# A Simulation Environment for Autonomous Robot Swarms with Limited Communication Skills

Alexander Puzicha(✉) and Peter Buchholz

Informatik 4, TU Dortmund, Dortmund, Germany
{alexander.puzicha,peter.buchholz}@cs.tu-dortmund.de

**Abstract.** We present a novel real-time simulation tool for modeling and analyzing a swarm of distributed autonomous mobile robots communicating over an unreliable and capacity restricted communication network. The robots are setup as ground vehicles and use C-PBP [8] as model predictive closed loop controller. This tool offers the ability to simulate rural as well as completely urban scenarios with static obstacles, dynamic obstacles with scripted movement, soil condition, noise floor, active jammers and static and dynamic obstacles for the links with adjustable damping. The goal of this simulation is the analysis of swarm behavior of robots for given missions such as terrain exploration, convoy escorting or creation of a mobile ad hoc network in disaster areas under realistic environmental conditions.

**Keywords:** Autonomous robots · Model predictive control · Real-time simulation · Swarm behavior

## 1 Introduction

Swarms of autonomous robots are used for a wide variety of missions during disasters. Typical tasks are the exploration of an area, the escorting of convoys or the set up of MANETs if infrastructure breaks down. The autonomous control of robots is a challenging task in particular under tough environmental conditions that limit the possibilities of the robots to communicate. This implies that each robot has only a local view with limited information about the rest of the swarm and of its environment. Before the robot swarm can be deployed in disaster areas, the control software has to be tested carefully. Unfortunately, field tests are expensive and sometimes almost impossible. Thus, the only viable alternative are simulations. However, the test of software with real-time requirements has to be done in an real-time environment which puts very strict demands on the simulation software. The control software has to be part of the simulator, the dynamics of the robots has to be simulated realistically, environmental conditions have to be described and the mobile communication has to be modeled including limited communication ranges and packet losses. All this should be done in user-friendly way and implemented efficiently to allow the real-time simulation of larger swarms as they are used in practice.


H. Song and D. Jiang (Eds.): SIMUtools 2020, LNICST 370, pp. 206–226, 2021.
https://doi.org/10.1007/978-3-030-72795-6_17

*Related Work:* The presented approach uses model predictive control which has been applied for autonomous robots in several papers [4,18]. However, the control algorithms often work under unrealistic assumptions like the availability of complete information for every robot or the existence of a central instance which computes the trajectories for every member of the swarm [3,16]. Only recently, feedback control algorithms over wireless networks have been published [2,14]. Several other simulators of autonomous robots exist, but only those related to our work are presented in this paper. [12] describe a simulation environment for swarms of robots without a detailed communication model. In [19], a co-simulation approach for communicating autonomous vehicles is introduced. The approach combines the robotic simulator Gazebo and the network simulator OMNeT++. In contrast to our tool, the software combines several tools and is mainly tailored to autonomous driving.

*Contribution of this Paper:* In this paper we present a new modular structured simulation environment for swarms of autonomous robots. The software allows one to describe complex environments in which the robots have to perform their mission. The environment may contain different types of obstacles which hinder the movement of robots and possibly also the communication between robots. A graphical interface shows the movement of the robots in their environment and allows the user to interact with the robots, e.g. by modifying the mission during operation. The simulator performs a detailed simulation of the swarm by exploiting the parallel features of contemporary processors. Some examples show that with the simulation software swarms of a realistic size can be simulated in real time on a modern workstation. The whole approach has been developed in [15].

*Structure of the Paper:* In the following section we describe the basic model of an autonomous robot and the model predictive control algorithm. Furthermore, modeling of the environment using appropriate cost functions and the communication model are introduced. Section 3 describes the simulation software. In Sect. 4 some example runs of the simulator are presented. The paper ends with a short summary and some topics for future research.

## 2 A Model of Autonomous Robots

At first we have to model the autonomous swarm robots. Their key behavior is adapted from people's strategic planning techniques. First, a robot gathers information from the environment, then a plan is created based on the information, mission target, communication possibilities and the prediction of states of the other robots and dynamic objects in world. Due to optimization, this results in a local plan that forms a trajectory in space and time. It is sent to other robots so their plan will be created with respect to this one. The plan leads to an action that causes a change of the environment which in turn produces new information for the sensors. This closed loop can be transformed to a model predictive

controller. Therefore, it has to be split into parallel working threads. One thread tackles the problem of sensor evaluation and object detection, another one serves as communication module to handle and filter messages and data packages from other robots. To maintain the connections and to be aware of signal losses it is important that this module is able to monitor and predict the connection quality (see Sect. 2.3). Furthermore, the actuator has to be represented by a separate thread as well as the planning algorithm.

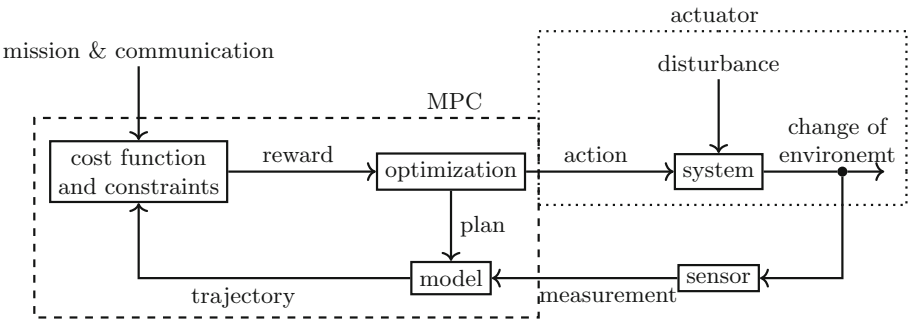


**Fig. 1.** Four components of the robot: communication module, model predictive controller (MPC), actuator and sensor

### 2.1 Autonomous Robots and Their Control

As shown in Fig. 1 the model predictive control (MPC, see [7]) algorithm needs a kinematic model of the system and due to the focus on ground based mobile robots and tracked vehicles, we use a simple discrete, nonlinear model for non-holonomic systems that have the ability to turn on the spot. Ground vehicles have less degrees of freedom than flying drones, but there are many more obstacles on the ground with limited possibilities to avoid those. Based on this model, a trajectory is created by the predicted control values and rated through cost functions and constraints which are presented in the next section. The basic model is time discrete and will be briefly described first.

$$\mathbf{x}_k = f_k(\mathbf{x}_{k-1}, \mathbf{u}_k) + \boldsymbol{\xi} = \underbrace{\begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix}}_{\mathbf{x}_k} = \underbrace{\begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \theta_{k-1} \end{bmatrix}}_{\mathbf{x}_{k-1}} + \begin{bmatrix} v_k \cdot \sin(\theta_{k-1}) \\ v_k \cdot \cos(\theta_{k-1}) \\ \dot{\theta}_k \end{bmatrix} \cdot \Delta k + \boldsymbol{\xi} \tag{1}$$

$$\text{with } \mathbf{u}_k = \begin{bmatrix} v_k \\ \dot{\theta}_k \end{bmatrix} \text{ and } \boldsymbol{\xi} \sim \mathcal{N}(0, \mathbf{Q}) \tag{2}$$

The state space vector $\mathbf{x}_k$ for discrete time $k$ consists of the $x \in \mathbb{R}$ and $y \in \mathbb{R}$ coordinates and the orientation angle $\theta \in [0, 2\pi]$. The discrete transfer function

$f_k : \mathbb{X} \times \mathbb{U} \rightarrow \mathbb{X}$ describes the transfer from $\mathbf{x}_{k-1} \in \mathbb{X}$ to $\mathbf{x}_k$ by using $\mathbf{u}_k \in \mathbb{U}$. The control vector $\mathbf{u}_k$ for the actuator describes velocity $v \in \mathbb{R}$ into current direction and rotation speed $\dot{\theta} \in \mathbb{R}$. Since knowledge about the current state is always faced with some uncertainty, let $\boldsymbol{\xi}$ be zero mean Gaussian noise with covariance matrix $\mathbf{Q}$. With this function it is possible to predict a trajectory $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$ from a starting point $\mathbf{x}_0$ in state space by using a control vector sequence $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u_k}$ (see [7]).

As *Model Predictive Control* (MPC) we used the *Control Particle Belief Propagation* (C-PBP) algorithm [8]. It combines a Markov Random Field factorization and multimodal, gradient-free sampling to perform simultaneous path finding and smoothing in high-dimensional spaces. A drawback of this sample based MPC is some noise in the output signal that causes a slightly modified trajectory after each calculation. In general, the algorithm can be divided into three functional steps. C-PBP starts with an initial point in state space. Then it samples the control space with additional information like upper and lower bounds for each control dimension and a normal distribution at the mean control value. The drawn control value samples are applied to the system model to predict the following states. This generates a tree of trajectories, known as guided random walkers (see Fig. 2). These are evaluated by cost functions. If the costs are *normal*, then the algorithm will proceed in the same way, otherwise it performs a *resampling step* (see blue lines in Fig. 2). This step prunes all *expensive* trajectories and copies the *cheap* ones up to the current step. Then the algorithm can proceed as before. After the maximum number of time steps, known as control horizon, is exceeded, the best trajectory selected by total minimal costs is chosen (see violet trajectory in Fig. 3). Then a backward local refinement algorithm smooths it to gain the optimal trajectory (see red trajectory in Fig. 3). The resulting trajectory does not need to be globally optimal, but it is the optimum of all sampled trajectories. After this calculation, the first control vector is applied to the system and the sensors update the real physical state. The former optimal trajectory will be shifted by one time step and used as initial guess for the next trajectory planning call (see Fig. 4).

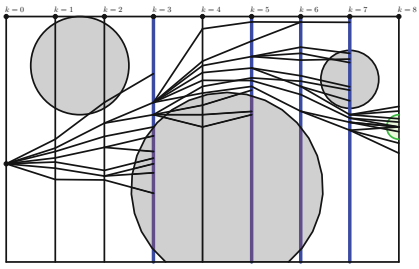


**Fig. 2.** Step 1 and 2 (see [2,8]) (Color figure online)

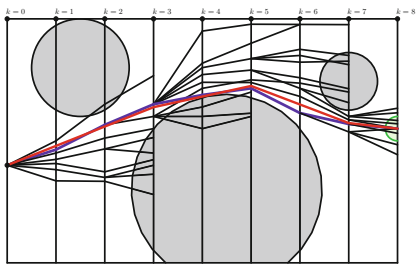


**Fig. 3.** Step 3 (see [2,8]) (Color figure online)

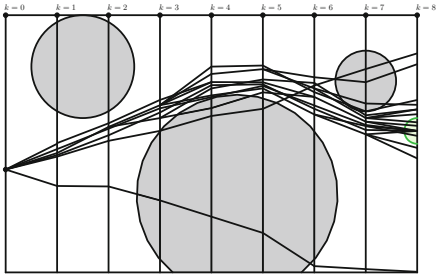


**Fig. 4.** Knowledge transfer (see [2,8])

A cost function based on the robot state and control vector has to be created to evaluate optimality of the control. However, no goal state or reference trajectory is available to evaluate the distance to the optimum. This is different from classic control tasks where the optimum is known. Since the only assumption of C-PBP is that the cost function $l(\mathbf{x}_k, \mathbf{u}_k, k)$ can be split into a function for the state space and one for the control space, problems because of discontinuities do not appear. Moreover, the functions have to be defined for the whole state, control and time space:

$$l(\mathbf{x}_k, \mathbf{u}_k, k) = s(\mathbf{x}_k, k) + c(\mathbf{u}_k, k) \tag{3}$$

Let $\mathbf{z}_k = [\mathbf{x}_k \ \mathbf{u}_k]^T$ denote the optimization vector consisting of state vector $\mathbf{x}_k$ and control vector $\mathbf{u}_k$ at time step $k$. For minimization, the cost function is transformed to maximize the probability density $\mathcal{P}(\mathbf{z})$ of the complete trajectory $\mathbf{z} = [\mathbf{z}_0, \mathbf{z}_1, ..., \mathbf{z}_K]$. Let $K$ be the planning horizon. The transformation is done through exponentiation of the cost function to approximate sampling from $\mathcal{P}(\mathbf{z})$.

$$\begin{aligned} \mathcal{P}(\mathbf{z}) &= \tfrac{1}{Z} \prod_k \exp\left[-\tfrac{1}{2}(s(\mathbf{x}_k, k) + c(\mathbf{u}_k, k))\right] \\ &= \prod_k \alpha_k(\mathbf{x}_k)\beta_k(\mathbf{u}_k) \qquad\qquad = \prod_k \psi_k(\mathbf{z}_k) \end{aligned} \tag{4}$$

$Z$ is a normalization constant and $\alpha_k$ and $\beta_k$ denote state and control potential functions. Equation 4 assumes that the trajectory fragments $\mathbf{z}_k$ always form a valid trajectory. Each $\mathbf{z}_k$ is a separated random variable, so it has to be connected to its adjacent neighbors. In general terms it can be written as *Markov Random Field* (MRF) model (see [9]):

$$\mathcal{P}(\mathbf{z}) = \frac{1}{Z}(\prod_s \psi_s(\mathbf{z}_s))(\prod_{\{s,t\} \in \mathbb{E}} \Psi_{s,t}(\mathbf{z}_s, \mathbf{z}_t)) \tag{5}$$

Based on the formulation as MRF we analyze how the *Particle Belief Propagation* (PBP), as a general sample based belief propagation method, can be applied. The main reason for the application of this method is the combination of global sampling to explore multimodal optimization landscapes "and dynamic programming to fight the curse of dimensionality" [8]. The method evaluates

$\mathcal{P}_k(\mathbf{z}_k)$ instead of much higher-dimensional $\mathcal{P}(\mathbf{z})$, so it processes trajectory segments and not the whole trajectory at once. The segments themselves are not interesting, but they offer the possibility to gain a set of complete trajectories due to the help of belief propagation. Equivalent to other belief propagation methods, PBP tries to calculate the beliefs $\mathcal{B}_k(\mathbf{z}_k)$ of the graph variables. If the graph is free of cycles, then the beliefs are proportional to the marginal probability density $\mathcal{P}_k(\mathbf{z}_k)$. Thus, messages $m_{s\to k}(\mathbf{z}_k)$ from other nodes form the belief of node $k$:

$$\mathcal{B}_k(\mathbf{z}_k) = \psi_k(\mathbf{z}_k) \prod_{s\in\Gamma_k} m_{s\to k}(\mathbf{z}_k) \tag{6}$$

$$m_{s\to k}(\mathbf{z}_k) = \sum_{\mathbf{z}_s\in\mathbb{Z}_s} \Psi_{s,k}(\mathbf{z}_s,\mathbf{z}_k)\psi_s(\mathbf{z}_s) \prod_{u\in\Gamma_s\setminus k} m_{u\to s}(\mathbf{z}_s) \tag{7}$$

"Here $\Gamma_s$ denotes the set of neighbors of node $s$, and $\mathbb{Z}_s$ is the domain of $\mathbf{z}_s$. The messages $m_{s\to k}$ from node $s$ to node $k$ can be considered as (unnormalized) probability density functions of the target node variables $\mathbf{z}_k$. The potentials $\psi_k(\mathbf{z}_k)$ represent the *evidence* for $\mathbf{z}_k$ (here based on the state and control costs), which propagate through the graphical model via the messages. The belief $\mathcal{B}_k(\mathbf{z}_k)$ equals the product of the direct and propagated evidence. In Particle Belief Propagation, the messages and beliefs of Eqs. 6 and 7 are estimated by samples $\mathbf{z}_k^{(i)} \sim q_k(\mathbf{z}_k^{(i)})$, where $i$ denotes the sample index and $q_k(\mathbf{z}_k^{(i)})$ is an arbitrary proposal distribution. Division by the proposal then gives the importance-weighted sample messages and beliefs [8,9]:

$$\hat{m}_{s\to k}(\mathbf{z}_k^{(i)}) = \frac{1}{N}\sum_{j=1}^{N} \Psi_{s,k}(\mathbf{z}_k^{(j)},\mathbf{z}_k^{(i)}) \frac{\psi_s(\mathbf{z}_s^{(j)})}{q_s(\mathbf{z}_s^{(j)})} \prod_{u\in\Gamma_s\setminus k} \hat{m}_{u\to s}(\mathbf{z}_k^{(i)}) \tag{8}$$

$$\hat{\mathcal{B}}_k(\mathbf{z}_k^{(i)}) = \frac{\psi_k(\mathbf{z}_k^{(i)})}{q_k(\mathbf{z}_k^{(i)})} \prod_{s\in\Gamma_k} \hat{m}_{s\to k}(\mathbf{z}_k^{(i)}) \tag{9}$$

Let $N$ be the sample size. The sample belief $\hat{\mathcal{B}}_k(\mathbf{z}_k^{(i)})$ represents the marginal probability density of the trajectory segment that is generated by a simulation step ending in $\mathbf{x}_k$ using $\mathbf{u}_k$. PBP has already the ability to approximate the multimodal marginal distributions corresponding to different paths to pass obstacles.

CPBP extends the basic PBP algorithm in several aspects for control [8]:

1. Selection of proposals to sample feasible trajectories.
2. Adaptive resampling to adjust between local and global search.
3. A local refinement backward pass.
4. Information propagation between function calls, to use the old optimal trajectory as optimizer warm start.

### 2.2 Cost Functions

After the model and optimization has been introduced, we tackle the last element of the model predictive controller. In classic control theory exists a working

state, a goal state or a reference trajectory, thus an error value can be calculated by metric distances. In a completely unknown environment there exists in principle a global minimum as desired working state and often an optimal path towards it, but both are unknown. Therefore, a relative rating is impossible. Consequently, the environment will be modeled with absolute cost referring to time and space. The C-PBP method does not require continuity nor differentiability of cost functions because it works gradient free, so the only assumption is that the functions are defined on the whole space created by the product of state space, control space and time. Here, a high positive value represents repulsive unwanted locations in the state space and low values or high negative values represent recommended locations of the cost function $l$.

$$l(\mathbf{x}_k, \mathbf{u}_k, k) = s(\mathbf{x}_k, k) + c_t(\mathbf{u}_k, k) \quad \text{with } l : \mathbb{X} \times \mathbb{U} \times \mathbb{R}_0^+ \to \mathbb{R} \tag{10}$$

For most of the objects in the environment as well as for collision avoidance, a linear or quadratic function based on the distance is suitable to prevent collisions.

$$d = \|\mathbf{p}_k - \mathbf{x}_k\|_{2D} = \sqrt{(\Delta x)^2 + (\Delta y)^2}$$
$$\text{with } \mathbf{p}_k = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}, \mathbf{x}_k = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \text{ and } a, b \in \mathbb{R}, d, r, \gamma \in \mathbb{R}_0^+ \tag{11}$$

$$l_{\text{lin}}, l_{\text{quad}} : \mathbb{R}_0^+ \to \mathbb{R}$$
$$l_{\text{lin}}(d) = \begin{cases} \frac{-a}{r} \cdot d + a & , \text{ if } d \leq r \\ 0 & , \text{ else} \end{cases} \tag{12}$$
$$l_{\text{quad}}(d) = \begin{cases} \frac{-a}{r^2} \cdot d^2 + a & , \text{ if } d \leq r \\ 0 & , \text{ else} \end{cases} \tag{13}$$

To form a global attractive point, for example as desired goal position, a rational function is more appropriate. However, the function should always be bounded to a maximum value.

$$\lim_{d \to \infty} l_{\text{unbounded}}(d) = 0 \tag{14}$$
$$\lim_{d \to 0} l_{\text{unbounded}}(d) = \pm\infty \tag{15}$$

$$l_{\text{unbounded}} : \mathbb{R}_0^+ \to \mathbb{R}$$
$$l_{\text{unbounded}}(d) = \begin{cases} \frac{b}{d} & , \text{ if } |\frac{b}{d}| \leq |a| \\ a & , \text{ else} \end{cases} \tag{16}$$

If entities like convoy vehicles have a repulsive area to prevent collision and an attractive area so that the robots surround vehicles, then the robot social

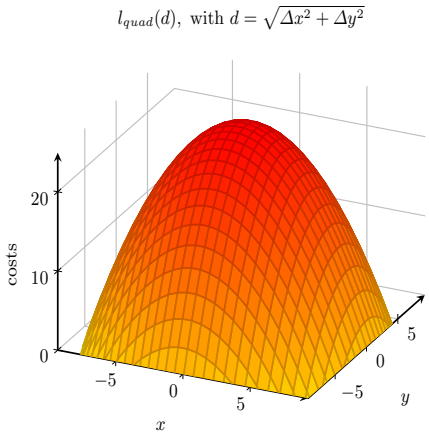


**Fig. 5.** Quadratic bounded cost function width radius $r = 10$ and maximum $a = 25$

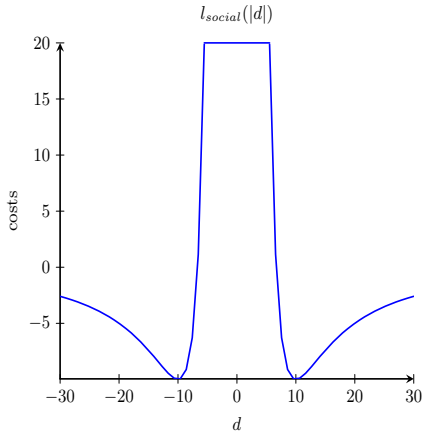


**Fig. 6.** Robot social function with desired minimum costs $L_{\min} = -10$, maximum costs $L_{\max} = 20$ and a desired distance $d_{\text{desired}} = 10$ for the minimum

function can be used. This is a rational function too, but with some additional constraints (Figs. 5 and 6):

$$l_{\text{social}}(d) = \begin{cases} \frac{b_1}{d^{\gamma_1}} - \frac{b_2}{d^{\gamma_2}} & , \text{ if } |\frac{b_1}{d^{\gamma_1}} - \frac{b_2}{d^{\gamma_2}}| \leq |a| \\ a & , \text{ else} \end{cases}$$
$$\text{with } b_1, b_2 \geq 0, \gamma_1 > \gamma_2 > 0 \tag{17}$$

Multidimensional functions for complex polygons and obstacles like sectors for towers, or complex missions like setting up a MANET or connection awareness are implemented as well, but they are often aggregations of the presented functions.

Furthermore, we developed a cost function considering energy consumption and accessibility of the terrain to cover different types of ground like rocks, mud, streets, fields, etc.

$$l_{\text{movement}}(\mathbf{x}_k, \mathbf{u}_k, k) = v^2 \cdot G(\mathbf{x}_k) \cdot \rho_T + \dot{\theta}^2 \cdot G(\mathbf{x}_k) \cdot \rho_R \tag{18}$$

$G(\mathbf{x}_k)$ denotes a position depending influence factor of the ground. $\rho_T$ and $\rho_R$ describe the effect of this influence factor on the translation and rotation movement. The velocity values are squared values to prefer slower actions.

### 2.3 Communication Model

A necessary condition for entities to form a swarm is the information transfer via communication. Therefore, we created a network simulation model based on signal power dissemination depending on the used frequency including free space

damping and radiation obstacles like walls. In addition to that, a noise floor and active signal jammers can be created. This realistic network simulation serves to analyze and implement strategic behavior to signal changes and losses of single swarm agents or of the whole swarm. Furthermore, one mission is to create a MANET. Therefore, it is mandatory to predict the connection status and quality. To filter the messages, each message gets an unique identification number and to create a logical as well as temporal order on the messages, lightly modified Lamport clocks [13] are used.

### 2.4 Bandwidth Prediction

Signals which are transmitted via *orthogonal frequency-division multiplexing* (OFDM) like *Long-Term-Evolution* (LTE) are divided into subcarriers. Using a channel width of $\Delta f_c = 10\,\text{MHz}$, a carrier distance of $\Delta f_t = 15\,\text{kHz}$ and including protection distances lead to $\#_{\text{carrier}} = 600$ subcarriers. Thus, the number of subcarriers is proportional to $60\,\frac{1}{\text{MHz}}$. The carrier distance determines the symbol time:

$$t_s = \frac{1}{\Delta f_t} \approx 66,7\,\mu\text{s} \tag{19}$$

Once per symbol time each subcarrier is modulated with one symbol. The symbol width $w_{\text{symbol}}$ depends on the modulation method. In the simulation environment *Quadrature Phase-Shift Keying* (QPSK), 16 *Quadrature Amplitude Modulation* (QAM), 64 QAM and 256 QAM are available. The maximum bandwidth for a channel width 10 MHz, carrier distance 15 kHz and 256 QAM modulation method is for example given by:

$$\Delta f_c \div \Delta f_t = 10\,\text{MHz} \div 15\,\text{kHz} \rightarrow 600 = \#_{\text{carrier}} \tag{20}$$

$$\frac{\#_{\text{carrier}} \cdot w_{\text{symbol}}}{t_s} = \#_{\text{carrier}} \cdot w_{\text{symbol}} \cdot \Delta f_t = 600 \cdot 8\,\text{bit} \cdot 15\,\text{kHz} = 72\,\text{Mbit/s} \tag{21}$$

To increase this bandwidth Multiple Input Multiple Output [1] methods are supported. Based on the maximum, we use the received signal strength index (RSSI) and the signal to noise ration (SNR) to estimate the real bandwidth.

$$\text{RSSI} = \underbrace{P_t + G_t + G_r}_{\text{transmission power}} + \underbrace{20 \cdot \log_{10}\left(\frac{c}{f \cdot 4\pi \cdot d}\right)}_{\text{free space damping}} - \underbrace{\sum^{i} P^i_{\text{obstacle}}}_{\text{obstacle damping in line of sight}} \tag{22}$$

The extended Friis Eq. 22 [5] describes the RSSI of the receiver in decibel based on one milliwatt (dBm). $P_t$ denotes the sending power, $G_t$ and $G_R$ denote the antenna gain of the sender and receiver. The free space damping depends on the speed of light $c$, the sending frequency $f$ and the distance $d$ between the modules. For calculating the SNR, a noise floor power $P_{noise}$ has to be given.

$$\text{SNR} = \text{RSSI} - P_{\text{noise}} \tag{23}$$

The software module can be easily extended by more complex reflection models for electromagnetic waves or interface OMNeT++ [11] for other models.

## 3 Structure of the Simulator

The simulation software is structured in a graphical visualization with an interface and the simulation core because this core should be usable for other simulation tools and visualization engines. Core functions can be used on real world robots to validate the simulation and to create digital twins of them in the visualization.

### 3.1 Structure of the Simulation Software

The software is implemented in C++ because of its efficiency, the available libraries and OpenMP support. We used an existing implementation of C-PBP as the optimization part for the model predictive controller (see [8]). In addition to OpenMP, *Eigen* [10] is included as a fast numeric library. Moreover, a *yaml-cpp* library offers the possibility to read YAML configuration files for the simulated scenarios. These files configure the whole scenario and all parameters of each entity. Furthermore, the appearance of the interface and images can be configured as well. Consequently, no programming knowledge is necessary to use the simulator.

The application is divided into two domains (see Fig. 7). The first domain is visualization of simulation results. It consists of one thread for rendering and a few callbacks for interaction with the user. The second domain is the simulation core. It has one thread to calculate the physics of the objects and one thread to calculate the network states like bandwidth. Furthermore, each robot creates four threads on his own. These are needed to realize a realistic simulation implementation of the hardware and software components in real world. A robot consists of a communication module, an actuator, a sensor unit and a planning component (see Fig. 1).

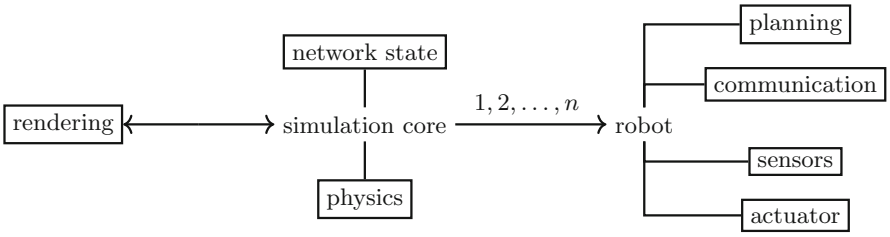


**Fig. 7.** Thread structure of the simulator

For analysis purposes, the software offers the possibility to display the value of cost functions in vicinity of a selected robot by using Gnuplot. In addition

to this function, the scenario data such as noise floor and terrain accessibility values and all robot states including planned trajectories, network states, cost function values of the surrounding area and calculation times can be exported as CSV and text files with a special format for easy processing in statistic tools, e.g. Matlab®.

### 3.2 Graphical User Interface

The graphical user interface uses the Cocos2d-x game engine. It is a lightweight cross-platform game engine written in C++ and optimized for 2D applications. The application can only be controlled via keyboard at the moment.

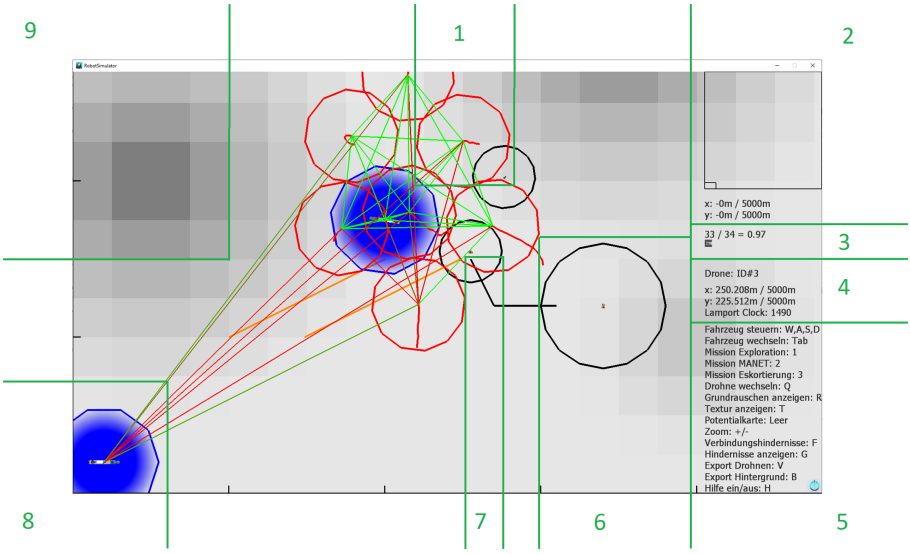


**Fig. 8.** Screenshot of the simulator (Color figure online)

Figure 8 shows a screenshot of the running simulation. The following description of the simulator is done clockwise. A representation of a robot is shown in area 1. The robot is located in the middle of the red approximated circle which corresponds to the sensor range of the robot and has a diameter of 60 m. The wavy line, which starts from the circle center towards the right side, is the planned trajectory of the robot. All other connected lines with colors between light green and red represent the active network connection and their quality. The green color denotes maximum bandwidth, and completely red the lowest possible bandwidth. The size of the robot images are so small because its real world dimension is less than $1\,m^2$ and the displayed area is 600 m by 275 m. The area number 2 shows a minimap and the current rendered extract of the simulated region which is $25\,km^2$ in total. The next section displays at first

the simulated time in seconds, then the actual running time of the application. Thus, the division result describes the real-time factor. Area number 4 presents an identification number of the currently selected robot and its coordinates as well as its logical Lamport clock state for messages. At the lower right corner, control information is given to the user. The 6th section highlights a radial obstacle with a bounded effect radius which is shown as a black approximated circle. The obstacle uses one of the distance based cost functions. Then a line based obstacle follows that represents a wall which shall not be passed. A guidance vehicle to send data or missions to the robots is presented in the lower left corner denoted by 8. This vehicle demonstrates the different connection qualities. Normally, the quality decreases with distance so it is dyeing from green to red. But some of the shorter connection lines intersect one of the orange lines. They represent obstacles which causes a damping on the transmission power. So, the bandwidth is significantly reduced. The last area number 9 shows the accessibility of the terrain. The darker a ground tile is, the harder is the access and movement, so it increases the use of energy and cause the robots to try to move slower. Additional features are zooming, hiding entities and displaying the noise floor instead of the terrain accessibility (see Fig. 9). Furthermore, the obstacles can move with scripted behavior and the guidance vehicles can be controlled by a script or the user and send missions to the robots.

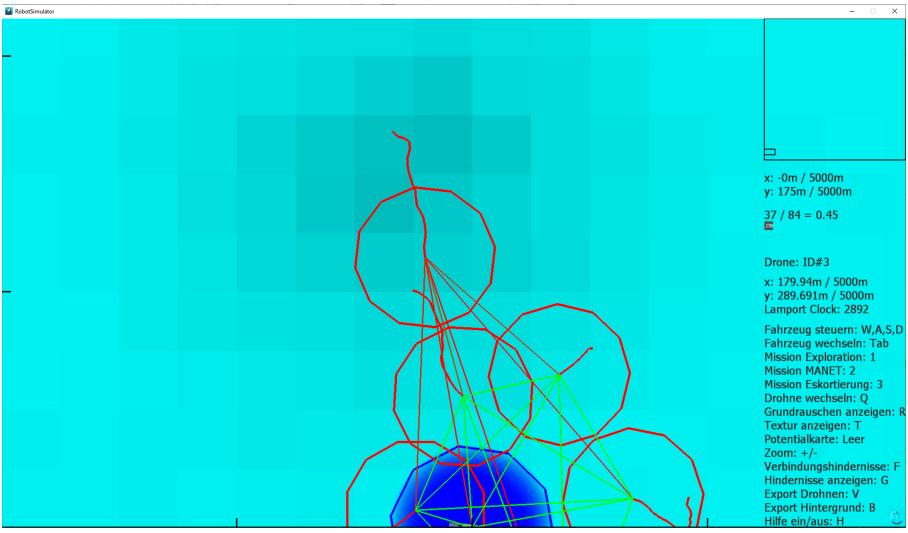


**Fig. 9.** Screenshot of the simulator (Color figure online)

Figure 9 shows the noise floor of the environment. Equal to the terrain accessibility, the noise level increases, the darker the cyan tiles are. The dark circular discoloration at almost the center of the screenshot indicates the position of an active jammer. Moreover, the qualities of the connections of the robot next to

the jammer is significantly reduced, although the connection module distances are as short as the light green connections with higher bandwidth between the other robots.

### 3.3 Simulation Internals

To solve the given missions each robot creates for each other known robot a model to maintain the data it receives from that robot and to do predictions concerning its states in the future.

### 3.4 Exploration Mission

The goal of the exploration mission is to discover a dynamic environment. Since the dynamic information becomes obsolete, it has to be renewed from time to time. The area covered by a robot sensor is deemed to be explored. Here, the precision of exploration decreases with the sensor distance, so peripheral areas are less discovered. In addition to that, the robot should create a vanishing trace of the discovered area. This specification leads to the use of radial cost functions with a limited effect radius, which represent the sensor range. Each robot model as well as the robot itself gets a FIFO list with 18 elements per default. After a specified time, the default is 10 s because of the given maximum speed, such an radial cost function object is added to this list. If the list is already filled, the oldest object gets deleted. Moreover, the cost of each object decreases during simulation time.

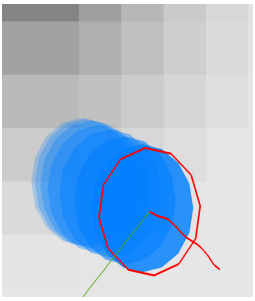

**Fig. 10.** Visualization of the trace. Gray scale represents terrain accessibility. (Color figure online)

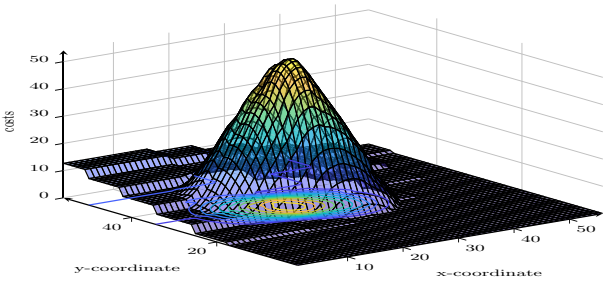


**Fig. 11.** Visualization of the cost function for exploration at the same time. The robot is located in the center.

In Fig. 10 the blue filled circles visualize the created cost function objects. The lighter the color is, the older is the object. The color as well as the height represent the costs in Fig. 11. The rectangular surfaces visualize the costs which are generated through movement on terrain with different accessibility.

### 3.5 Convoy Escorting Mission

During the escorting of a convoy the robots should place themselves around the specified target vehicle in an optimal distributed way. To maintain the desired distance, the robot social function is used. Therefore, the target vehicle sends its position and velocity vector to the robots Thus, they are able to predict the movement, even when the connection gets lost. To gain an optimal distribution around the target, each robot calculates the intersecting sensor area with other robots and tries to minimize it by using the robot model. This leads to a maximized covered area at the desired distance. Figure 12 shows a target vehicle in the center and an optimal distribution of the robots around. The blue circle denotes the desired distance and the red circles the sensor range of each robot. The green lines are the connections like before. The corresponding cost function is shown in Fig. 13.

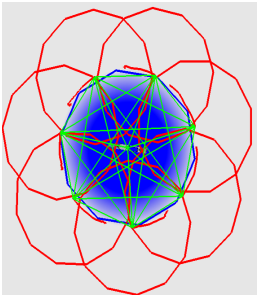

**Fig. 12.** Visualization of the distribution of the robots around the escorted vehicle. (Color figure online)

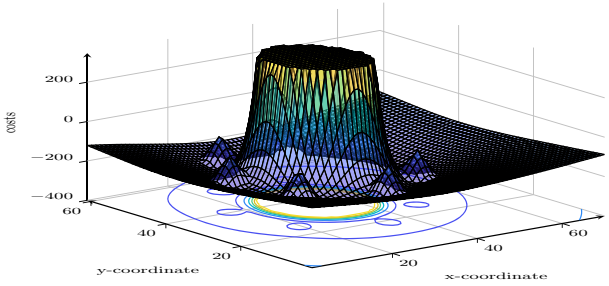


**Fig. 13.** Visualization of the convoy mission cost function at the same time.

### 3.6 MANET Mission

The MANET cost function cannot be created by an aggregation of the presented functions, because as long as the connection quality is constant, a large distance is desired to cover a larger area. If the quality decreases it has to be weighed out whether a larger distance or a higher bandwidth is prioritized. However, if the connection gets lost, the distance is not a positive aspect anymore and has to be reduced. There are two reasons for a connection loss; it could be the distance or it could be a change in the environment, which cannot always be undone by the robot itself. Thus, a distance reduction is necessary. This is the reason why the active jammer in Fig. 14 causes smaller distances between some robots while they have at most the connection quality of the other robots. Algorithm 1 shows the implementation of the MANET cost function (Fig. 15).

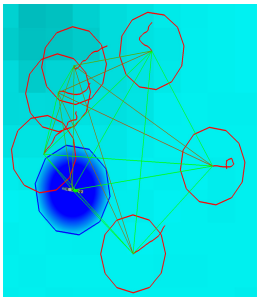

**Fig. 14.** Visualization of the creation of a mobile ad hoc network.

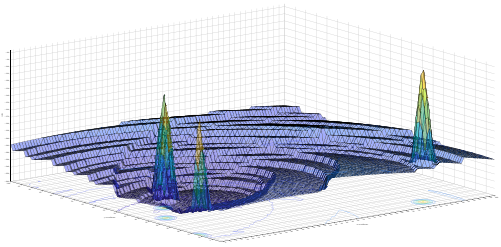

**Fig. 15.** Visualization of the mobile ad hoc network cost functions for nearest robot to the top of the image. Each cone visualizes the collision avoidance area of other robots.

**Require:** $d$: distance between communication modules, *link*: connection between communication modules, *partner*: sender or receiver, $factor_{distance}$: weighting of distance, $MANET_{importance}$: weighting of quality

**function** GETMANETPOTENTIAL($d$)
  $quality \leftarrow link$.predictConnection($partner$, $d$)
  **if** $quality = 0$ **then**
    $costs \leftarrow d \cdot factor_{distance}$
  **else**
    $costs \leftarrow -(quality \cdot MANET_{importance} + factor_{distance} \cdot d)$ ▷ later: $\ln(d)$
  **return** *costs*

**Algorithm 1:** MANET cost function calculation

## 4 Experiments

We begin with some performance experiments for a reference scenario. It consists of two static radial obstacles, two static wall obstacles, one dynamic radial obstacle, one static and one dynamic radiation obstacle with damping as well as terrain accessibility between 0 and 36. Moreover, the noise floor is equally distributed between −120 dBm and −60 dBm. So, most of the simulation features are present in this scenario. Based on the maximum robot speed we figured out 300 ms as the upper bound for trajectory planning time. Table 1 shows that we can simulate up to 680 simulation steps of a trajectory for 8 robots which uses $N = 20$ parallel paths for model predicted controller optimization each. A simulation step has a time delta of 0.5 s. Thus, each robot predicts the next 340 s each 300 ms. This is a waste of resources in a highly dynamic environment.

Instead, we increase the number of parallel plans per robot to $N = 24$ to increase the sampled area to get better trajectory results. Furthermore, we use the available resources to simulate a larger number of swarm agents. Figure 17

**Table 1.** Comparison of the calculation times per trajectory in ms with different amount of simulation steps

| Robots | Steps | Plans | Min. | Max. | Mean | Standard deviation |
|---|---|---|---|---|---|---|
| 8 | 15 | 20 | 4.728 | 7.381 | 6.005 | 0.567 |
| 8 | 40 | 20 | 8.55 | 16.886 | 10.999 | 1.685 |
| 8 | 200 | 20 | 29.161 | 71.702 | 37.739 | 4.476 |
| 8 | 680 | 20 | 111.631 | 242.886 | 124.861 | 8.926 |
| 8 | 840 | 20 | 141.346 | 342.455 | 196.722 | 42.165 |

shows that up to thirty robots, which correspond to $3 + 30 \cdot 4 = 123$ threads, can be simulated on a gaming PC in real time. However, 200 equidistant simulation steps do not perform well. The majority of steps are in a later uncertain future and unknown environment which causes the robots to ignore obstacles in vicinity. Thus, we reduce the number of predicted steps to $K = 60$ and switch to an exponential distribution of simulation steps while maintaining the visual range (Fig. 16).

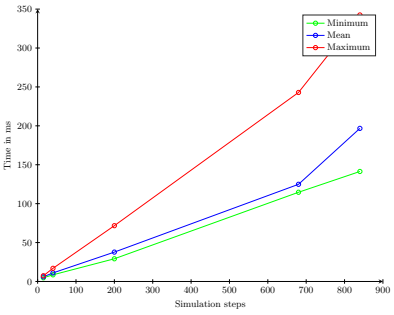


**Fig. 16.** Statistic of plan calculation time with 8 robots and $N = 20$ parallel plans per robot

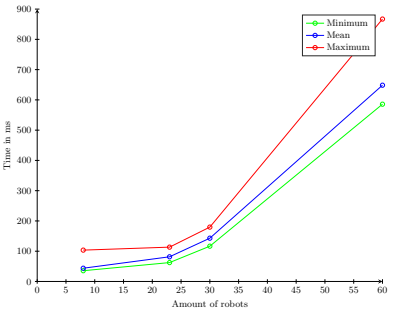


**Fig. 17.** Statistic of the plan calculation time with $K = 200$ steps and $N = 24$ parallel plans per robots

$$\Delta t_i = t_{step} \cdot t_{factor}^i$$
$$\text{with } i = 0, 1, \ldots, K-1; t_{step} = 0,075 \text{ and } t_{factor} = 1,055 \tag{24}$$
$$\sum_{i=0}^{59} \Delta t_i \approx 32,5\,\text{s} \approx 60 \cdot 0,5\,\text{s}; \sum_{i=0}^{39} \Delta t_i \approx 10,2\,\text{s} \tag{25}$$

This has the advantage that obstacles in vicinity are not ignored. Equation 25 shows that relatively more sampled points are in near future, as well as that a coarse outlook on the target is still given. Next, we compare the performance of the simulation on different hardware. First, a mobile PC with an quad core Intel® Core™ i7-8650U CPU, 16 GB RAM and a desktop PC with an eight

core Intel® Core™ i9-9900K CPU, 32 GB RAM and a NVIDIA® GeForce® RTX 2080 Ti GPU. Both systems use SSD storage. Figure 18 indicates that a real-time simulation with up to 18 robots is possible on the mobile device, if additional robots are added it suffers from missing cores to handle the threads as well as a GPU to render so many objects.

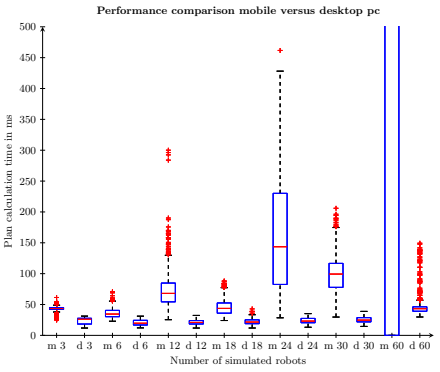


**Fig. 18.** Performance comparison between mobile and desktop PC for different swarm size. "m" denotes mobile PC and "d" desktop PC. The number denotes the amount of swarm agents.

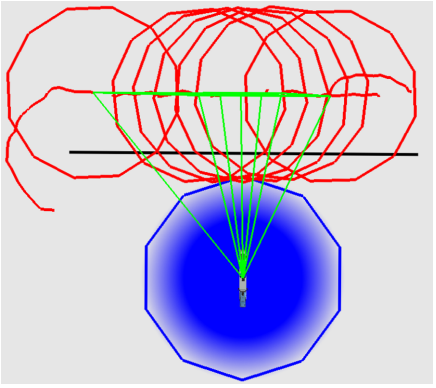

**Fig. 19.** Bypassing of obstacles

### 4.1 Convoy Escorting

In disaster areas or in rural environments a lot of situations exist where trucks or the convoy can pass an obstacle, e.g. because of a higher wading depth. The robots have to find a way around the obstacle. This situation is presented in Fig. 19. The goal is to analyze the maximum dimensions of the obstacles that do not lead to a disintegration of the convoy structure.

**Table 2.** Measurement series to analyze the escorting behavior at obstacles

| Length | 120 m | 150 m | 180 m | 210 m | 240 m | 360 m | 390 m | 420 m |
|---|---|---|---|---|---|---|---|---|
| Following robots | 7 | 6 | 5 | 5 | 4 | 3 | 0 | 0 |

Table 2 contains the relevant obstacle lengths. It should be noted that the obstacles at each end also have an effective radius of 20 m, so that the actual range of influence is 40 m larger than the ranges given in the table. This effect prevents the robots from cutting corners which would lead to a collision. The

sensor range, which is the radius of the red circle, is set to 30 m, so only many times the amount of the range is analyzed. Up to four times the range all robots can follow the vehicle and up to 12 times the range the outer robots can follow. Crucial for bypassing of the obstacle is the swarm size and the cost function that penalizes the intersecting area. Therefore, the robots are distributed along the line. With seven robots, the minimum distance for overlap-free distribution is 420 m. If the influence of the obstacle exceeds this distance (see Table 2 at 390 m + 40 m), no robot can detect an alternative path. Furthermore, the increased distance to the edge of the obstacle reduces the attractive potential of the mission. Here, the activation of the exploration mission can support to further distribute the robots.

### 4.2 MANET

The functionality of the MANET is evaluated by the distribution of the robots. A circular or star-shaped arrangement is desired in order to achieve a high and

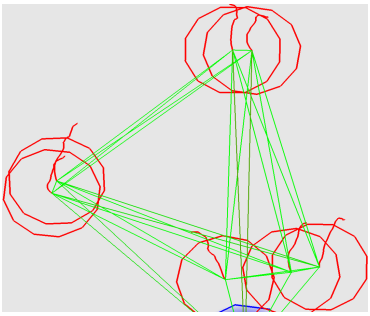

**Fig. 20.** Triangle structure with old MANET cost function

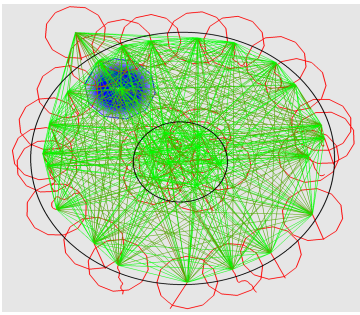

**Fig. 21.** Creation of ring structure with 30 robots, version 1 (Color figure online)

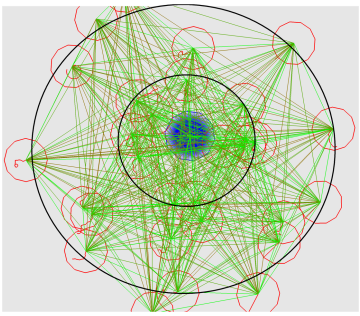

**Fig. 22.** Creation of ring structure with 30 robots, version 2 (Color figure online)

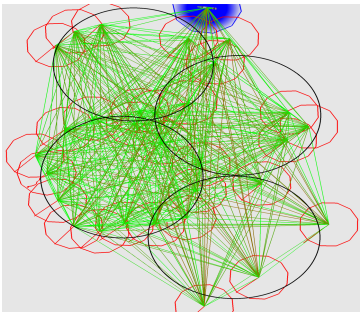

**Fig. 23.** Creation of ring and star structure with 30 robots (Color figure online)

slightly redundant network coverage of an area. However, experiments using the current cost function produce approximately equilateral triangles (see Fig. 20). This is because an equilateral triangle has optimum properties regarding to distances. As a result, the cost function has to be modified. The idea of always maximizing the minimum distance is not appropriate because the connection with the minimum distance does not have to be the optimal connection due to obstacles and sources of interference. As a solution, a logarithmic function may be used. However, the strictly monotonous and concave logarithmic function for the distance prefers small equal distances between the robots, instead of one large and many smaller distances. Figures 21, 22 and 23 demonstrate the functionality of the adjusted cost function because the desired ring or star structures are always formed. The position of the blue marked guidance vehicle is irrelevant and has no influence on the cost functions. Black circles are only for visualization purpose.

## 5 Conclusion

We present a novel real-time simulation tool for robot swarms under limited communication capabilities. It allows the analysis of swarm behavior for three basic scenarios. Due to an appropriate cost function method, the missions can be combined and parameterized, whereby the objectives are automatically selected and prioritized. The target positions of the individual robots in the swarm do not have to be explicitly specified. Instead, the required information for the target and the trajectory to the target are automatically extracted and optimized from the current information about the environment, the swarm agents and the swarm target. This automatic extraction is implemented with the help of an MPC which includes a non-linear model of a specific type of robot. The C-PBP algorithm [8] is used to optimize the control sequence. Since the goal of the task emphasizes a limited communication capability, the simulation of a complete wireless network is implemented. This takes into account fundamental properties of propagation, shadowing and attenuation of electromagnetic signals. In addition, the possibility to include further network influences by jamming transmitters is given. The calculation of the network is based on physical laws and also considers signal propagation times and data transmission times. By transmitting the planned trajectories and detecting obstacles, an emergent behavior is generated which favors the convergence to an optimal swarm behavior. This work differs fundamentally from previous works that accept communication as ideal (see [3,6,16,17]). The simulation tool allows to reproduce realistic situations. The behavior of a swarm of robots can be analyzed with up to 60 robots simultaneously and in real time. Due to the built-in visualization and interfaces to other programs, the behavior of individual agents can be visualized. By using simple configuration files, the user has the possibility to modify almost all settings and parameters. [Gefördert durch die Deutsche Forschungsgemeinschaft (DFG) – 276879186/GRK2193]

## References


1. Ahlers, E.: Funk-übersicht: WLAN-Wissen für Gerätewahl und Fehlerbeseitigung. c't **2015**(15), 178–181 (2015). https://www.heise.de/ct/ausgabe/2015-15-WLAN-Wissen-fuer-Geraetewahl-und-Fehlerbeseitigung-2717917.html
2. Baumann, D., Mager, F., Zimmerling, M., Trimpe, S.: Control-guided communication: efficient resource arbitration and allocation in multi-hop wireless control systems. IEEE Control Syst. Lett. **4**(1), 127–132 (2020)
3. Euler, J.: Optimal cooperative control of UAVs for dynamic data-driven monitoring tasks. Dissertation, Technische Universität Darmstadt, Darmstadt (2018)
4. Euler, J., von Stryk, O.: Optimized vehicle-specific trajectories for cooperative process estimation by sensor-equipped UAVs. In: 2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, 29 May–3 June 2017, pp. 3397–3403. IEEE (2017)
5. Friis, H.T.: A note on a simple transmission formula. Proc. IRE **34**(5), 254–256 (1946)
6. Fukushima, H., Kon, K., Matsuno, F.: Distributed model predictive control for multi-vehicle formation with collision avoidance constraints. In: Proceedings of the 44th IEEE Conference on Decision and Control, pp. 5480–5485. IEEE (2005)
7. Grüne, L., Pannek, J.: Nonlinear Model Predictive Control: Theory and Algorithms. CCE. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-46024-6
8. Hämäläinen, P., Rajamäki, J., Liu, C.K.: Online control of simulated humanoids using particle belief propagation. In: Proceedings of SIGGRAPH 2015. ACM, New York (2015)
9. Ihler, A., McAllester, D.: Particle belief propagation. In: International Conference on Artificial Intelligence and Statistics, vol. 5, pp. 256–263 (2009)
10. Jacob, B.: Eigen. http://eigen.tuxfamily.org/index.php?title=Main_Page
11. Kuntz, A., Schmidt-Eisenlohr, F., Graute, O., Hartenstein, H., Zitterbart, M.: Introducing probabilistic radio propagation models in OMNeT++ mobility framework and cross validation check with NS-2. In: Molnár, S., Heath, J.R., Dalle, O., Wainer, G.A. (eds.) Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops, SimuTools 2008, Marseille, France, 3–7 March 2008, p. 72. ICST/ACM (2008)
12. Lächele, J., Franchi, A., Bülthoff, H.H., Robuffo Giordano, P.: SwarmSimX: real-time simulation environment for multi-robot systems. In: Noda, I., Ando, N., Brugali, D., Kuffner, J.J. (eds.) SIMPAR 2012. LNCS (LNAI), vol. 7628, pp. 375–387. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34327-8_34
13. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM **21**(7), 558–565 (1978)
14. Mager, F., Baumann, D., Jacob, R., Thiele, L., Trimpe, S., Zimmerling, M.: Feedback control goes wireless: guaranteed stability over low-power multi-hop networks. In: Liu, X., Tabuada, P., Pajic, M., Bushnell, L. (eds.) Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2019, Montreal, QC, Canada, 16–18 April 2019, pp. 97–108. ACM (2019)
15. Puzicha, A.: Modeling and analysis of a distributed non-linear model-predictive control for swarms of autonomous robots with limited communication skills (in German). Master's thesis, Department of Computer Science, TU Dortmund (2019)
16. Ritter, T.: PDE-based dynamic data-driven monitoring of atmospheric dispersion processes. Dissertation, Technische Universität Darmstadt, Darmstadt (2017)

17. Strobel, A.: Verteilte nichtlineare modellprädiktive Regelung von unbemannten Luftfahrzeug-Schwärmen. Dissertation, Technische Universität Darmstadt, Darmstadt (2016)
18. Tricaud, C., Chen, Y.: Optimal Mobile Sensing and Actuatin Policies in Cyber-physical Systems. Springer, London (2011). https://doi.org/10.1007/978-1-4471-2262-3
19. Vieira, B., Severino, R., Filho, E.V., Koubaa, A., Tovar, E.: Copadrive - a realistic simulation framework for cooperative autonomous driving applications. In: Proc. 8th IEEE International Conference on Connected Vehicles and Expo (ICCVE 2019). IEEE (2019)