additional algorithms can be done by simply implementing a new class, in the prototype solution, that uses one of the two prediction interfaces, which are shown in listing 5.7.

Listing 5.7: Reasoning Interfaces

```
interface IReasoning
{
List<PredictionOutput> Generate(List<Vector> p_Input, int p_Length);
}

interface IReasoningNeighbor
{
List<PredictionOutput>
Generate
(List<Vector> p_Input, List<Vector> p_InputNeighborLeft,
List<Vector> p_InputNeighborRight, int p_Length);
}
```

The input for the prediction class consists either of a list of input vectors **p_Input** and an integer **p_Length** or those two variables and two additional vectors **p_InputNeighborLeft** and **p_InputNeighborRight**.

When deciding which of the interfaces should be used it should be considered whether the algorithm is dependent only on historical data from one point on the perimeter or on the expansion history of its neighbors as well. **p_Input** contains the historical expansion data for one point, **p_InputNeighborLeft** and **p_InputNeighborRight** the same data for its neighboring points. **p_Length** is an integer describing how many historical values should be used for each prediction, a $'2'$ as value would mean that the last two measured expansion values are considered in the prediction algorithm.

The **PredictionOutput** variable consists of an integer variable for the **Index** and three vector variables **Real**, **Prediction** and **Difference**, where **Real** holds the real expansion values, **Prediction** the prediction made corresponding to the **Real** value and **Difference** the difference between those two. **Index** is the number of predictions made up to this point. **StdDeviation** is the standard deviation for each prediction and **InferenceTime** the time that inference for the corresponding point took.

Difference and **Index** are necessary as additional variables, because the Windows Presentation Foundation (WPF) feature of direct data binding is used to connect the values of **Index** and **Difference** directly to the graphs used for representation.

The output variable **PredictionOutput** can be filled with output data using the function shown in listing 5.8. In this listing the type of each variable as well as a placeholder name is shown, to execute the function computed values fitting this type have to be used instead. **Difference** is automatically computed when using this creation method.

Listing 5.8: PredictionOutput definition and value assignment

```
PredictionOutput Output =
new PredictionOutput(int p_Index, Vector p_Prediction, Vector p_Real,
Vector p_StdDeviation, long p_InferenceTime));
```

For both interfaces the defined output is a list of predictionOutput variables. This list should contain all predictions made, in the order they have been made in, at the point where this output is returned.

The class itself should offer an arbitrary function that transforms the input variables into the required output.

5.4 Simulation Results

In order to evaluate the different models proposed in chapter 4.3 the underlying algorithms have been implemented for all stochastical models and an additional model called “simple neighbor”. The simple neighbor model is basically identical to the simple model, but it adds the newest measurement of one neighbor as an observed value for the current point. This measurement influences the same distribution as another observed value for the current point would. Due to this the prediction can be improved, if there is a very high correlation between neighboring points. This model is not usable for real use cases, because this correlation can not be assumed. It is, however, a good measurement for the possible improvements for the other algorithms.

For these implementations the expansion vectors provided by the data generator have been split up into X and Y direction, which are evaluated by the algorithms separately. This allows to find correlation in only X or Y direction. A set of different evaluation scenarios covering the possible expansion behaviors for a fire model has been prepared and the corresponding fire expansion data sets generated. The different scenarios include

- constant expansion
- constant increase in expansion speed
- phases with increasing and decreasing expansion speed with change in every second time-step

with both high and lower correlation in the expansion between neighbors. High correlation has been achieved by only using the *wind* variable in the data generator, which adds a fixed vector to all expansion points. For lower correlation the *fire center* has been used additionally, which adds an expansion for each point based on its relative position to the fire center. In addition to those models there are two more, one with small changes in expansion speed every few time-steps, which is closer to a real fire scenario than the other, more extreme, cases. Last but not least there is another model which expands purely based on the perimeter points position in relation to the fire center. This model has a very low correlation with its two neighbors for X or Y direction of the expansion respectively, which allows for an evaluation of the performance of the algorithms for expansion data with almost no correlation.

For all algorithms only the last 3 measurements of expansion speed have been used to make predictions of the future fire behaviour in order to adjust faster to changes in expansion parameters.

All simulations have been executed on a Intel® Core™ i5 CPU with 3.20GHz and 6 GB RAM running a 64-bit Windows 7 Enterprise operating system.

5.4.1 Visual output

In the upcoming subsections figures are used to convey the results for each algorithm. There are two types of figures, **Expansion values** and **Visual output**, there caption always begins with these words. For both figures the x-axis shows the number of the measurement and therefore the time steps. For **Expansion values** the y-axis shows the expansion of the fire in X and Y direction for the corresponding time step. The y-axis in **Visual output** figures shows the difference between the prediction of an algorithm and the real expansion value. Each peak in these graphs responds to a prediction that is considerably worse than usual. A constant high value for y corresponds to a continuously bad prediction. Ideally this curve should always be close to 0 or, after a peak return to 0 as fast as possible.

5.4.2 Constant expansion

For the constant expansion an expansion speed of 1 unit, which could be meters, feet or whatever measurement unit is desired, has been selected. This means in the time between measurements the fire has expanded 1 in X and 1 in Y direction.

High correlation

For the high correlation, both neighbor points expand exactly in the same way as the point itself as can be seen in figure 5.5.

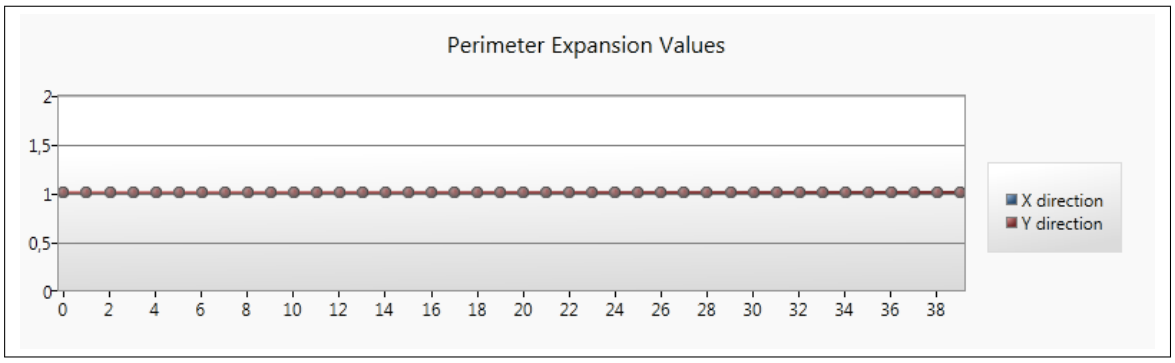

Figure 5.5: Expansion values for constant expansion

The predictions made by the different algorithms are all very similar and the visual output looks close to the one shown in figure 5.6. The higher difference between prediction and real value for the first two predictions can be explained by the number of observations made up to that point. At time step 0 there was only one observation available to infer the next expansion value, at time step 1 only two. From time step 2 on there were always 3 measurements available to base the inference on, and the expansion was constant for all measurements, therefore the prediction does not change anymore.

Although the visual output looks similar for all algorithms there are differences in the predictions which can be seen in table 5.1, which shows the mean differences between predictions and real values

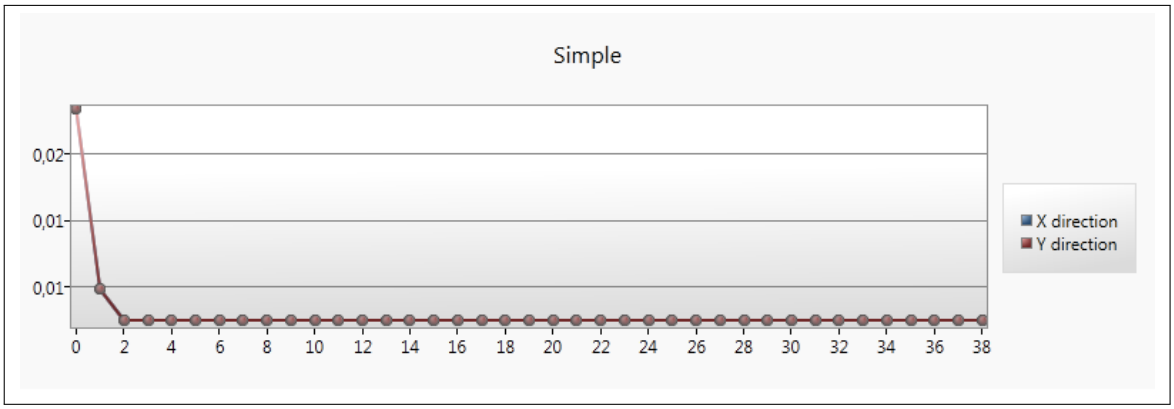

Figure 5.6: Visual output for the simple algorithm - Constant expansion - High correlation

as well as the maximum differences and table 5.2, which shows the standard deviation of the predictions and the execution times.

Table 5.1: Constant expansion- High correlation - Differences

Constant Expansion - High Cor	Mean Diff X	Mean Diff Y	Max Diff X	Max Diff Y
Simple	0.0029	0.0029	0.0183	0.0183
Simple Attenuation	0.0024	0.0024	0.0099	0.0099
Simple Neighbor	0.0007	0.0007	0.0020	0.0020
Neighbor	0.0003	0.0003	0.0018	0.0018
Neighbor No Common Distribution	0.0004	0.0004	0.0005	0.0005

Table 5.2: Constant expansion- High correlation - Standard Deviation and Time

Constant Expansion - High Cor	StdDev X	StdDev Y	Total Time in ms	Step Time in ms
Simple	1.1227	1.1227	987	25
Simple Attenuation	0.8230	0.8230	1163	29
Simple Neighbor	0.7609	0.7609	1167	29
Neighbor	0.8449	0.8449	1631	41
Neighbor No Common Distribution	0.8930	0.8930	1884	48

As expected the more complex algorithms involving the neighboring points in the prediction perform better than both simple algorithms, depending on the algorithm between three and ten times in relation to the mean difference. The neighbor algorithm with a common distribution between all neighbors performs better than the one with two shared distributions between the point and each one of it neighbors in relation to the mean difference, but worse in relation to the maximum difference. Execution time increases with an increase of the complexity of each algorithm, which was to be expected.

Lower correlation

For the lower correlation the expansion for neighboring points differs from the ones for the original point. Each one has a higher difference in either X or Y direction. The expansion values for the left and right neighbor can be seen in figure 5.7.

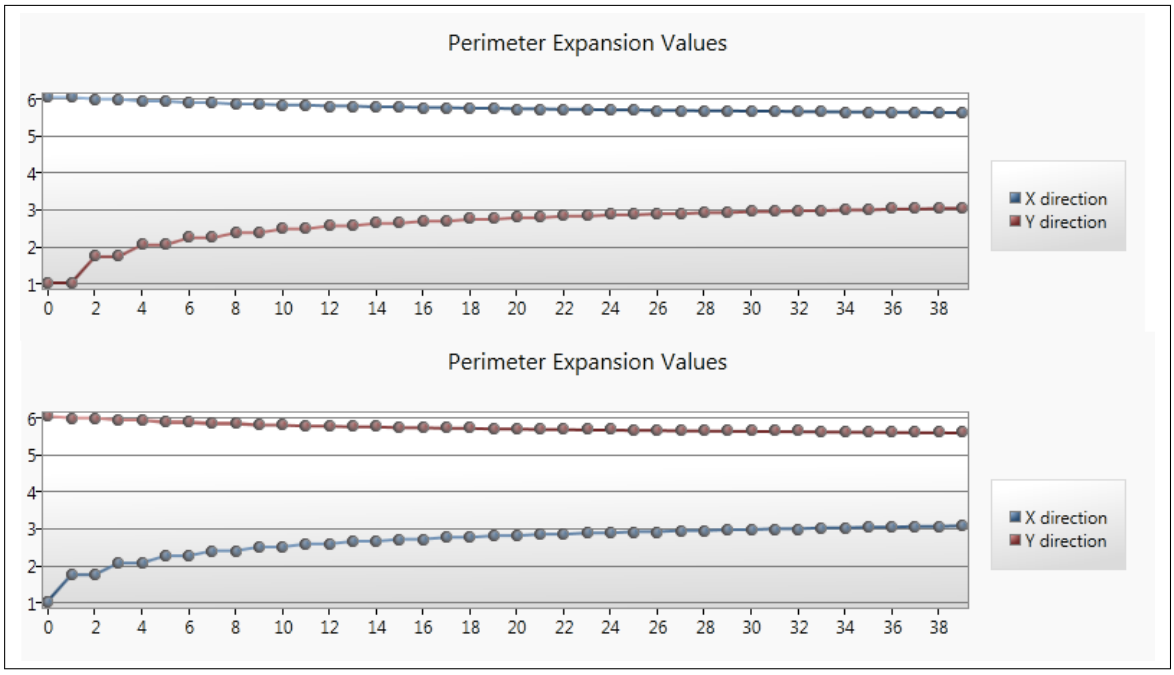

Figure 5.7: Expansion values for both neighbors for constant expansion

Again the visual output for most of the algorithms is similar to the one of the simple algorithm shown in figure **??**, but this time the output for **Simple Neighbor** and **Neighbor No Common Shared** differ noticeably. Their output can be seen in figure 5.8.

As before the differences in the predictions can be seen in table 5.3 and table 5.4.

Table 5.3: Constant expansion- Lower correlation - Differences

Constant Expansion - Low Cor	Mean Diff X	Mean Diff Y	Max Diff X	Max Diff Y
Simple	0.0132	0.0132	0.0831	0.0831
Simple Attenuation	0.0111	0.0111	0.0449	0.0449
Simple Neighbor	0.9623	1.0012	1.8949	2.6657
Neighbor	0.0022	0.0022	0.0148	0.0139
Neighbor No Common Distribution	0.0035	0.0023	0.0065	0.0123

Notable about these results is that the **Neighbor** algorithm still performed best, although the correlation between neighbors is lower. The **Simple Neighbor** algorithm performed very bad, which is due to the fact that it assumes a high correlation between neighbors, but this is not the case for this input data.

5.4.3 Increasing expansion speed

The data set with increasing expansion speed consists of expansion data that increases by one with every two time steps. The visual representation of this input data is shown in figure 5.9.

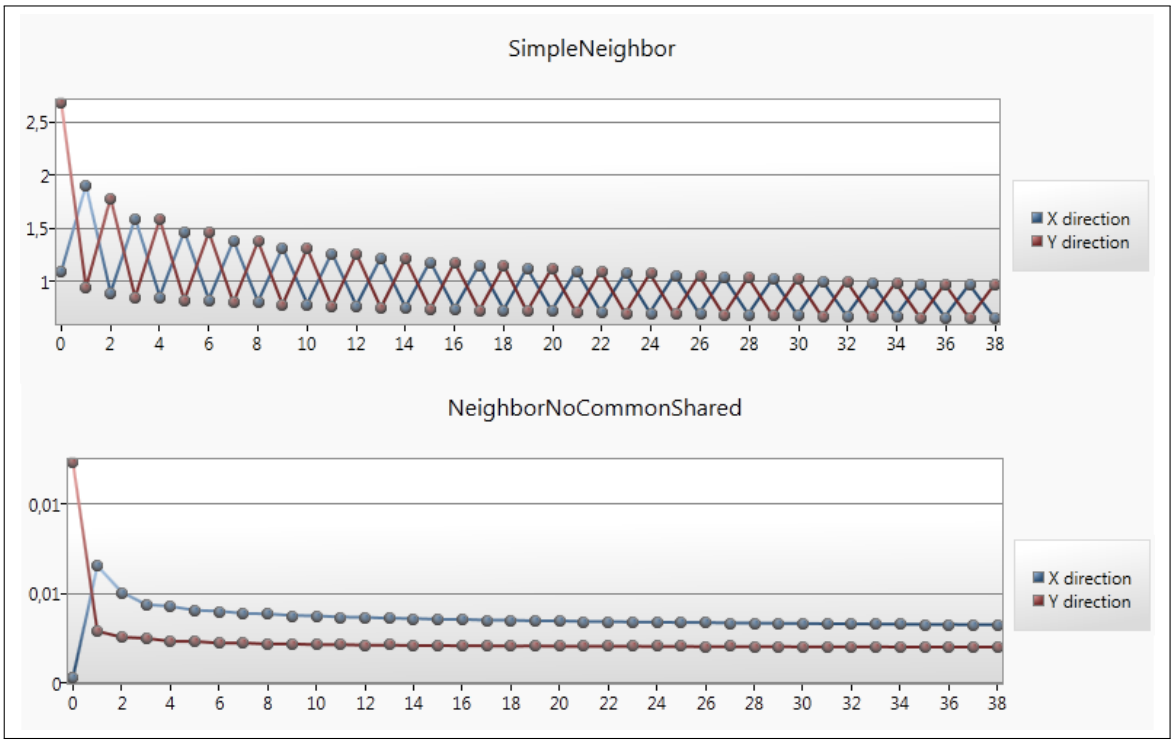

Figure 5.8: Visual output for the **Simple Neighbor** and **Neighbor No Common Shared** algorithms - Constant expansion - Low correlation

Table 5.4: Constant expansion- Lower correlation - Standard Deviation and Time

Constant Expansion - Low Cor	StdDev X	StdDev Y	Total Time in ms	Step Time in ms
Simple	1.1229	1.1229	1017	26
Simple Attenuation	0.8231	0.8231	1085	27
Simple Neighbor	0.9176	0.9286	1648	41
Neighbor	0.8450	0.8450	2149	54
Neighbor No Common Distribution	0.8931	0.8931	2536	64

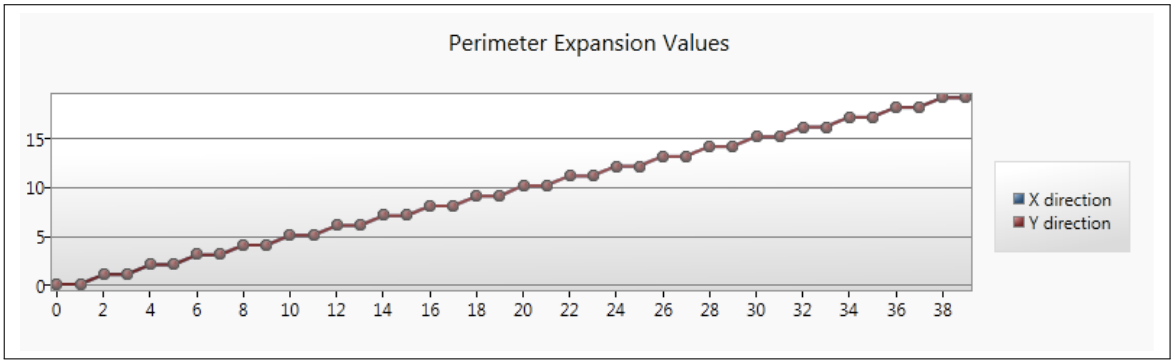

Figure 5.9: Expansion values for increasing expansion speed

High correlation

The graphical output for all models looks exactly the same, the Y axis values just differ by fixed factor for every algorithm. Figure 5.10 shows the output for the **Simple** algorithm.

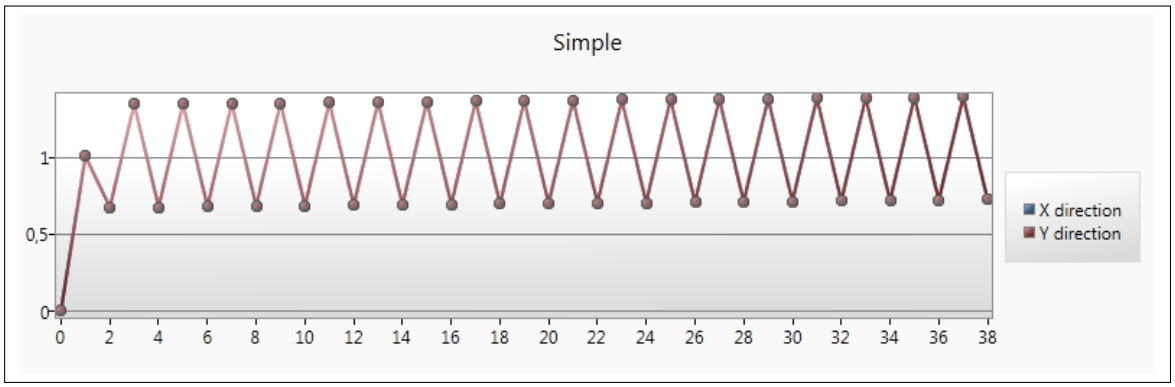

Figure 5.10: Visual output for the **Simple** algorithm - Increasing expansion - High correlation

Table 5.5 and 5.6 show the simulation results.

Table 5.5: Increasing expansion- High correlation - Differences

Constant Expansion - High Cor	Mean Diff X	Mean Diff Y	Max Diff X	Max Diff Y
Simple	0.9907	0.9907	1.3844	1.3844
Simple Attenuation	0.9885	0.9885	0.8853	0.8853
Simple Neighbor	0.3699	0.3699	0.5192	0.5192
Neighbor	0.9673	0.9673	1.3360	1.3360
Neighbor No Common Distribution	0.9711	0.9711	1.3459	1.3459

Table 5.6: Increasing expansion- High correlation - Standard Deviation and Time

Constant Expansion - High Cor	StdDev X	StdDev Y	Total Time in ms	Step Time in ms
Simple	1.2277	1.2277	1849	47
Simple Attenuation	0.8853	0.8853	1919	48
Simple Neighbor	0.8846	0.8846	1930	49
Neighbor	0.9101	0.9101	2535	64
Neighbor No Common Distribution	0.9641	0.9641	2732	69

Interesting for these results is how good the **Simple Neighbor** algorithm performed. This means it should be possible to get better results with all other neighbor algorithms as well, but the ones currently used are not able to use the correlation between neighbors as an advantage in the case of constant expansion. Displaying the correlation between neighboring points that has been used by the **Neighbor** algorithm shows that only part of the distribution has been assumed to be shared. This can be seen in figure 5.11. To improve the algorithm it would be necessary to maximize this shared distribution, which might require making changes to the inference engine.

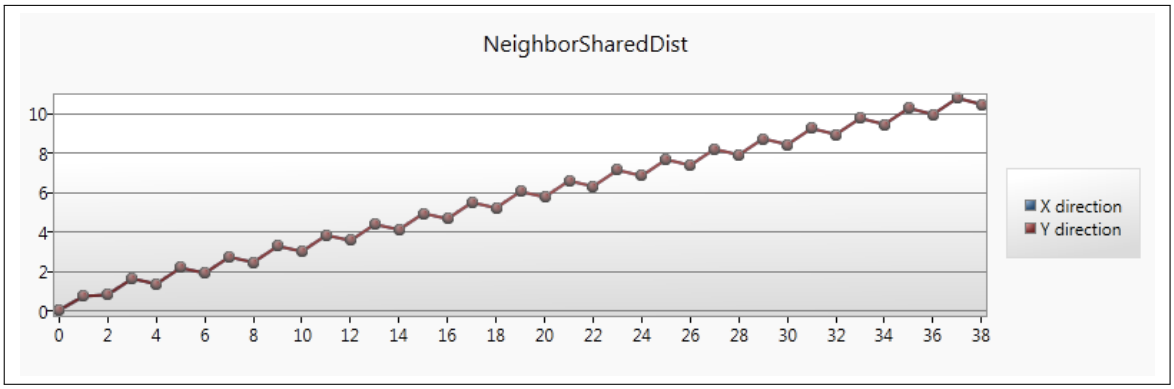

Figure 5.11: Visual output for the correlation between neighbors assumed by the **Neighbor** algorithm for high correlation

Lower correlation

Figure 5.12 shows the difference in expansion speed for the neighbors with increasing expansion speed. In comparison with the expansion of the points in figure 5.7 the difference to the current point is not as high, which means the correlation between neighboring points is higher than in that data set.

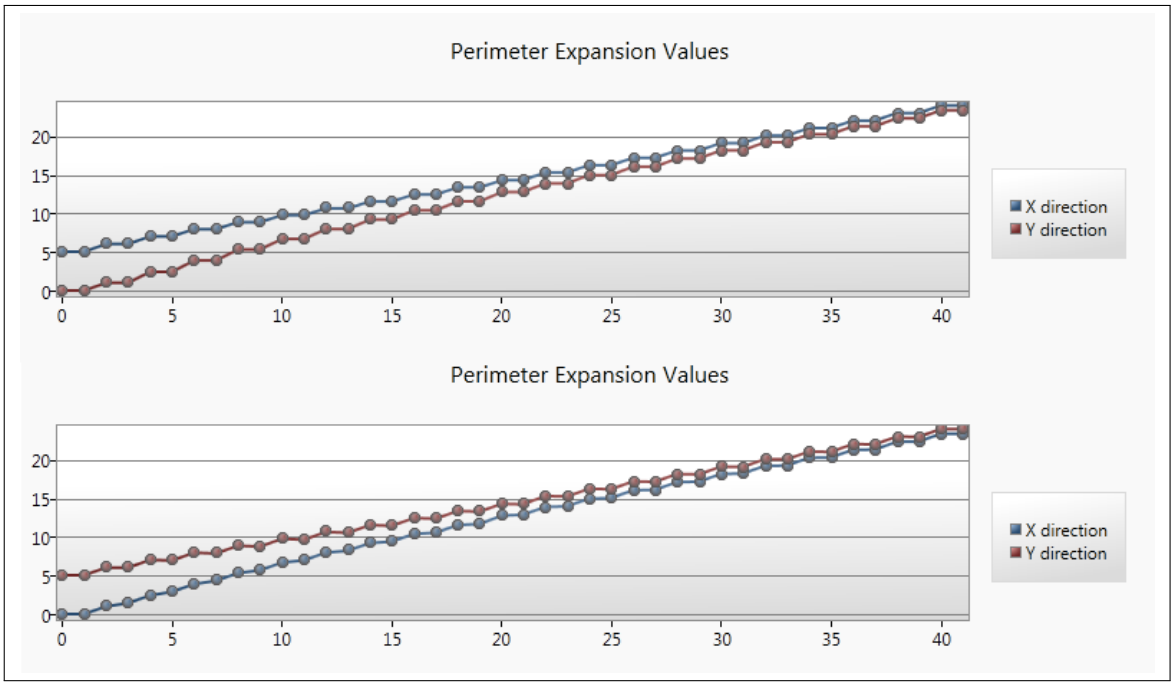

Figure 5.12: Expansion values for both neighbors for increasing expansion

The graphical output for almost all algorithms is close to the one shown in 5.10 again. The output for the **Simple Neighbor** algorithm, however, differs. It performs by nearly factor two worse than all other algorithms in the beginning, this is due to the lower correlation in this data. The correlation increases from time step to time step and the predictions made by the **Simple Neighbor** algorithm increase accordingly. The output can be seen in 5.13.

Table 5.7 and 5.8 show the simulation results for all algorithms.

Notable about these results is only the high maximum difference for the **Simple Neighbor** algorithm,

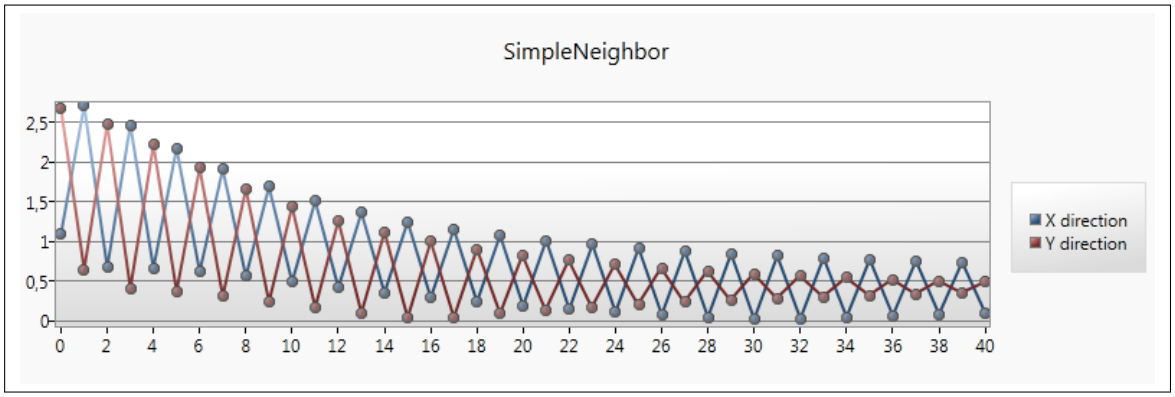

Figure 5.13: Visual output for the **Simple Neighbor** algorithm - Increasing expansion - Lower correlation

Table 5.7: Increasing expansion- Lower correlation - Differences

Constant Expansion - Lower Cor	Mean Diff X	Mean Diff Y	Max Diff X	Max Diff Y
Simple	1.0053	1.0053	1.3974	1.3974
Simple Attenuation	1.0014	1.0014	1.3908	1.3908
Simple Neighbor	0.7700	0.6817	2.6965	2.6606
Neighbor	0.9705	0.9703	1.3368	1.3367
Neighbor No Common Distribution	0.9770	0.9751	1.3497	1.3489

Table 5.8: Increasing expansion- Lower correlation - Standard Deviation and Time

Constant Expansion - Lower Cor	StdDev X	StdDev Y	Total Time in ms	Step Time in ms
Simple	1.2271	1.2271	2037	49
Simple Attenuation	0.8853	0.8853	2115	51
Simple Neighbor	0.9154	0.9887	2042	49
Neighbor	0.9100	0.9103	2543	61
Neighbor No Common Distribution	0.9645	0.9649	2937	71

the explanation for this is the same as for the graphical output.

5.4.4 Change in every second step

The next set of test data consists of continuous changes in the expansion speed, every second step the expansion speed changes, in the beginning it increases, after a while it decreases and increases again a few time steps later. The visual representation of this input data is shown in figure 5.14.

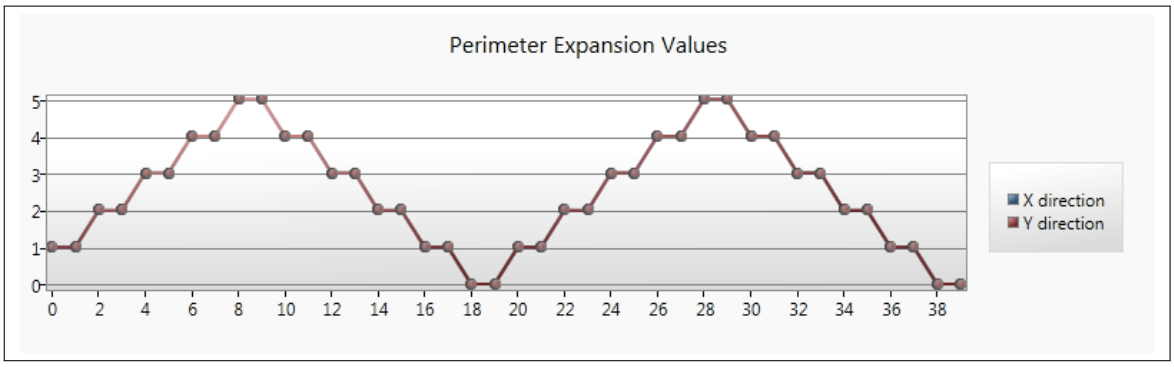

Figure 5.14: Expansion values for change in every second step

High correlation

Again the visual output is almost the same for all algorithms, the only noticeable difference being that the **Simple Neighbor** algorithm performs by more or less factor 2 better than every other algorithm. Figure 5.15 shows the output for the **Simple** algorithm.

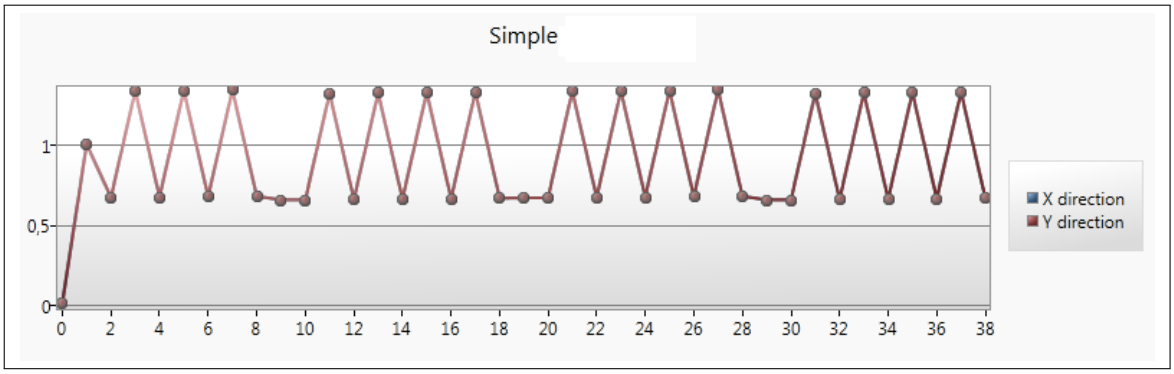

Figure 5.15: Visual output for the **Simple** algorithm - Change every second step - High correlation

Table 5.9 and 5.10 show the simulation results for all algorithms.

As with the graphical output the only notable difference between the algorithms is the very good performance of the **Simple Neighbor** algorithm due to the high correlation.

Lower correlation

Figure 5.16 shows the difference in expansion speed for the neighbors with increasing expansion speed. Again the correlation is not as low as the one in figure 5.7.

The main difference between the algorithms can, yet again, be found in the performance of the **Simple Neighbor** algorithm. While the graphical output for the other algorithms is comparable to that

Table 5.9: Increasing expansion- High correlation - Differences

Constant Change- High Cor	Mean Diff X	Mean Diff Y	Max Diff X	Max Diff Y
Simple	0.9136	0.9136	1.3447	1.3447
Simple Attenuation	0.9136	0.9136	1.3428	1.3428
Simple Neighbor	0.3417	0.3417	0.5043	0.5043
Neighbor	0.9142	0.9142	1.3336	1.3336
Neighbor No Common Distribution	0.9138	0.9138	1.3355	1.3355

Table 5.10: Increasing expansion- High correlation - Standard Deviation and Time

Constant Change - High Cor	StdDev X	StdDev Y	Total Time in ms	Step Time in ms
Simple	1.2273	1.2273	1853	47
Simple Attenuation	0.8851	0.8851	1930	49
Simple Neighbor	0.8783	0.8783	1935	49
Neighbor	0.9100	0.9100	2332	59
Neighbor No Common Distribution	0.9639	0.9639	3337	85

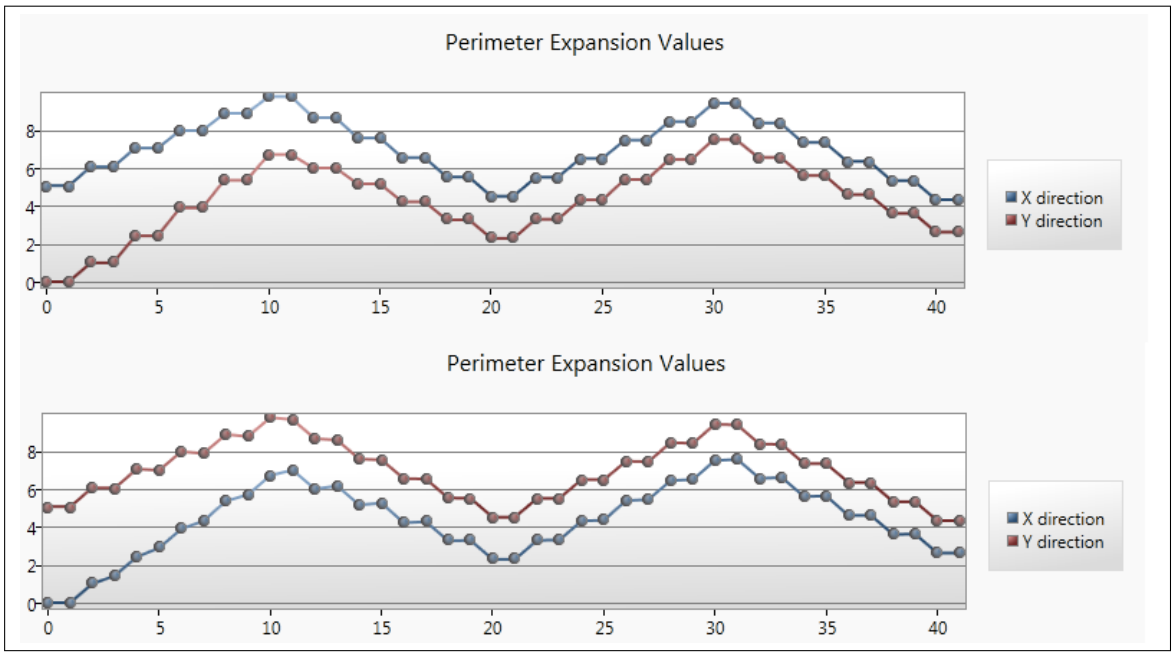

Figure 5.16: Expansion values for both neighbors for constant change

in figure 5.15, the one for **Simple Neighbor** differs an a way similar to the difference in figure 5.13, again due to the same explanation, the lower correlation. The output can be seen in figure 5.17.

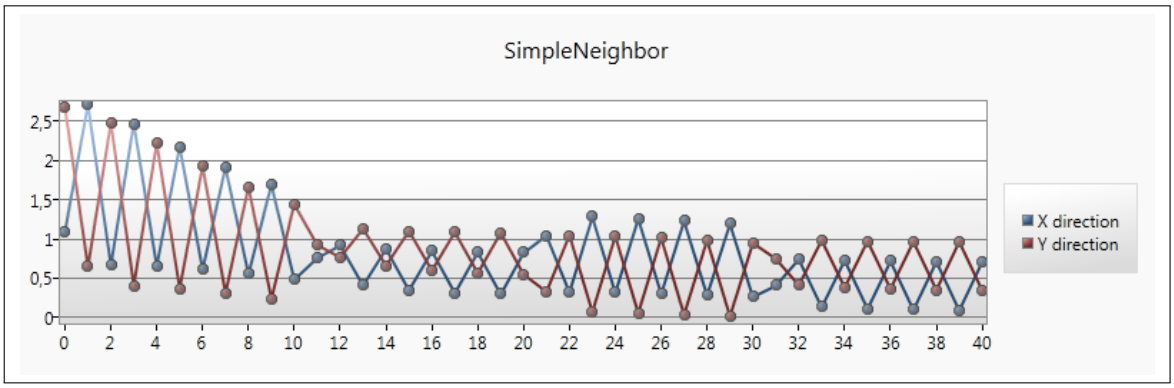

Figure 5.17: Visual output for the **Simple Neighbor** algorithm - Change every second step - Low correlation

Table 5.11 and 5.12 show the simulation results for all algorithms.

Table 5.11: Increasing expansion- Lower correlation - Differences

Constant Change- Low Cor	Mean Diff X	Mean Diff Y	Max Diff X	Max Diff Y
Simple	0.9191	0.9191	1.3547	1.3547
Simple Attenuation	0.9185	0.9185	1.3519	1.3519
Simple Neighbor	0.8051	0.8365	2.6965	2.6606
Neighbor	0.9188	0.9187	1.3349	1.3355
Neighbor No Common Distribution	0.9188	0.9182	1.3411	1.3376

Table 5.12: Increasing expansion- Lower correlation - Standard Deviation and Time

Constant Change - Low Cor	StdDev X	StdDev Y	Total Time in ms	Step Time in ms
Simple	1.2265	1.2265	1868	45
Simple Attenuation	0.8850	0.8850	2102	50
Simple Neighbor	0.9621	0.9903	2128	51
Neighbor	0.9099	0.9099	2626	63
Neighbor No Common Distribution	0.9641	0.9641	2902	70

Since again the only notable difference can be noticed for the **Simple Neighbor** algorithm with the same explanation other test cases yielding similar results will not be further presented or discussed here. They can, however, be reproduced using the simulation program by loading the different pre-stored data sets and the desired algorithms.

5.4.5 Realistic model

The perimeter expansion in this section is designed as similar as possible to real fire behavior by using a manual reconstruction of a fire expansion simulated using a more sophisticated fire simulation program.

The expansion values for three points on the perimeter can be seen in 5.18. The expansion changes every few time-steps and stays the same after that, there is some randomness in the expansion values as well as some correlation between neighboring points.

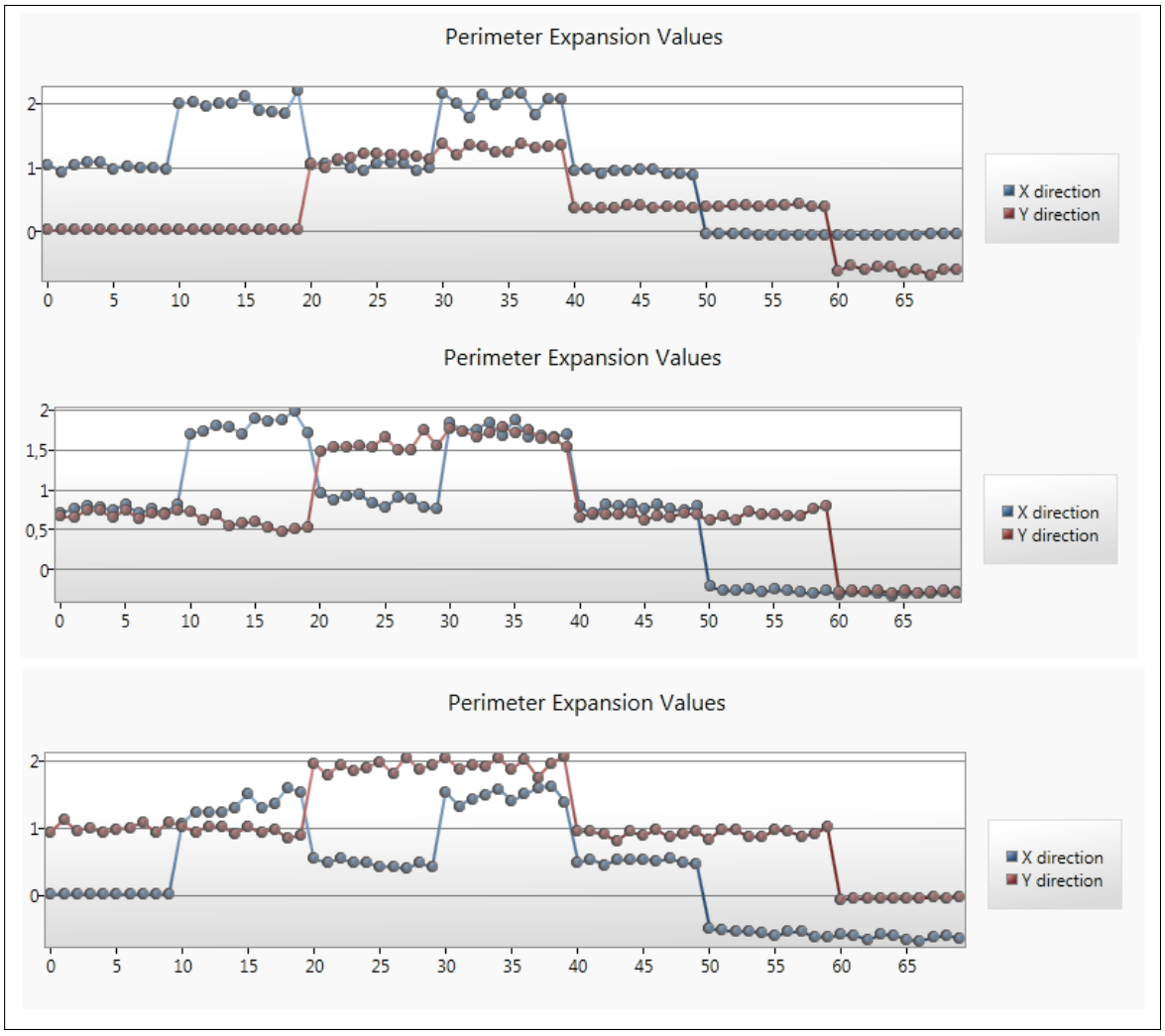

Figure 5.18: Expansion values for three perimeter points for a realistic expansion model

The visual output for this data-set shows goes in the same direction as the other experiments, the results for most algorithms are similar to the one for the **Simple** algorithm shown in figure 5.19. **Simple Neighbor** differs and offers a better overall performance as can be seen in figure 5.20.

The general performance of all algorithms can be seen in table 5.13 and 5.14.

5.4.6 Very low correlation

The last model is one featuring a very low correlation between neighboring points with a constant expansion. The neighboring points are either only expanding into X or into Y direction, while the current point expands into both in same parts. The expansion values can be found in figure 5.21.

The graphical output for most the **Simple** algorithm, which is very similar to that of most other algorithms can be seen in figure 5.22. The one for the **Simple Neighbor** algorithm which differs due to the low correlation is shown in figure 5.23.

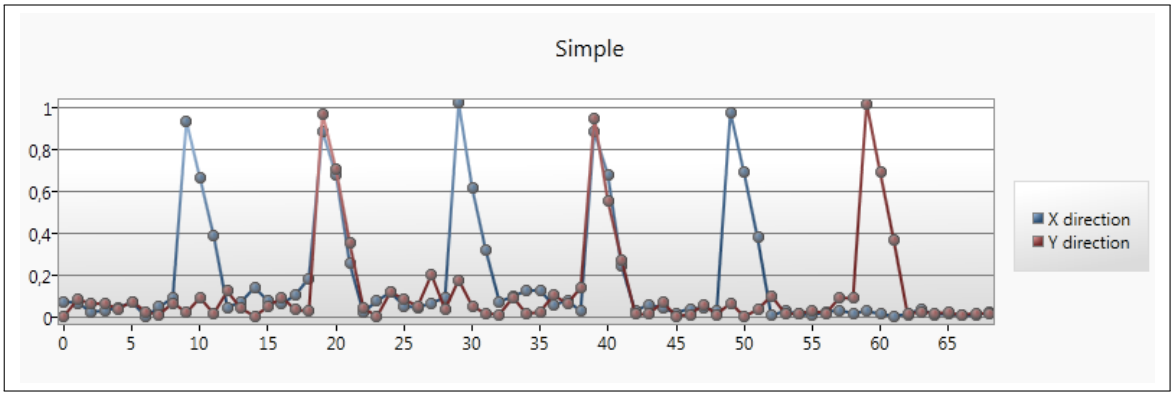

Figure 5.19: Visual output for the **Simple** algorithm - Realistic model

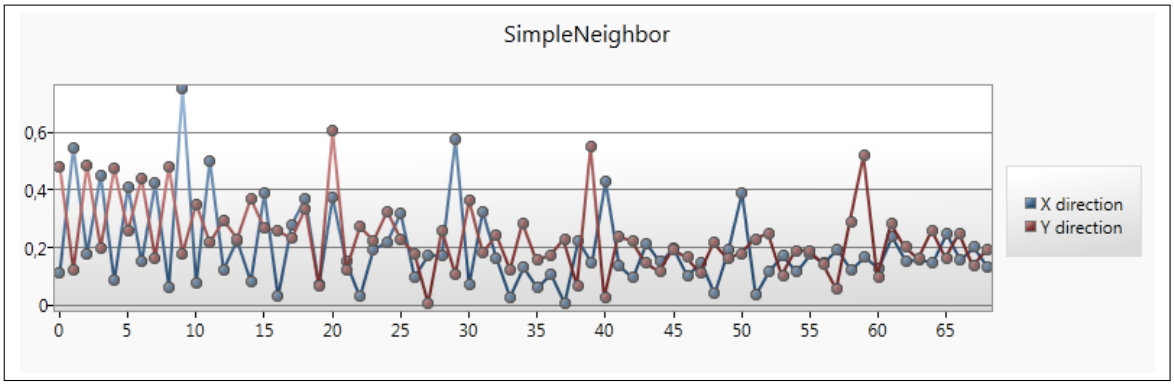

Figure 5.20: Visual output for the **Simple Neighbor** algorithm - Realistic model

Table 5.13: Realistic - Differences

Realistic	Mean Diff X	Mean Diff Y	Max Diff X	Max Diff Y
Simple	0.1776	0.1262	1.0195	1.0165
Simple Attenuation	0.1775	0.1262	1.0193	1.0167
Simple Neighbor	0.1962	0.2292	0.7471	0.6000
Neighbor	0.1774	0.1265	1.0176	1.0179
Neighbor No Common Distribution	0.1773	0.1264	1.0181	1.0176

Table 5.14: Realistic - Standard Deviation and Time

Realistic	StdDev X	StdDev Y	Total Time in ms	Step Time in ms
Simple	1.1258	1.1206	2709	38
Simple Attenuation	0.8292	0.8261	2856	40
Simple Neighbor	0.7818	0.7785	2724	38
Neighbor	0.8512	0.8479	3118	44
Neighbor No Common Distribution	0.8996	0.8959	3443	49

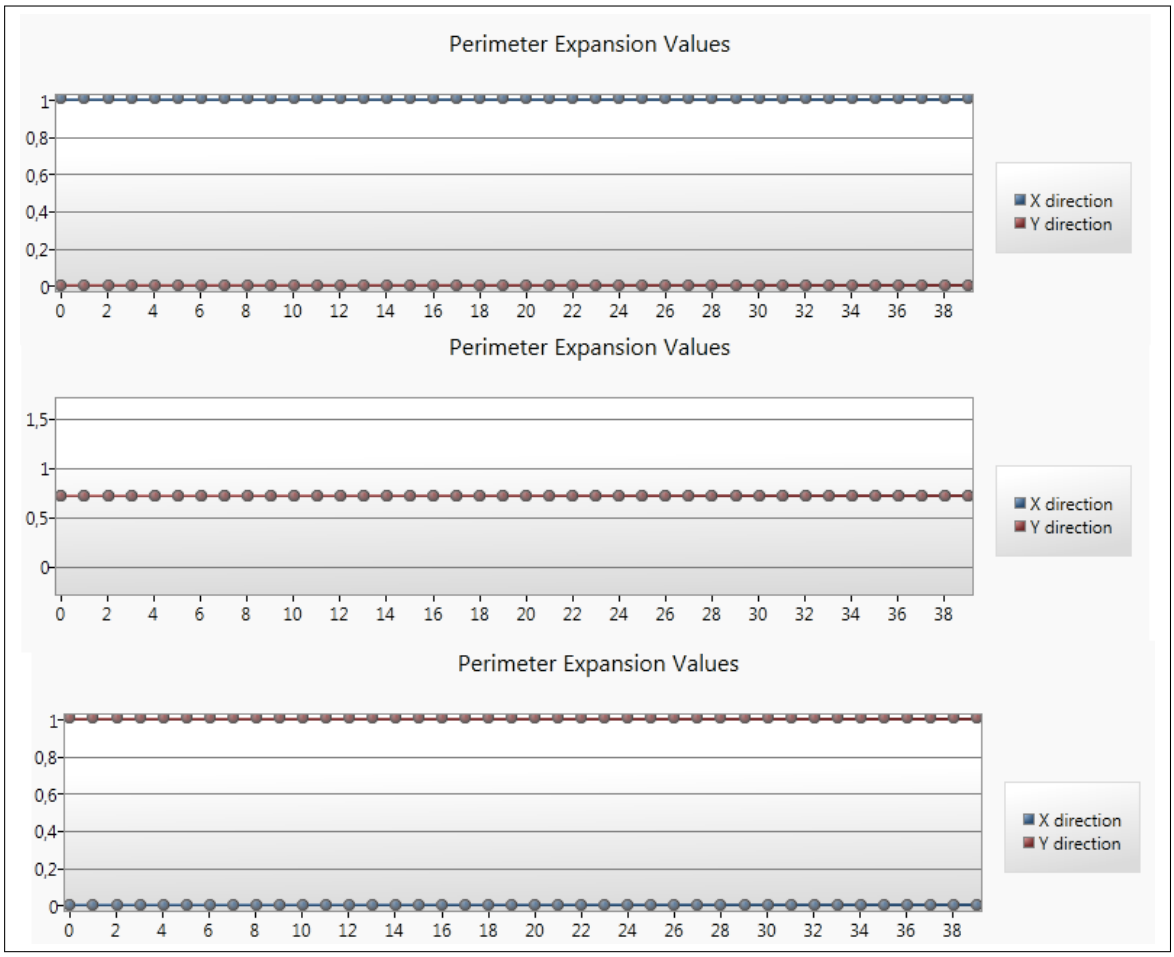

Figure 5.21: Expansion values for three perimeter points for a model with low correlation between neighboring points

The second algorithm with small differences is the **Neighbor No Common Distribution** algorithm, which can be seen in figureOutput11. This algorithms output differs in the beginning, because the inference engine does not have enough observations yet to infer the different distributions the expansion is made up of. Its predictions for X and Y direction stay different, even for a higher number of observed values.

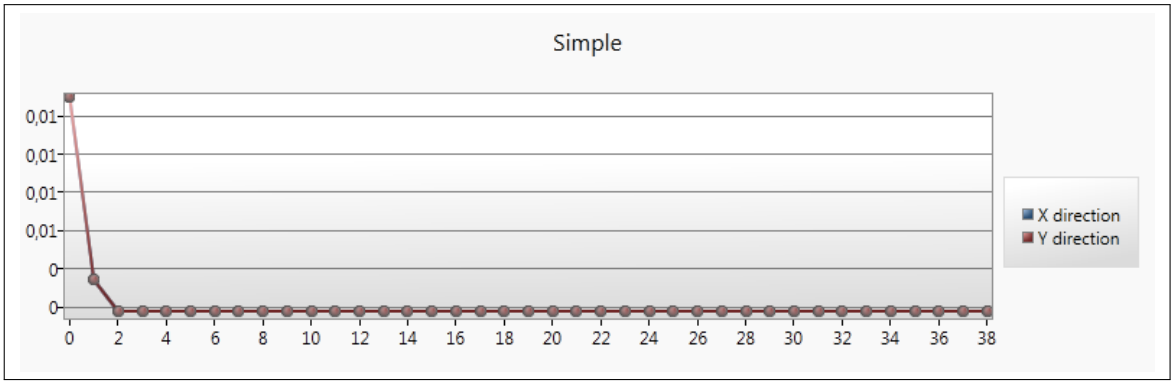

Figure 5.22: Visual output for the **Simple** algorithm - Low correlation

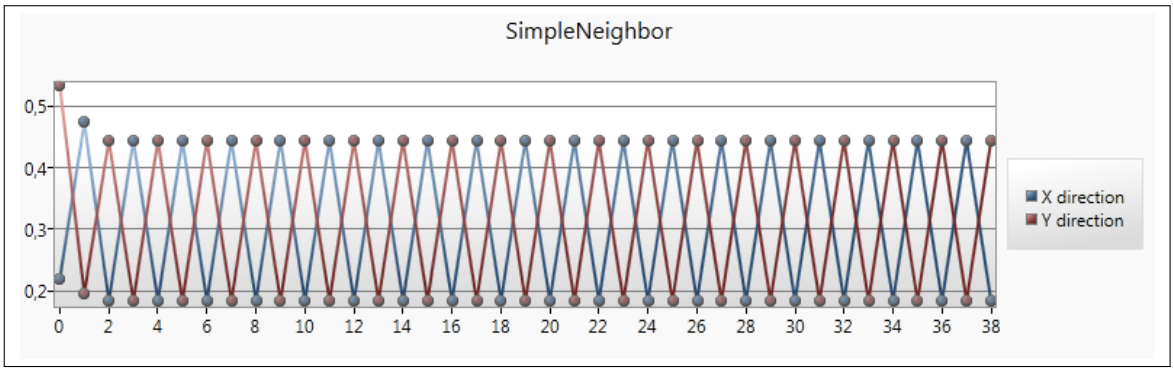

Figure 5.23: Visual output for the **Simple Neighbor** algorithm - Low correlation

The general performance of all algorithms can be seen in table 5.15 and 5.16. Most notable about those is the better performance of the more sophisticated neighbor algorithms in comparison to the simple ones. Even though the correlation is low for at least one of the directions the neighbor algorithms get $2-4$ times better results. The **Simple Neighbor** algorithm does not perform very well, because it is based on the assumption of a high correlation between the expansion of neighboring points.

Table 5.15: Low Correlation - Differences

Low Correlation	Mean Diff X	Mean Diff Y	Max Diff X	Max Diff Y
Simple	0.0020	0.0020	0.0129	0.0129
Simple Attenuation	0.0017	0.0017	0.0070	0.0070
Simple Neighbor	0.3108	0.3183	0.4720	0.5311
Neighbor	0.0005	0.0005	0.0026	0.0024
Neighbor No Common Distribution	0.0009	0.0005	0.0015	0.0024

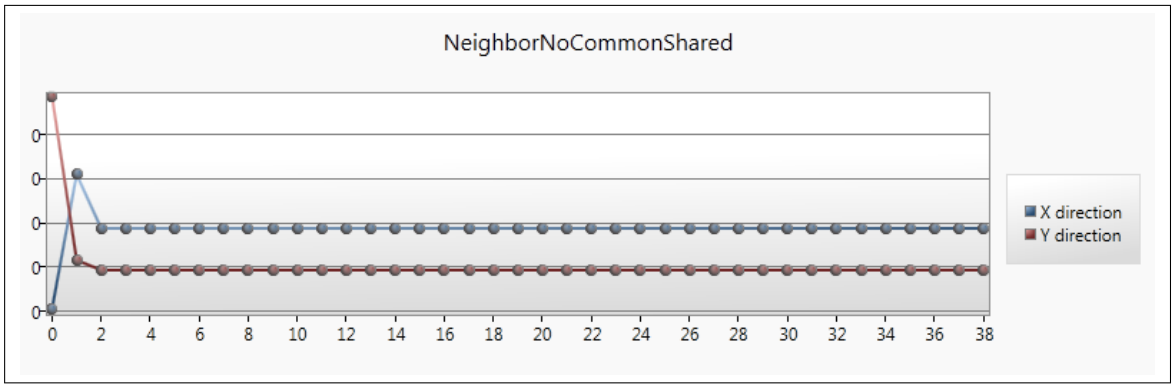

Figure 5.24: Visual output for the **Neighbor No Common Distribution** algorithm - Low correlation

Table 5.16: Low Correlation - Standard Deviation and Time

Low Correlation	StdDev X	StdDev Y	Total Time in ms	Step Time in ms
Simple	1.1227	1.1227	1005	25
Simple Attenuation	0.8230	0.8230	1122	28
Simple Neighbor	0.7809	0.7815	1819	45
Neighbor	0.8450	0.8450	1804	46
Neighbor No Common Distribution	0.8930	0.8930	2102	53

5.4.7 Conclusion

After simulating the performance of the proposed algorithms with different test data sets, which cover all possible scenarios the overall performance of each algorithm can be compared. For the overall performance it is necessary to note that these algorithms can not reliably predict change in expansion behaviour. The algorithms do not have any knowledge of influencing factors like wind or steepness. Sudden change in the expansion speed therefore always results in a high error for the first points measured after this change. For the neighboring points the changed expansion speed can be taken into account. It is evident that the **Neighbor** algorithm offers the best performance in most cases. It is beaten by the **Neighbor No Common Shared** algorithm in some cases, but only by a less than 0.1% in the cases with a lower correlation between neighbors. Even in cases with absolutely no correlation the performance of the **Neighbor No Common Shared** algorithm only beats the **Neighbor** algorithm by less than 1%. It takes, however, around 7ms longer to infer a prediction after a new measurement, a time that would be higher when executed on a microprocessor instead of the computer used for the simulations. Additionally it performs worse for cases with a higher correlation. Since a high correlation between neighbors can be assumed the **Neighbor** algorithm should be superior in all realistic use cases. The **Simple Neighbor** algorithm performs superior for use cases with a high correlation between neighbors while requiring a shorter inference time. It is, however, very vulnerable to cases with a low correlation between neighbors. Cases that can always happen in reality, for example if there is a difference in steepness or a lake or river blocking the expansion of the fire at some perimeter points. Nevertheless the algorithm is a good indicator for possible improvements on the **Neighbor** algorithm. Some more work is required to make better use of the correlation between neighbors, but until then

the **Neighbor** algorithm is the best choice. Further testing is required to see whether the algorithms can be ported to work on a microprocessor and to assess how much time inference takes in such an environment. Otherwise it is necessary to revise the originally proposed hardware architecture by adding an additional processing unit with the required support and computational power.

6 Conclusion

During this thesis an agent architecture has been presented that allows for the deployment of UAVs for autonomous fire surveillance. There is, however, additional work required, especially in order to verify some of the proposed algorithms. The conclusions made in this chapter are general conclusions as well as explanations about future work required. Each chapter offers its own conclusion chapter, which goes into more detail.

First of all chapter 2 discusses the necessary hardware required to enable UAVs to observe fires and make decisions based on their expansion. Most important in the is the ability to sense fire. This can be done easiest by using an infrared camera. Furthermore the UAVs need to be able to communicate with each other which can be facilitated by using n-standard WLAN. This fast communication allows the UAVs to transmit all necessary information whenever they meet. As future work satellite communication for single or all of the UAVs could be considered. Satellite communication is usually slow for portable terminal, highest typical speed is around 492 kbit/s. Since the antenna of the satellite terminal should point in the general direction of the satellite it would have to be automatically adjusted to movement of the UAV to enable a reliable communication. If tests, however, proof that this is possible, information from wildfires could be directly reported back to the crisis manager. Either by one UAV equipped with satellite communication that collects all available information from the other UAVs, or from each UAV directly. Since the equipment is expensive and a communication channel to a satellite in one area is usually shared, one UAV equipped with a satellite terminal would be the most sensible choice. This UAV could even be shared between multiple wildfires, flying to each one in turn and transmitting the information. The UAVs used for fire surveillance require the ability to stay airborne for a considerable time in order to be viable, which, for the time being, disqualifies most battery powered solutions.

Chapter 3 reviews algorithms and discusses their drawbacks. None of the reviewed algorithms can be used for fire surveillance as required for the proposed use case. This is due to assumptions being made in the original research articles that are not viable in this thesis. Either they require a permanent communication between UAVs or they assume fixed endpoints for the perimeter that should be observed. Neither of this holds true for the cases discussed in this thesis, because the assumptions discussed in 2 include a moving fire, which does not have a start or endpoint as well as communication that is limited to UAVs that are in very close proximity to each other. To overcome this problem another algorithm has been proposed. This solutions is theoretically viable for a distributed perimeter surveillance of a moving fire. It is, however, necessary to verify this assumption, ideally using real UAVs, but at least by using simulations. Possible implementation platforms for this testing would be the Unreal Tournament 3 based 3D Simulator proposed in [Sie11] or the formation control software proposed in [Ins12]. If the

algorithm should prove inefficient or unusable the algorithms discussed in the review section could be made viable by using one stationary UAV as perimeter endpoint.

Chapter 4 proposes a way to describe a fire perimeter using only a few points. While there are published algorithms available to do this and the additions made in this chapter are theoretically solid they need to be tested in a real environment. The same is valid for the fire prediction algorithms proposed in this chapter and implemented in chapter 5. They have been tested and compared using a simulator. Nevertheless, the input data is not directly taken from real wildfires. The simulations need to be repeated with either real data or data taken from professional fire simulation programs.

One problem arising in a real environment could be the positioning of the UAV in relation to the fire. Because the fire is sensed by using cameras an exact positioning is not possible. If the position of the UAV in relation to the fire would change with each pass along it, even if the fire perimeter did not change this could result in significant errors in the prediction algorithms. In order to overcome this problem this variability in the UAVs position needs to be small in comparison the the expansion speed of the fire. For a fire that is not expanding or only expanding slowly the predictions would be mostly based on the random positioning of the UAV along the perimeter and therefore be random themselves. If the distance from the UAV to the fire perimeter increases multiple times, this would result in a higher predicted expansion. The influence of this effect needs to be evaluated.

Additionally there is still room for improvement for the proposed algorithms as discussed in the conclusion subsection 5.4.7. Even with this improvement the prediction is simply based on the historical expansion data of the fire. In order to make more accurate predictions of future fire behavior it is necessary to include additional information into the prediction algorithms. In order to use additional information for the prediction algorithms, such as weather forecasts or maps of the region, and thereby making DBN prediction algorithms viable, it is necessary to implement the model including additional data from subsection 4.3.5. This model than needs to be tested with real input data, which requires expansion data of a wildfire in combination with weather forecasts for the duration of the fire as well as well as measurements of the wind speed and direction at several points along the fire perimeter. A possibility to test the models including additional information would be to use the wildfire flame mapper [BS], which uses the Google maps API in combination with different fire expansion models to display historical, as well as current wildfires in combination with their expansion speeds and wind direction. The data, however is not directly available, but only displayed on an interactive map. It is necessary to contact the developers of this tool, or any other partner with access to this kind of information in order to adequately implement and test the advanced prediction models.

Additionally it would be possible to extend the stochastical models in order to use pre-known expansion factors such as wind or the surrounding area. In order to do this one additional influencing factor would be added into the models and, by using the observed value method, set to the result of the pre-known factors of the expansion speed and direction.

In conclusion this thesis provided the basis to enable the deployment of UAVs for fire surveillance, additional work and testing, however, as discussed in this chapter, is still required before a functional prototype can be constructed.

List of Figures

Bibliography

[ABU03] ANDREI B. UTKIN, Fernando Simoes Alexander Lavrov Rui V. Armando Fernandes F. Armando Fernandes: Feasibility of forest-fire smoke detection using lidar. In: *International Journal of Wildland Fire* 12 (2003), S. 159–166

[al.05] AL., Grejner-Brzezinska et: On Improving Navigation Accuracy of GPS/INS Systems. In: *Photogrammetric Engineering & Remote Sensing* 71 (2005), S. 377–389

[APK07] A. P. K.G.TOLHURST, D.M. C.: *PHOENIX a dynamic fire characterization simulation tool*. http://people.stat.sfu.ca/~dean/forestry/updates/dokuwiki-2011-05-25a/data/media/talks/nicds-tolhurst.pdf. Version: 2007

[Arm] ARMY, US: *US Army UAS RoadMap 2010 - 2035*. http://www-rucker.army.mil/usaace/uas/USArmyUASRoadMap20102035.pdf

[BS] http://www.flamemapper.com/about-project.php

[Cam] CAMBRIDGE, Microsoft R.: *An Introduction to Infer.NET A look at probabilistic programming and Microsoft Infer.NET*. http://research.microsoft.com/en-us/um/cambridge/projects/infernet/docs/InferNet_Intro.pdf

[Cam11] CAMBRIDGE, Microsoft R.: *Infer.net Documentation*. http://research.microsoft.com/en-us/um/cambridge/projects/infernet/codedoc/Index.html. Version: 2011

[Cen] CENTER, National Interagency C.: *Wildland Fire Summary and Statistics Annual Report 2011*. http://www.predictiveservices.nifc.gov/intelligence/2011_statssumm/fires_acres.pdf

[(CI12] (CIFFC), Canadian Interagency Forest Fire C.: *Prometheus*. http://www.firegrowthmodel.com/. Version: January 2012

[Coma] COMMONS, Wikimedia: *Gamma Distribution*. http://upload.wikimedia.org/wikipedia/commons/e/e6/Gamma_distribution_pdf.svg

[Comb] COMMONS, Wikimedia: *Normal Distribution*. http://upload.wikimedia.org/wikipedia/commons/7/74/Normal_Distribution_PDF.svg

[DKH07] DEREK KINGSTON, Randal B. ; HOLT, Ryan: Decentralized perimeter surveillance using a team of small UAVs. In: *Robotics, IEEE Transactions on* 24 (September 2007), S. 1394 – 1404

[Dro] *CIA Drone Strikes in Pakistan 2004 2012*. http://www.thebureauinvestigates.com/category/projects/drones/

[DWCM06] DAVID W. CASBEER, Randal W. Beard Timothy W. McLain Sai-Ming L. Derek B. Kingston K. Derek B. Kingston ; MEHRA, Raman: Cooperative forest fire surveillance using a team of small unmanned air vehicles. In: *International Journal of Systems Sciences* 37 (May 2006), S. 351360

[FB09] FRANCESCO BULLO, Sonia M. Jorge Cortes C. Jorge Cortes ; BULLO, Francesco (Hrsg.): *Distributed Control of Robotic Networks*. Princeton Press, 2009

[Fin98] FINNEY, M. A.: *Farsite: Fire area simulator model development and evaluation*. http://www.landsinfo.org/ecosystem_defense/federal_agencies/forest_service/Region_1/Idaho_Panhandle_NF/Bonners_Ferry_District/MyrtleHFRA/MyrtleCreekHFRAObjectionreferencesdisk4/fireareaall.pdf. Version: 1998. – res. Pap. RMRS-RP-4 (revised)

[Gat98] GAT, E ; KORTENKAMP, Bonasso R. D. (Hrsg.) ; MURPHY, R. (Hrsg.): *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*. AAAI Press, Menlo Park, 1998

[Gem84] GEMAN, Stuart: Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6 (1984), S. 721–741

[hel] *UAV Helicopter*. http://upload.wikimedia.org/wikipedia/commons/6/63/MQ-8B_Fire_Scout.jpeg

[HR08] HUQ R, Gosine R. Mann GKI G. Mann GKI: Mobile robot navigation using motors schema and fuzzy context dependent behaviour modulation. In: *Appl. Soft. Comput.* 8 (2008), S. 422–436

[HS] HONGXING SUN, Xiuxiao Yuan Weiming T. Jianhong Fu F. Jianhong Fu: *Analysis of the kalman filter with different INS error models for GPS/INS integration in aerial remote sensing applications*. http://www.isprs.org/proceedings/XXXVII/congress/5_pdf/154.pdf

[Ins12] INSELMANN, Bastian: *Entwicklung einer Software-Basis für die Formationsregelung eines Multi-Agenten Systems am Beispiel von Quadrocoptern*, TU Hamburg-Harburg, Diplomarbeit, 2012

[JRH] JOHN R. HALL, Jr.: *The total cost of fire in the United States*. http://www.nfpa.org/assets/files/PDF/totalcostsum.pdf. – National Fire Protection Association Fire Analysis and Research Division

[M.D99] M.D.FLANNIGAN, B.M. W. B.J. Stocks S. B.J. Stocks: *Climate Change and Forest Fires*. http://www.environmentportal.in/files/cc-SciTotEnvi-2000.pdf. Version: 1999

[Mi11] MI: *Infer.net User Guide*. http://research.microsoft.com/en-us/um/cambridge/projects/infernet/docs/default.aspx. Version: 2011

[Min01] MINKA, Thomas P.: *A family of algorithms for approximate Bayesian inference*, Massachusetts Institute of Technology, Diss., 2001

[ML98] MCKINLEY, Sky ; LEVINE, Megan: *Cubic Spline Interpolation*. http://web.archive.org/web/20090408054627/http://online.redwoods.cc.ca.us/instruct/darnold/laproj/Fall98/SkyMeg/Proj.PDF. Version: 1998

[MLB06] MARBACH, Giuseppe ; LOEPFE, Markus ; BRUPBACHER, Thomas: An image processing technique for fire detection in video images. In: *Fire Safety Journal* 41 (2006), Nr. 4, 285 - 289. http://dx.doi.org/10.1016/j.firesaf.2006.02.001. – DOI 10.1016/j.firesaf.2006.02.001. – ISSN 0379–7112. – <ce:title>13th International Conference on Automatic Fire Detection, Duisburg, Germany</ce:title> <ce:subtitle>AUBE '04</ce:subtitle> <xocs:full-name>13th International Conference on Automatic Fire Detection, Duisburg, Germany</xocs:full-name>

[Mur02] MURPHY, Kevin: *Dynamic Bayesian Networks: Representation, Inference and Learning*, UC Berkeley, Computer Science Division, Diss., Juli 2002

[MWGK10] MINKA, T. ; WINN, J.M. ; GUIVER, J.P. ; KNOWLES, D.A.: *Infer.NET 2.4*. 2010. – Microsoft Research Cambridge. http://research.microsoft.com/infernet

[New04] NEWCOME, Laurence R. ; (Hrsg.): *Unmanned Aviation: A Brief History of Unmanned Aerial Vehicles*. American Institute of Aeronautics and Astronautics, 2004

[Nga12] NGATCHA, Eric Stephen N.: *Bildverarbeitung für Waldbranderkennung*, TU Hamburg-Harburg, Diplomarbeit, 2012

[PB09] POWERS, Matthew ; BALCH, Tucker R.: A learning approach to integration of layers of a hybrid control architecture. In: *IROS*, 2009, S. 893–898

[Pea88] PEARL, J. ; KAUFMANN, Morgan (Hrsg.): *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988

[pio] *Pioneer Drone*. http://upload.wikimedia.org/wikipedia/commons/4/4d/Pioneer_Unmanned_Aerial_Vehicle.jpg

[PLA98] PATRICIA L. ANDREWS, Collin D.: *BEHAVE Fire Modeling System-Redesign and Expansion*. http://maps.wildrockies.org/ecosystem_defense/Federal_

Agencies/Forest_Service/Region_1/Idaho_Panhandle_NF/Bonners_Ferry_District/MyrtleHFRA/MyrtleCreekHFRAObjectionreferencesdisk4/behave.pdf. Version: 1998

[qua] *Aeryon Scout picture*. http://upload.wikimedia.org/wikipedia/commons/a/a6/Aeryon_Scout_With_Camera.jpg

[R86] R, Brooks: A robust layered control system for a mobile robot. In: *EEE J. Robot Autom.* 2 (1986), S. 14–23

[RC89] RC, Arkin: Motor schema-based mobile robot navigation. In: *Int. J. Robot Autom.* 8 (1989), S. 92–112

[RC98] RC, Arkin ; RC, Arkin (Hrsg.): *Behavior- Based robotics*. The MIT press, 1998

[Sie11] SIEGEMUND, Gerry: *AURIS Autonomous Robot Interaction Simulation*, TU Hamburg-Harburg, Diplomarbeit, Januar 2011

[SJ83] SCHWARTZ JT, Sharir M.: On the piano movers problem. General techniques for computing topological properties of real algebraic manifolds. In: *Adv. Appl. Math.* 4 (1983), S. 298–351

[Weia] WEISSTEIN, Eric W.: *"Central Limit Theorem."*. http://mathworld.wolfram.com/CentralLimitTheorem.html. – From MathWorld–A Wolfram Web Resource.

[Weib] WEISSTEIN, Eric W.: *"Gamma Distribution"*. http://mathworld.wolfram.com/GammaDistribution.html. – From MathWorld–A Wolfram Web Resource

[Weic] WEISSTEIN, Eric W.: *"Gamma Function"*. http://mathworld.wolfram.com/GammaFunction.html. – From MathWorld–A Wolfram Web Resource

[Weid] WEISSTEIN, Eric W.: *"Normal Distribution"*. http://mathworld.wolfram.com/NormalDistribution.html. – From MathWorld–A Wolfram Web Resource

[Win03] WINN, John M.: *Variational Message Passing and its Applications*, Cambridge, Diss., 2003