A Simulation Framework for Testing GMTI Trackers

Gereon Schüller
Dept. Sensor Data and Information Fusion
Fraunhofer FKIE
Wachtberg, Germany.
gereon.schueller@fkie.fraunhofer.de

Michael Mertens
Dept. Sensor Data and Information Fusion
Fraunhofer FKIE
Wachtberg, Germany.
michael.mertens@fkie.fraunhofer.de

Abstract – *Trackers for Ground Moving Target Indicator (GMTI) radar have to be benchmarked in order to yield good results. For many benchmarking purposes, the usage of real data may become expensive and infeasible. It may also lack of knowledge of the ground truth. Simulating GMTI situations is a less-expensive alternative to real measurements. In order to get a realistic scenario, the motion and radar cross section (RCS) of ground targets, the motion of sensor platforms and the properties of GMTI sensors have to be simulated. In this paper, considerations for a realistic motion and sensor model are presented. Then, an implementation of a framework for simulation is shown, together with first experiences in tracker benchmarking using this framework.*

Keywords: Target simulation, sensor simulation, tracker benchmarking

1 Introduction

Ground Moving Target Indicator (GMTI) is a radar process that detects moving objects on the ground by exploiting the Doppler shift of an emitted electromagnetic signal reflected by such objects. As the Doppler shift is directly connected to the radial velocity of the reflector, GMTI sensors are able to distinguish between the stationary background and moving objects. Additionally, GMTI sensors can detect the direction towards the object by means of physical or synthetic aperture and the distance by means of delay measurement [1]. Thus, GMTI is the method of choice when the movement of cars, trucks etc. in a given area shall be monitored. Typically, GMTI sensors are mounted on airborne platforms, like aircrafts or unmanned aerial vehicles (UAVs, "drones"). There is a wide range of applications for the these scenarios in the civil and military sector, e. g. surveillance of traffic flows on highways, coverage of wildlife preservation areas or – in the military applications – detection of adversary vehicles that are approaching a military camp.

For the practical application of these systems, tracking algorithms for GMTI are indispensable, as the raw data are hard to interpret and suffer from many deficiencies. Especially GMTI suffers from the effect of "'Doppler blindness", i. e. objects disappear due to their low radial velocity, either due to intrinsic behaviour or projection effects. In that case, the object is invisible to the sensors. It is the task of an appropriate tracking algorithm to pick up a previously identified track once the corresponding object comes out of the Doppler-blindness.

For evaluating appropriate algorithms, benchmarking is necessary to avoid problems and to select the best tracking algorithm for a given situation. However, real exercises are expensive and can be carried out only with already existing systems. A major withdraw is that the ground truth has to be recorded in an independent way. To avoid these problems, a simulation is desirable.

In this paper, a simulation framework for typical GMTI scenarios is presented that is able to simulate the motion of ground objects, the motion of a flying platform and the characteristic of GMTI sensors. The systems features many tunable parameters, a modular architecture and a graphical user interface (GUI) for easy usage.

The rest of the paper is organized as follows: In the following section, the scenario is formally defined. In Section 3 the implementation of the road target simulator is presented. The simulation of an airborne platform is presented in 4. Section 5 shows the implementation of the sensor simulator. In Section 6 evaluation results of several tracking algorithms are presented.

2 Scenario

In the simulation, the scenario consists of M targets, moving on a fixed road network. A number of S airborne sensors measures the movement on the ground. It is assumed that each platform has one sensor, but every sensor may consists of $N_s, s \in [1, S]$ array elements.

Each vehicle m moves according to the linear model

$$\vec{x}_{m,t+1} = \vec{F_m}\vec{x}_{m,t} \tag{1}$$

where the the state

$$\vec{x} = (\ddot{x}, \ddot{y}, \ddot{z}, \dot{x}, \dot{y}, \dot{z}, x, y, z)^\top \tag{2}$$

comprises the acceleration vector, the velocity vector and the current position (indices omitted for the sake of simplicity). The state transition matrix $\vec{F}$ describes the movement of the vehicle from time step t to time step $t+1$. It is fixed on each road and changes at crossroads. The on-road targets move on a road network that is defined by a set $\mathcal{R} = \{(c, c')|c, c' \in \mathcal{C}\}$ of roads and a set of crossroads $\mathcal{C} = \{(x, y, z)\}$ defined by the their three-dimensional position. It is required that each crossroad is connected at least to one road. So the designation *crossroad* is different from its usual understanding, but it simplifies the following considerations. In the road map, all roads are assumed to be straight between the crossroads, while curves can be simulated by many consecutive "crossroads" that give a nearly curve-shaped form, the same structure as in digital vector road maps. The curvature of the earth can be neglected due to the relatively small field of view.

Each airborne sensor platform flies at a fixed altitude h. Its movement is also considered as a linear one, comparable to the considerations above. The flight path is defined by several supporting points that play a similar role as the crossroads.

The measurements of the GMTI sensor are carried out by emitting a radio impulse at a frequency ν, repeated in a pulse repetition interval PRI. Each scan covers a region of to angles (α, ϵ), where α, the direction in the $x-y$-plane towards north is called *azimuth* and ϵ is called *gazing angle*, the angle against the horizontal plane.

3 Simulating Road Targets

The task of the Road Target simulator is to generate target positions from a road map on given way. In detail, the input parameters for the road simulator consist of:

- Starting point, end point and (optional) points to be visited (stopovers)
- Start velocity, maximal acceleration, maximal deceleration and standard deviation for the process noise
- The road map
- A sampling interval

The stopping behaviour is defined by marking points on the road map as stop-points or by defining a stop-probability that "dices" a stop event. When the target has stopped, it will restart after a random time. The algorithm now works as follows: First, a graph is build using the given road map. Then, a shortest way in the road map between start and end point is calculated using a Dijkstra algorithm [2], that uses the Cartesian distance as weight function. After this preparation step, the driving simulation itself is carried out. The outline of the RoadSim algorithm is shown in alg. 1 with the following definitions: The edged of the path are stored in a queue [3] (a first-in first-out buffer), with an operation called *pop* that removes the front element of the queue and returns its content. The function $pos(e)$ returns the position vector of a node, Δt is the sampling interval for the position data. The function $n(\mu, Q)$ returns a normal distributed random number with covariance Q and mean μ. In the algorithm, the covariance for position and speed are stored in the variable v and P. The speed is modeled as a one-dimensional normal distribution, while the position is modeled as three-dimensional distribution to simulate small left-right movements of the vehicle, e.g. due to side winds, imperfect roads etc. The variance in up-down directon is usually very small.

The subroutine `AccDec` decides whether the object should accelerate or decelerate. In the *normal* mode, the vehicle drives with a constant frontal speed (plus some noise). When a stop event (either due to the stop probability or a node marked as "stop point") occurs, the algorithm will decelerate the vehicle by decreasing the frontal speed in every step by a predefined value (e. g. 4 ms^{-2}) until the speed has reached 0. While stopped, the last position is assumed a current position. The algorithm "waits" a Bernoulli distributed random amount of time laps (n.b. for security reasons, the time could be bounded to avoid unpredictable run times). After that, the algorithm increases the velocity until it has reached the normal frontal speed. When the distance between the current position and the next node has reached zero or below (the function dist(a,b) is supposed to deliver negative values, w.r.t the direction of motion), the next edge is taken and the actual position is corrected to lie on the new edge (see fig. 1). In the current implementation, the user can also mark points where the speed can change permanently to simulate highways, traffic jams etc. This feature is straightforward.

The algorithm finishes when there is no edge left in the list $\mathbb{E}$. After execution, the output arrays $currPos(t)$ and $v(\vec{t})$ contain the position and velocity vector at all time steps t. The algorithm doesn't perform any smoothing when the direction changes. Thus, the smoothing depends on the quality (resolution) of the underlying map material.

Convois can be simulated by starting the algorithm with different start time points, followed by a merge of the output arrays. The current implementation uses an ID number for each target so that the members of a convoy can be distinguished.

Algorithm 1 Calculation of target positions

Require: $\mathbb{E}$ a queue of consecutive edges $E = (e, e')$, *speed* the velocity of the target

```
t ← 0
while 𝔼 ≠ ∅ do
  currPos(t) ← pos(e)
  v⃗(t) = (pos(e′)−pos(e)) / |pos(e′)−pos(e)| · (speed + n(0, v))
  while dist(currPos(t), pos(e′)) > 0 do
    /*accelerate/decelerate (see alg. 2)*/
    currPos(t) ← currPos(t) + v⃗(t) · Δt + n(0, P)
    d ← dist(currPos(t), pos(e′))
    t ← t + Δt
  end while
  (e, e′) = pop(𝔼)
  v⃗(t) = (pos(e′)−pos(e)) / |pos(e′)−pos(e)| · speed + n(0, V)
  if d < 0 then
    pos(t) ← pos(t − Δt) + v⃗(t)/|v⃗(t)| · |d| + n(0, P)
  end if
end while
```

Algorithm 2 Deceleration/Acceleration

```
if rnd() > prob_stop ∨ mode = 'dec' ∨ stop(e) then
  decelerate()
  stop(e) = false
  mode ← 'dec'
end if
if speed = 0 then
  wait()
  mode ← 'acc'
end if
if mode = 'acc' then accelerate()
if speed = max_speed then mode ←′ normal′
```

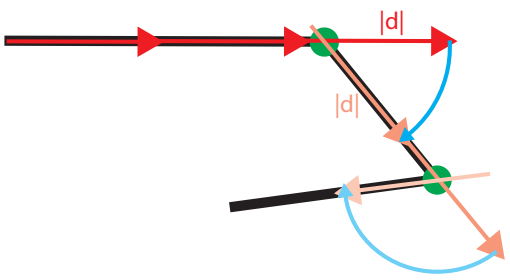

Figure 1: Principle of the overshoot correction. When the end of an edge is reached, the remainder of the vector is projected on the next edge.

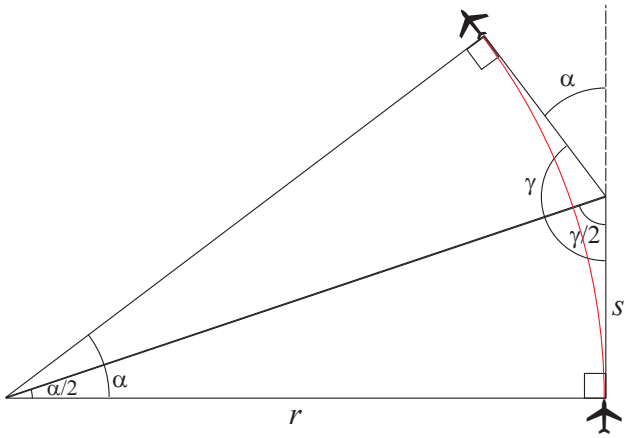

Figure 2: Illustration of the roll-radius geometry. At a constant speed v and a given turn time interval t, the plane will cover the distance s. To achieve a turn by angle α, the plane will fly a circle with radius r, rather than flying a sharp edge.

4 Simulating Platforms

The task of simulating airborne platforms is somewhat similar to the simulation of ground moving vehicles. The orbit of a platform in the current simulation is defined by sequence of points the platform has to visit. This way, the orbit forms a polygonal chain and resembles a road path. The main differences are that airborne vehicles can move freely in the three-dimensional space and that they achieve changes in direction by changing their roll angle, which is especially important for a side-looking radar as it changes the grazing angle under which the ground is seen. The roll-angle ϕ of a plane depends on the frontal speed v and the radius r of the circle flown. It can be calculated as [4]

$$\phi = \arctan\left[\frac{v^2}{rg}\right] \tag{3}$$

where g is the gravitation constant of 9.81m/s^2. Under the assumption that the vehicle flies with a constant frontal velocity, radius r can be calculated by

$$r = \frac{s}{\tan\frac{\alpha}{2}}. \tag{4}$$

Figure 2 illustrates this dependency. As the velocity is defined by $v = s/t$, we get that

$$\phi = \arctan\left[\frac{\tan\frac{\alpha}{2}}{tg}\right] \tag{5}$$

The difference between the arc length of the circle and the "sharp" way can be neglected, as it will only be 5% for angles up to 45°, while higher angles are unlikely for surveillance missons.

The process of the simulation is as follows: The user defines points (with appropriate altitudes) that have to

be visited by the plane and the velocity. At each point, the plane will turn to the reach the next point. When the last defined point is reached, the plane will return to the first and the process restarts. The overall implementation is similar to the ground target simulator, except for the missing stop events (these may only arrive for helicopters, so-called "UAV-copters") and the additional roll angle calculation.

5 Sensor Simulation

The task of the sensor simulation is to simulate those effects of a real GMTI sensor that are relevant for the tracking perfomance. This means that the simulated "ground truth" has to be modified in a way that the resulted GMTI plots contain the same errors that would occur during a radar measurement.

There are two possible ways of radar simulation: First, the *phenomenological* approach that deals with the main effects observed in radar applications and the more complicated bottom-up simulation that simulates every single step of the radar measurement process.

In the following, we will present a simulation that deals with the most important effects, based on the radar equation and some specific features of GMTI sensors. These are

- Non-detection
- Errors in measured position
- Merging of several targets
- False detection
- Ambiguities

5.1 Non-detection of targets

The problem of non-detection mainly arises from the target-sensor geometry, but also from the characteristics of the target and of the sensor themselves. First of all, an object may simply be shadowed, i. e. it is hidden by another static object, e. g. by a tunnel, a bridge or a hill. This effect is inevitable for any radar sensor and should be simulated first, as it surely prevents any measurement at all. In our simulation framework, roads can be marked as invisible ones, e. g. tunnels or deep valleys. Currently, we are working on a digital height model, that allows the calculation of visibility by a ray tracing or a z-buffer [5] algorithm. Second, targets may be outside the field of view (FoV) of the sensor, i. e., the beam from the radar will not hit the target or the reflected signal will not hit the detector. In our simulation, the field of view can be parameterized by the minimal and maximal angles that can be reached by a scan. Another problem in detection stems from the signal-to-noise ratio SNR that describes how strong the power of the reflected signal is in comparison to the noise received by the detector. From the radar equation [6] we can determine that the SNR can

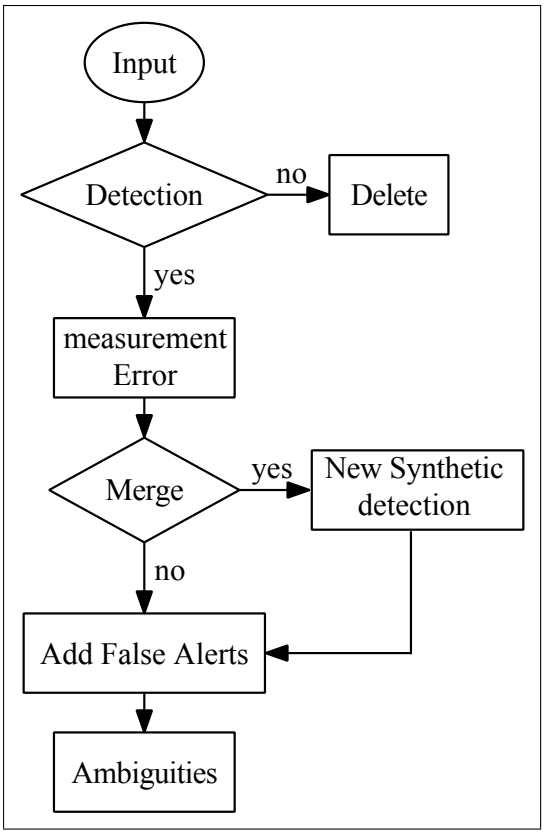

Figure 3: General work flow of the sensor simulator.

be calculated as

$$\mathrm{SNR} = \frac{\tau}{\mathrm{PRI}} \frac{P_S}{P_N} \frac{G_t G_r G_I \lambda^2 \sigma}{(4\pi)^3 R^4}. \tag{6}$$

where τ is the duration of a pulse, PRI the pulse repetition interval, P_S the emission power, $P_N = \tau^{-1} k_B T_0 F B$ the noise power, k_B the Boltzmann constant, temperature T_0, noise figure F, bandwidth B, λ the wavelength, σ the radar cross section, G_t and G_r the transmission and receiving gain and G_I the integration gain.

The transmission gain can be calculated as [7]:

$$G_t = \pi N_{\mathrm{elem.}} (\cos\phi)^{3/2} f_{\mathrm{mismatch}}(\phi - \phi_t) \tag{7}$$

with $N_{\mathrm{elem.}}$ the number of antenna elements, ϕ the beam's azimuth angle and ϕ_t the azimuth angle towards the target. The mismatch function can be empirically described as

$$f_{\mathrm{mismatch}}(\phi - \phi_t) = \exp\left[-\log(2) \left(\frac{\cos\phi - \cos\phi_t}{\mathrm{BW}} \right)^2 \right] \tag{8}$$

with BW the beamwidth that can be calculated as

$$\mathrm{BW} = k \frac{\lambda}{d_a \cos(\phi - \phi_t)} \tag{9}$$

where d_a is the diameter of the antenna aperture and $k \approx 1$ an antenna-specific parameter. If no STAP

(Space-time adaptive processing) is done, then the transmission gain will be equal to the receiving gain. For STAP, the receiving gain can be expressed as

$$G_r^{\mathrm{STAP}} = G_r \left(1 - \exp\left[-\log(2)\left(\frac{\dot{r} - \dot{r}_{\mathrm{MLC}}}{v_{\mathrm{MDV}}}\right)^2\right]\right). \tag{10}$$

With $\dot{r}$ the radial velocity of the target, $\dot{r}_{\mathrm{MLC}}$ (MLC = main lobe clutter) the radial velocity towards the ground around the target and v_{MDV} the minimal detectable velocity. The radar cross section σ can be parameterized as a constant for an object or can be read in from a file containing the radar cross section for distinct aspect angles. In our tests, we used RCS-files that were created using a CAD model and an external RCS simulation program.

Finally, the detection probability can be calculated using the well-known Swerling-I-Case

$$P_D = P_F^{\frac{1}{1+\mathrm{SNR}}} \tag{11}$$

by testing the resulting detection probability against a generated pseudo-random number, the decision between detection and non-detection is made. Non-detected targets are then deleted from the list.

5.2 Measurement errors

Next, the errors in the actual measurement (errors of the actual positions) have to be estimated. A simple approach would be to just blur the "true" values in azimuth and distance. However, the error parameters are not constant. In an ideal case, the azimuth errors depend on the Beamwidth, a parameter γ_ϕ that depends on the antenna configuration and the SNR [8]:

$$\sigma_\phi = \gamma_\phi \frac{\mathrm{BW}}{\sqrt{\mathrm{SNR}}}. \tag{12}$$

In practice, the measurement accuracy is bounded and will not rise infinitely for strong targets. A reasonable lower bound for the error is

$$\sigma_\phi = \kappa_\phi \frac{\mathrm{BW}}{10} \tag{13}$$

where κ_ϕ typically lies between 1 and 2.

Similiar considerations hold for the range error

$$\sigma_r = \min\left(\gamma_r \frac{\Delta r}{\sqrt{\mathrm{SNR}}}, \kappa_r \frac{\Delta r}{\sqrt{\mathrm{SNR}}}\right). \tag{14}$$

with $\Delta r = k\frac{c}{2B}$ (c the speed of light).

For the range rate $\dot{r}$, the error can be calculated using the doppler frequency f_D. It holds that

$$\Delta\dot{r} = \Delta f_d \lambda/2 \tag{15}$$

where λ is the wavelength. The size of a doppler cell can be calculated by

$$\Delta f_d = k_D \frac{1}{\mathrm{CPI}} \tag{16}$$

where CPI is the coherent integration time. So the standard deviation of the range error is

$$\sigma_{\dot{r}} = \min\left(\gamma_{\dot{r}} \frac{\Delta f_D}{\sqrt{\mathrm{SNR}}}\frac{\lambda}{2}, k_{f_d}\frac{\Delta f_D}{10}\frac{\lambda}{2}\right) \tag{17}$$

The measurement errors are simulated by a random number generator according to Gaussian distributions using $\sigma_\phi, r, \dot{r}$ as standard deviations and zero mean.

5.3 Merging of targets

Targets close together may be subject to merging as the receiver will only receive one merged reflection signal. The probability for two targets i, j to be merged is modeled as [9]

$$P_u(i,j) = \exp\left(-\frac{1}{2} d^\top R_u^{-1} d\right) \tag{18}$$

with

$$d = \begin{pmatrix} r_i - r_j \\ \phi_i - \phi_k \end{pmatrix}, R_u = A/(2\log 2), A = \begin{pmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\phi^2 \end{pmatrix}$$

It is easy to see that $P_u(i,j) = P_u(j,i)$. The merging simulation process is follows: In a first step, the merging probabilities $P_u(i,j)$ for all pairs $(i,j),\ i < j$ are calculated. The results are stored in a matrix. Then, all elements with a merging probability exceeding a chosen threshold are merged. In the simulations, a sparse field of view with few objects is assumed, and merging is stopped after one iteration. In a denser scenario, merging could be iterated until there is no object with higher merging probability left. In that case, it is advisable to partition the elements according to their position before merging as the runtime will quadratically increase with the number of targets to be considered.

After merging, a synthetic target with a new position, using the geometrical average of the ancestor position, is created. The ancestors are then removed from the target list.

5.4 Generation of False Detections

False alerts occur when the detector measures a signal that is seen as a target but does not correspond to a target in reality. The reasons for false detections may be due to the noise in the detector itself, clutter from the ground or quasi-moving objects like high-speed rain clouds, trees, waves or windmill power-plants. In the current version of the sensor simulator, areas with a parameterizable clutter density can be defined. These areas can be circular, ring- or polygonal-shaped. Additionally, a constant clutter rate for the whole FoV can be defined. Thus, a constant false alarm rate as well as areas with a higher clutter rate, like lakes, windmills etc. can be simulated.

ϕ_d	Antenna orientation against flight direction
P_{FA}	Constant false alert density
SNR_0	Standard Signal to noise ratio
R_0	Standard Radius for SNR_0
mdv	Minimal detectable velocity
ν	Pulse frequency
N_s	Number of antenna elements
G_R	Antenna gain
d_a	Aperture diameter
P_S	Emmission power
G_I	Integration gain
τ	Pulse duration
PRI	Pulse repetition interval
CPI	Coherent integration time
T_0	Antenna temperature
F	Noise figure
B	Bandwidth
k	Beamwidth factor
κ	Resolution factor
$\alpha_{\min}, \alpha_{\max}$	Azimuth limits
$\epsilon_{\min}, \epsilon_{\max}$	Elevation limits
$r_{\min}, r_{\max}$	Range limits
$\kappa_{\phi,r,f_D}, \gamma_{\phi,r,\dot{r}}$	Antenna configuration parameters

Table 1: Antenna parameters used in the simulation of the GMTI process

5.5 Ambiguities

The simulator is also able to simulate ambiguities. These arise due to the fact that consecutive pulses may not be distinguished, thus it is ambiguous whether the run-time of a signal is caused by traveling time or by traveling length. However, in most GMTI applications, these ambiguities can be identified as they will lead to implausibly results, like targets that lie below the surface level.

An overview of all parameters used in the sensor simulation can be found in table 1.

5.6 Implementation

The system consists of two main parts: The simulation core and the graphical user interface (GUI). The core of the simulation was implemented in JAVA, while the GUI and parts of the map-loading routine were programmed in MATLAB.

A great advantage of this approach is that the system will stay platform-independent, as long as there exists a Java Virtual Machine and a Matlab implementation for the computer platform. The flexibility is improved by the usage of a object-oriented language. For instance, interfaces serve for the input to the sensor simulator that can be used to extend the system for any input source, like databases, text files, XML etc. The only condition is that it will deliver an array of target or sensor descriptions. Similar conditions hold for output formats. This way, we plan to develop an on line simulator soon that will take the input from a flight simulator that is manually controlled by an operator (pilot), then simulates road targets and the effects of the GMTI measurement process. With this approach, only the platform simulator module has to be replaced, while the rest of the program can be left unchanged.

The GUI was implemented in Matlab, as it allows a rather simple display of map data and visualization of results. It is also able to read VMAP-0-data directly (in connection with the Mapping Toolbox). Another advantage is that the interconnection between both programming languages is easy to realize.

5.7 Performance issues

The system was tested under Windows as well as under Linux. The runtime is mainly dominated by the clutter density as the number of false alerts drastically increases if the Field of View is large (for performance reason, clutter will not be generated outside the FoV as it is uninteresting). The runtime generally increases linearly with the time interval to be simulated and may – in the worst case – increase quadratically with the number of road targets due to the calculation of merging probabilities. By pre-partition, the run-time will only increase quadratically according to the number of targets per cell and will most likely be practically linear. On a state-of-art PC, the calculation time for scenarios spanning an interval of half an hour only took half a minute and will be sufficient for real-time simulation.

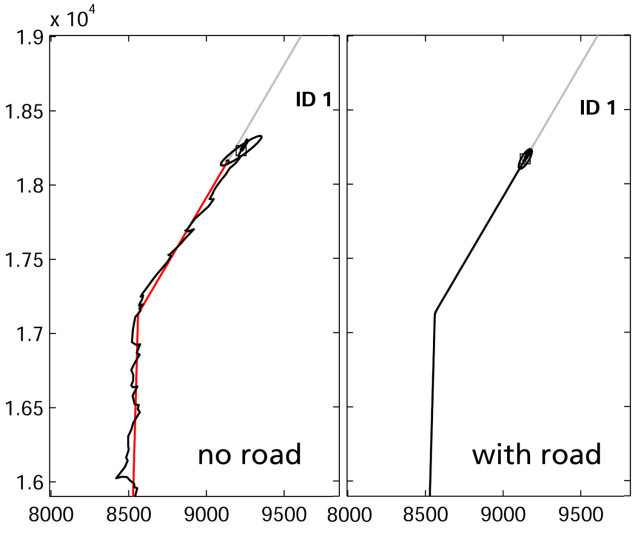

Figure 4: Track results (black solid line) of single target scenario at final revisit without (left) and with (right) exploitation of road-map information (grey solid line).

6 Results

Several single and multi target tracking algorithms (PDAF, JPDAF, NN-Kalman, GM-CPHD), which are all based on the BAYESian formalism [10], were implemented into the presented simulation framework in order to test and compare the performance in individual simulation scenarios. In addition, these algorithms were also capable of exploiting the implemented road network information [11].

To test the accuracy of the tracker results, the tracker component will output a graphical representation of the track as well as the ground truth and several defect sizes like error in range rate (radial velocity), absolute velocity and position. It turned out that the simulation was a great advantage in competition with other methods like usage of pre-recorded data. We were able to simulate special situations, like crossing events (that bear the risk of identity switches), stop events, take-over maneuvers, off-road vehicles etc. In conduction of the tests, the feasibility of tracking algorithms using road map date could be shown.

As an illustration, the track results for a single road target based on the GM-CPHD algorithm are presented in Fig. 4 with and without taking advantage of the road-map information. The corresponding track errors in position, velocity and range-rate for a single run related to the ground truth information are shown in Fig. 5. A performance evaluation of the mentioned tracking algorithms can be found in [12].

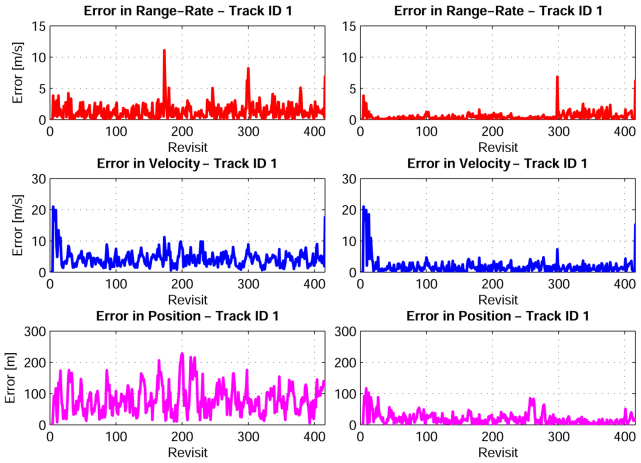

Figure 5: Track errors wrt. ground truth information for the single target scenario, see Fig. 4, without (left column) and with (right column) exploitation of road-map information.

7 Conclusion and Outlook

In this paper we have presented a new framework for simulating GMTI scenarios, comprising the whole detection chain from the generation of ground targets, over the simulation of flight maneuvers till the detection by a GMTI sensor. We have given a simple implementation for ground target simulation using digital road maps. We have shown how this technique can be transformed to the simulation of flying objects. We have investigated the question how flight maneuvers will affect the radar measurements due to the roll angle and how this effect can be calculated depending on the flight path. Then we proposed which phenomens ought to be considered for an empiric and realistic simulation of GMTI radar and stated their calculation. We then described our way of implementation. Finally, we gave first tracker estimation results we obtained using the simulator.

The next step for GMTI tracker performance estimation is the definition of criteria that describe the quality of track data. The quantities shown in fig. 5 give a first idea of these criterias, but have to be investigated in depth. For instance, the loss of track identities after stop events is a great problem for GMTI trackers, and a quantification for this quality must be developed. In order to get comparable results, test scenarios for GMTI should be designed, showing realistic challenges from the practice. After all, the results from the simulation must be compared with real data as far as possible.

References

[1] R. Klemm, *Principles of Space-Time Adaptive Processing*. London, UK: Institution of Engineering and Technology, 2002.

[2] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[3] D. E. Knuth, *Art of Computer Programming, Volume 1: Fundamental Algorithms (3rd Edition)*, 3rd ed. Addison-Wesley Professional, 7 1997.

[4] M. Häge and E. Ruthotto, "Emitterlokalisierung durch Peil- und Bilddaten-Fusion," Fraunhofer FKIE, Wachtberg, Germany, Tech. Rep. 188, 2010.

[5] J. Bittner and P. Wonka, "Visibility in computer graphics," *Journal of Environmental Planning*, vol. 30, pp. 729–756, 2003.

[6] M. I. Skolnik, *Introduction to Radar Systems*. Singapore: McGraw-Hill Book Company, 1980.

[7] W.-D. Wirth, *Radar Techniques Using Array Antennas*. London, UK: The Institution of Engineering and Technology, 2001.

[8] S. Blackman and R. Popoli, *Design and Analysis of Modern Tracking Systems*. Boston MA, USA: Artech House Publishers, 1999.

[9] W. Koch and G. Van Keuk, "Multiple hypothesis track maintenance with possibly unresolved measurements," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 33, no. 3, pp. 883 – 892, july 1997.

[10] Y. Bar-Shalom and X.-R. Li, "Estimation and tracking: Principles, techniques, and software," *Artech House, Boston*, 1993.

[11] M. Ulmke and W. Koch, "Road-map assisted ground moving target tracking," *IEEE Trans. on Aerospace and Electronic Systems*, vol. 42(4), pp. 1264–1274, October 2006.

[12] D. Svensson, J. Wintenby, and L. Svensson, "Performance evaluation of MHT and GM-CPHD in a ground target tracking scenario," in *Proceedings of the 12th International Conference on Information Fusion, 2009*, 6-9 2009, pp. 300–307.

[13] W. Koch, J. Koller, and M. Ulmke, "Ground target tracking and road map extraction," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 61, no. 3-4, pp. 197–208, 2006.

[14] J. Koller and M. Ulmke, "Data fusion for ground moving target tracking," in *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems, 2006*, sept. 2006, pp. 217–224.