Simulation of sensors of advanced driver assistance systems (ADAS) using UM Sensors and UM Video Flow modules

The features of the Universal Mechanism software package for modeling sensors in driver assistance systems (ADAS) and exporting video data are considered

2024

Contents

1. Simulation of sensors in advanced driver-assistance systems

Advanced driver-assistance systems (**ADAS**, Advanced Driver-Assistance Systems) is a set of devices, such as sensors and on-board computers, and algorithms that help the vehicle driver to make decisions for safe and easy driving. Such systems are able to distinguish and identify different kinds of objects (pedestrian, car, lane lines, road signs, cross-road, etc.) and warn driver when quickly approaching a pedestrian or car, crossing the lane line and about the possibility of meeting other dangerous situations on the road. The effective implementation of these features depends on the properties of the sensors used.

UM Sensors module implements the following sensors:

- GPS sensor;
- ray sensor;
- beacon;
- camera sensor.

1.1. Samples

Two configurations with the preset sensors are prepared for the {UM Data}\SAMPLES\Automotive\TruckTrailer model. These two configurations use scenes created with the help of **UM Scene** module. **UM Scene** requires the **collection of standard 3D objects**. This collection is not included into installation package of Universal Mechanism and should be install separately.

Download the collection using the link below and install on your computer: http://www.universalmechanism.com/download/umscenecollection.exe.

1.1.1. Samples with ray sensors and beacons

1. Load the {UM Data}\SAMPLES\Automotive\TruckTrailer model**.**
2. Load one of two prepared configurations using the [**File | Load configuration**] menu item, see Figure 1.1. The model includes two configurations with sensors: [**Industrial zone Trajectory 1 + Ray sensor**] and [**Industrial zone Trajectory 1 + Beacon**]. The first one includes ray sensor. The second one includes beacons. General view of the main window of **UM Simulation** after loading the [**Industrial zone Trajectory 1 + Ray sensor**] configuration is shown in Figure 1.2

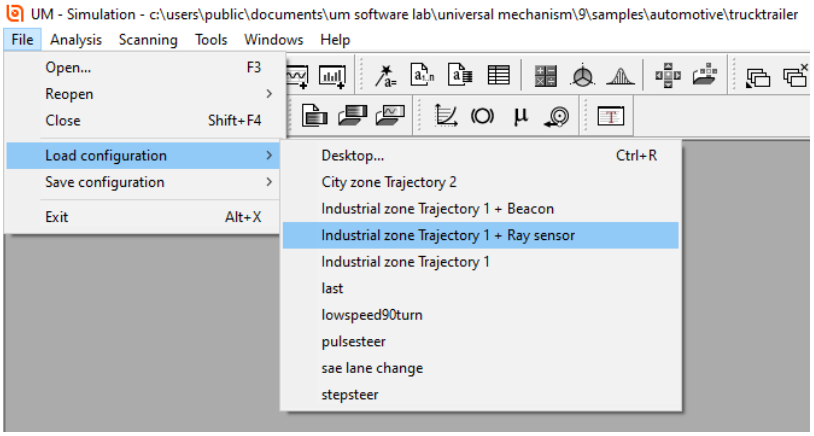

Figure 1.1. Loading configuration

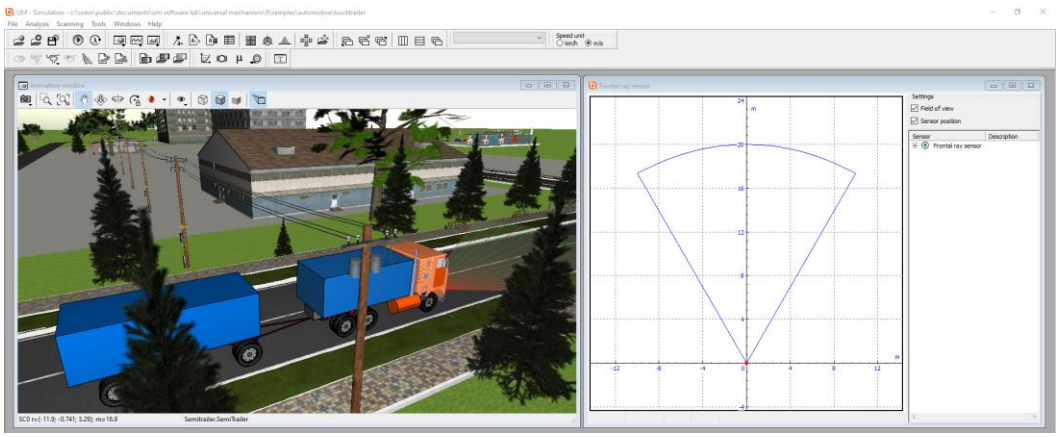

Figure 1.2. UM Simulation main window after loading
the [**Industrial zone Trajectory 1 + Ray sensor**] configuration

1.1.2. Samples with camera sensor

Sample 1

1. Load the {UM Data}\SAMPLES\Automotive\TruckTrailer model**.** With the help of the **File | Load configuration** menu item load the **[Industrial zone Trajectory 1 + Camera]** configuration, see Figure 1.3. This configuration includes camera sensor installed in the top of the windshield and directed towards. General view of the main window of **UM Simulation** after loading the [**Industrial zone Trajectory 1 + Camera**] configuration is shown in Figure 1.4
2. Run simulation. You will see video in the camera monitor.

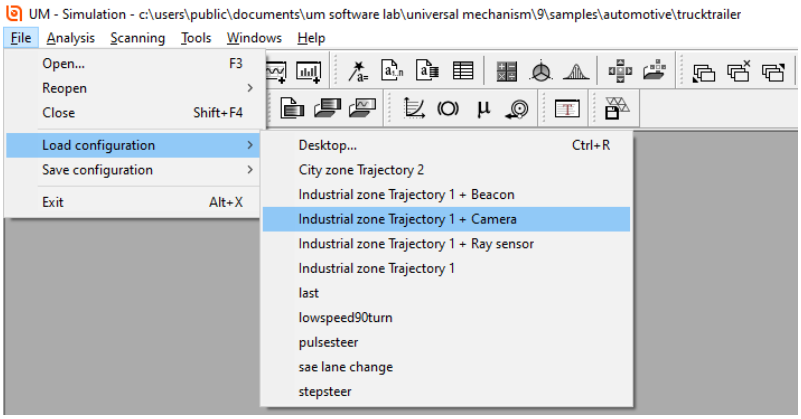

Figure 1.3. Loading configuration

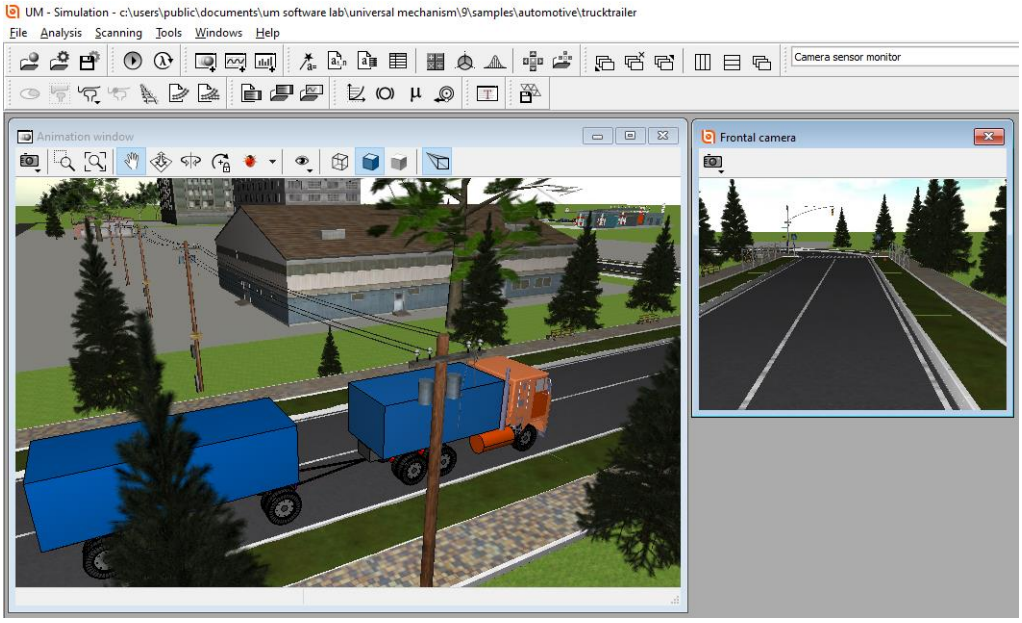

Figure 1.4. UM Simulation main window after loading
the [**Industrial zone Trajectory 1 + Camera**] configuration

3. Now let us consider steps for exporting the video flow from UM to Matlab. Run Matlab. Load the {UM Data}\Video Flow\UMVideoFlowTestReceiver.m file into Matlab and run in under Matlab environment with the help of **Tool panel | EDITOR | Run** button or **F5**

key. Code implemented in the **UMVideoFlowTestReceiver.m** file comes to waiting for data from the external sender via TCP/IP protocol.

4. Come back to **UM Simulation**. Select the **Tools | Sensors** menu item. **Wizard of sensors** window appears, see Figure 1.5. Double click on the **Frontal camera** list item. **Camera setting** window appears, see Figure 1.6.

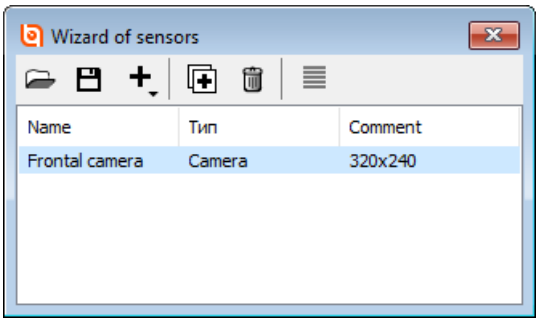

Figure 1.5. Wizard of sensors

5. In the **Camera setting** window select the **Export video** tab, turn on the **Export video** check box, click the **Connect** button and then close the window by the **OK** button, see Figure 1.6.

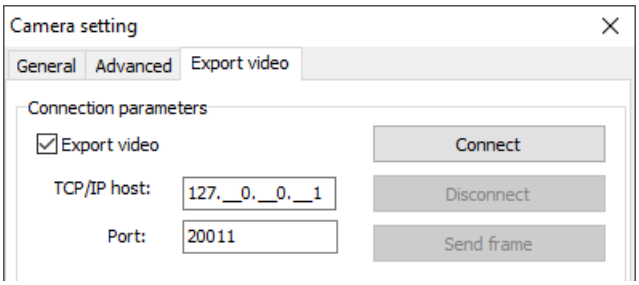

Figure 1.6. Camera setting / Export video

After clicking the **Connect** button the Matlab code from the **UMVideoFlowTestReceiver.m** file will output the following listing in the Matlab **Command Window**:

```
Connected via TCP/IP {Date} {Time}, Host: 127.0.0.1, Port: 20011

Camera data:
 Focal length: 7.5 mm
 Sensor size: 6.4 x 4.8 mm
 Image type: 8-bit color RGB
 Image resolution: 320 x 240
 FoV: 46.2 x 35.5 deg
 FPS: 25
```

6. Run simulation in Universal Mechanism with the help of the **Analysis | Simulation** menu item and then the **Integration** button. Matlab starts receiving frames from the camera sensor. Wait for the end of the simulation. After that the video flow will be stopped and the TPC/IP connection will be disconnected.

Sample 2

1. Load the {UM Data}\SAMPLES\Automotive\Audi A3 model.
2. With the help of the **File | Load configuration** menu item load the [**Road signs and marking recognition with camera**]. This configuration includes camera sensor installed in the top of the windshield and directed towards. Vehicle speed is 60 km/h and relatively high irregularities of cobbles result in rather jerky movement of the car body.
3. Change the camera settings. Select the **Tools | Sensors** menu item and open settings for the **Frontal camera**, see Figure 1.10. **Camera setting** window appears. Select the Advanced tab sheet and change the camera settings according to the given below in Figure 1.10.
4. Close the window by clicking **OK**.

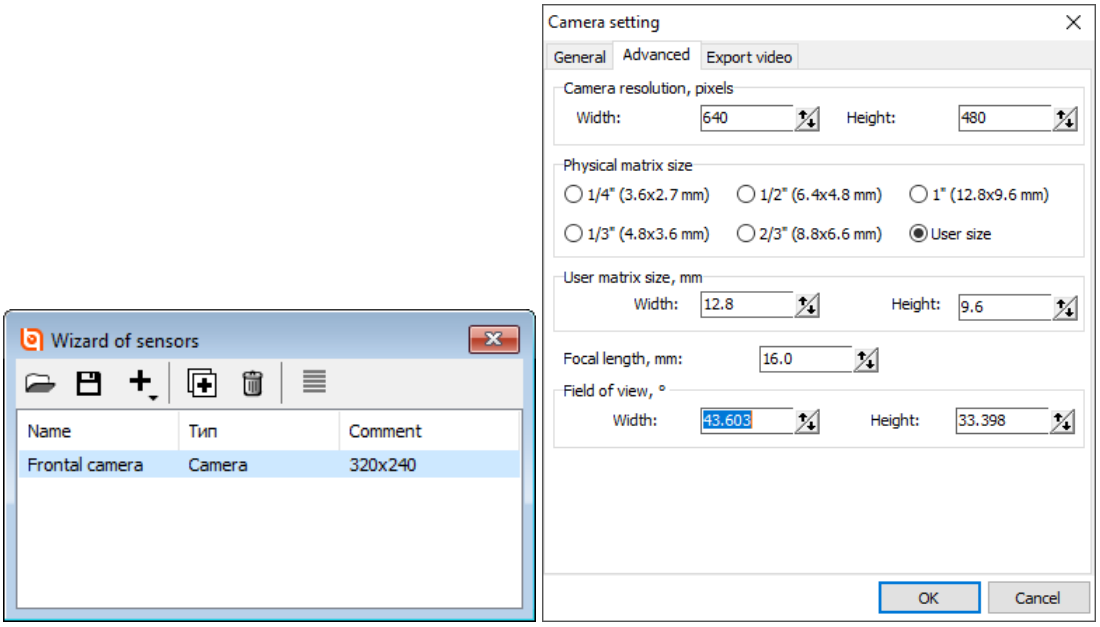

Figure 1.7. Advanced camera settings

5. Run or switch to Matlab and run the **UMVideoFlowTestReceiver.m** file.
6. Come back to **UM Simulation**. Open the **Camera settings** dialog and select the **Export video** tab sheet, see Figure 1.8. Turn on the **Export video** check box and click **Connect**. Finally click the **Send frame** button to send the single frame from UM to Matlab.

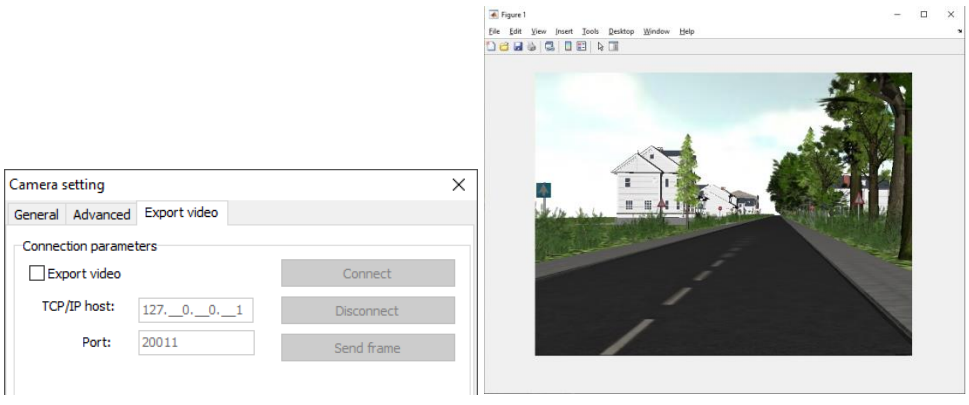

Figure 1.8. Sending and receiving a single frame

Matlab window with the received frame appears, Figure 1.8. After that UM will disconnect the TCP/IP connection and code from the **UMVideoFlowTestReceiver.m** file comes to the end. If the test sending frame finished successfully, you can see the frame under Matlab environment and the picture is correct, you can run simulation of the model and send the video flow as it described in the Sample 1 above.

Some more details about working with the camera sensor is described the Sect. 1.7 "*Camera sensor*", page 1-22.

1.2. GPS sensor

1.2.1. Basic definitions

GPS is Global Position System. GPS subsystems are shown in Figure 1.9[1]. Using GPS system requires special devices, such as a GPS sensor.

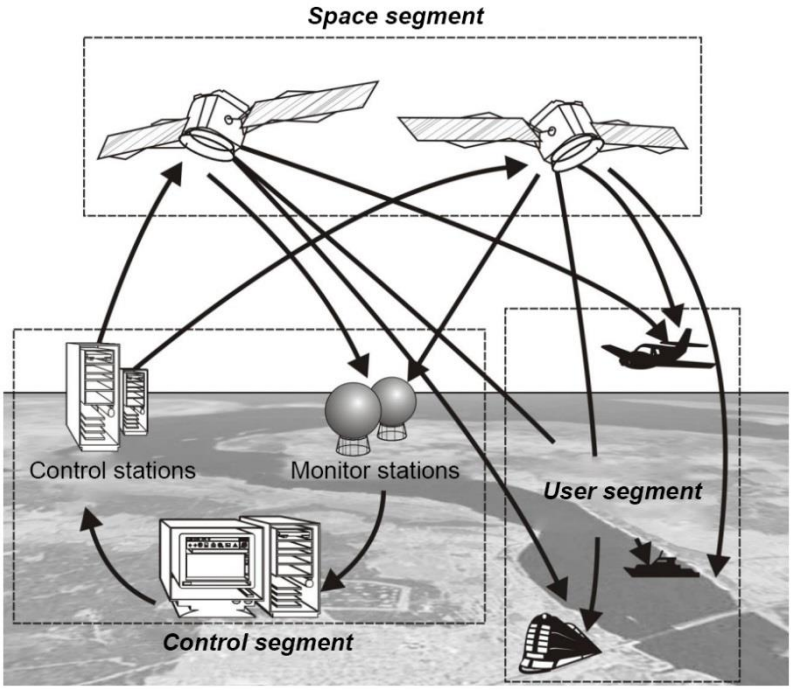

Figure 1.9. Elements of global positioning system elements (GPS, GNSS)

GPS sensor (receiver) is a radio receiver that receives radio signals from a satellite navigation system. Using radio signals from GPS satellites, user receivers determine the current location coordinates, time and speed of an object steadily and accurately at any point on the Earth's surface, at any time of the day and in any weather. The use of GPS sensors helps controlling the movement of road vehicles, ensuring road safety. A prerequisite for the successful use of a GPS sensor is a connection to a satellite navigation system.

Global Navigation Satellite System, **GNSS** consists of the space and control segments. Currently, only two satellite systems provide complete and uninterrupted coverage of the globe — GPS and GLONASS. These systems are controlled by the ministries of defense of the USA and the Russian Federation, respectively.

For mathematical processing of the signals received from GPS and GLONASS, geodetic datums WGS-84 and PZ-90.11 are used, respectively.

---

[1] Aeronavigation (textbook), Saint Petersburg State University of Civil Aviation is an aviation school in Saint Petersburg, Russia. Chapter by Yu. I. Liberman, A.V. Lipin, Yu. N. Saraiskiy, o be published. URL: http://uaecis.com/files/13/teoria%20SNS.pdf (accessed 25.03.2020).

WGS-84 (**World Geodetic System 1984**). Currently, the 6-th version of the reference frame is used. It is referred to epoch 2005.0, denoted as WGS-84(G1762) and agreed to the reference frame ITRF-2008 at submillimeter level.

PZ-90.11 – the state geocentric coordinate system "Earth parameters of 1990" (PZ-90), referred to the epoch of 2010.0.

Epoch is a term introduced due to the influence of the precession and nutation of the Earth's rotation axis on the accuracy of determining location of an object.

Location is a term which only makes sense in the selected frame of reference. In traffic simulation geodetic and topocentric frames of reference are the most interesting.

1.2.2. Geodetic reference frame

The real shape of the Earth can be closely approximated by an ellipsoid of revolution (reference spheroid), see Figure 1.10.

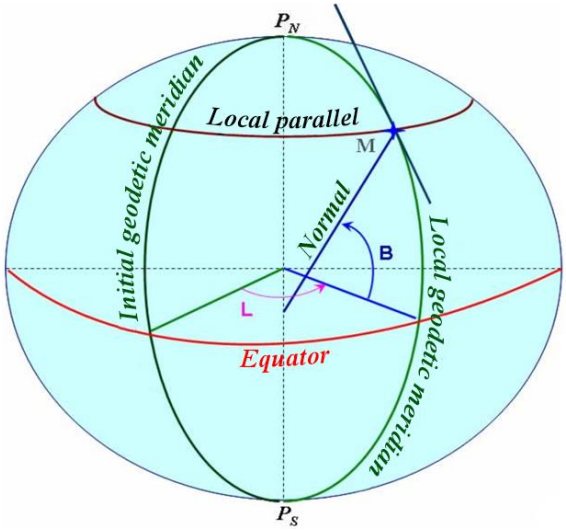

Figure 1.10. Geometric model of the Earth

Geodetic coordinates define the point position at the surface of the reference spheroid. In this reference frame the latitude (B) and the longitude (L) are the coordinates of the point M; meridians and parallels are the datum lines.

Meridians are the lines of intersection of the reference spheroid by the planes passing through its minor axis. Parallels are the lines of intersection of the reference spheroid by the planes perpendicular to its minor axis. The **equator** is a parallel whose plane passes through the center of the spheroid.

Latitude is measured from the equator to the north and south from 0° to 90° and is called north latitude and south latitude, respectively. North latitude is considered positive, and south latitude is negative.

The Greenwich meridian passing through the Greenwich Observatory in the vicinity of London was taken as the **initial meridian**.

Geodetic longitude is the dihedral angle made up by the plane of the initial meridian and the meridian of the given point, see Figure 1.10. Longitude is measured from the initial meridian

east and west from 0 to 180°. East longitude is considered positive, and west longitude is negative.

Geodetic altitude of a point **M** is the distance from this point to the surface of the reference spheroid, which corresponds to the sea level.

In solving engineering problems the transition from geodetic frame of reference to a simpler rectangular (Cartesian) frame of reference is preferable. For that purpose the topocentric coordinates are used.

1.2.3. Horizontal topocentric reference frame

In the **topocentric reference frame** the position of the observed object is defined from the observer point at the Earth's surface (point M in Figure 1.11).

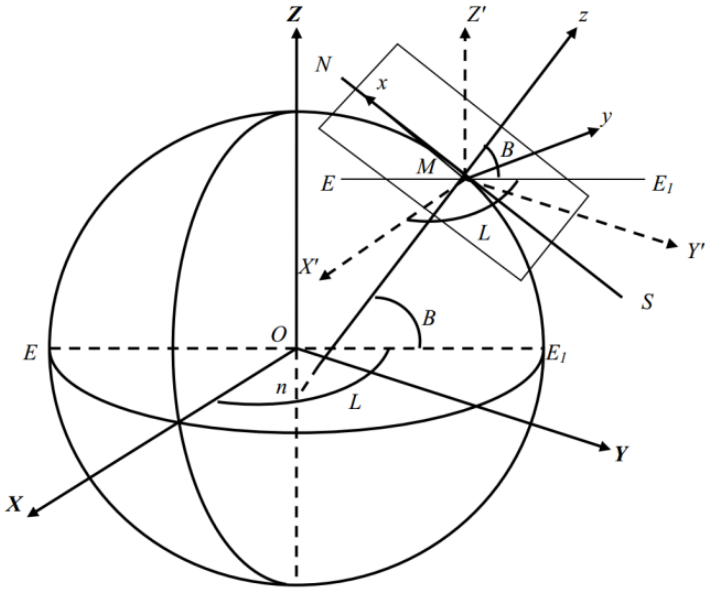

Figure 1.11. Topocentric reference frame (x, y, z) and its relationship with rectangular geocentric (X, Y, Z) and geothetic (B, L, H) reference frames

In the topocentric reference frame (x, y, z) axis z is aligned with the normal to the spheroid surface; axes x and y are located in the horizontal plane. The x axis lies in the plane of the geodetic meridian and is directed to the north, and the y axis is directed to the east. We note that the reference frame considered is left-handed.

1.2.4. Position of the UM global reference frame in geodetic reference frame

In UM software Cartesian right-handed reference frame (SC0) is used, see Figure 1.12. This base inertial reference frame, in which the car motion is considered, satisfies the following standard requirements:

- axis Z is vertical; axis X corresponds to the position of the longitudinal axis of the vehicle at its ideal position at the time moment when the motion begins;
- origin of SC0 lies at the ideal road level.

The position of SC0 in UM is set in the **General sensors settings** window (**Tools | Sensors** menu item, then **General sensors settings**), see Figure 1.13. The position of the UM SC0 origin is specified by **latitude**, **longitude** and **elevation**. The Z axis of the global RF coincides with the normal to the tangent plane of the spheroid; the X and Y axes are located in this tangent plane. The angle between the X axis of the global SC0 and the direction to the north is specified by the **Azimuth**.

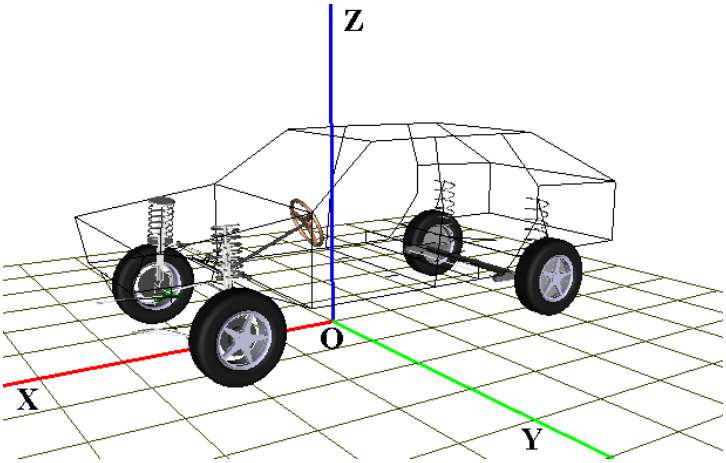

Figure 1.12. Base system of coordinates (SC0)

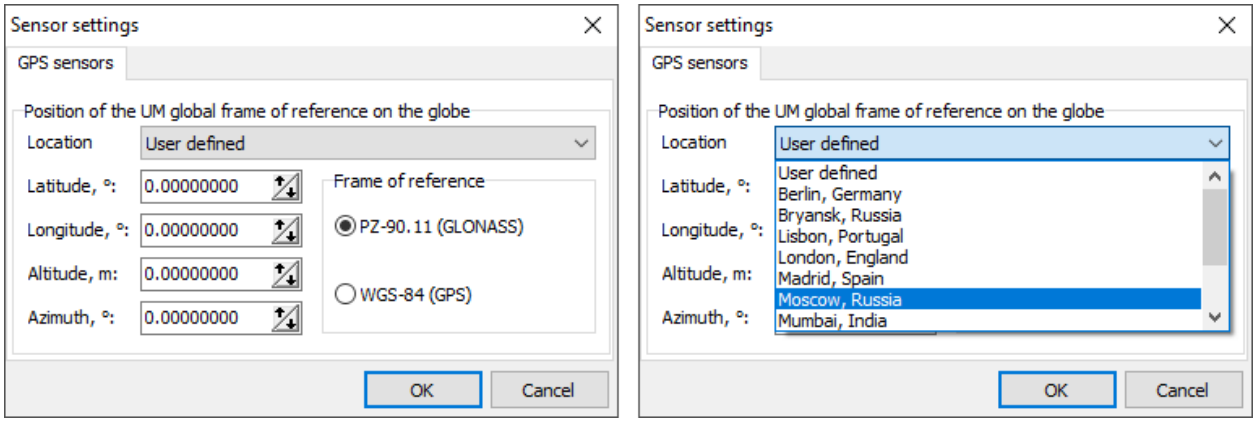

Figure 1.13. Position of the UM global SC0 in the Earth-wide reference frame

To simplify the input of geodetic coordinates, the concept of geoobject is introduced. Geoobject is the object the global reference frame origin is associated with. When the object from the drop-down list is selected, the fields with its geodetic coordinates are filled in automatically.

1.2.5. GPS sensor settings

Consider the procedure of setting GPS sensor parameters when preparing models of ADAS in UM. A typical scenario of adding a sensor to the model is the same for any type of sensor with minor changes. In this section, adding a sensor is considered on the example with a GPS sensor.

The sensor setting begins with selecting the main menu item **Tools | Sensors** and opening the **Wizard of sensors window**, see Figure 1.14. The numbers in Figure 1.14 denote the toolbar buttons that allow performing the following actions:

1 – load sensor configuration from file;
2 – save sensor configuration to file;
3 – add sensor;
4 – duplicate current sensor;
5 – delete selected sensors;
6 – display common sensor settings dialog.

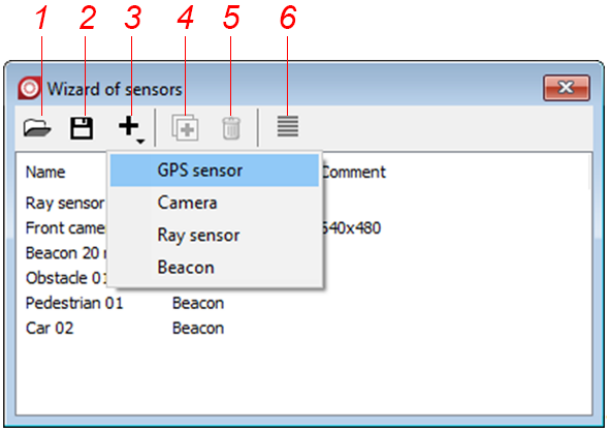

Figure 1.14. Sensors settings window

Left-click on the "+" and select **GPS sensor** in the drop-down list, the **GPS sensor setting dialog appears (**Figure 1.15). You can configure the GPS sensor by changing the following parameters. Like other sensors, the GPS sensor is identified by a name, which must be specified in the **Name** field. The **Comment** field is used to provide additional information, if needed.

Any sensor must be attached to an object of a simulated road scene. This object is selected from the drop-down list **Attached to**. The sensor coordinates in the local reference frame of the object are set in the **Sensor Position** group box.

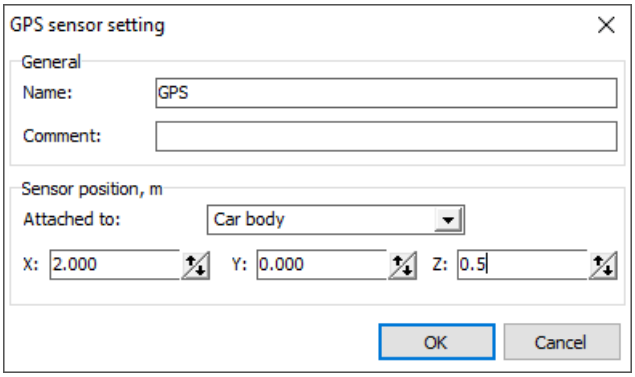

Figure 1.15. GPS Sensors setting

1.3. Ray sensors

New technologies for driver assistance systems include ensuring safe highway driving, assistance in various traffic conditions, such as heavy traffic, stops and starts when in traffic jams, poor lighting conditions, when parking and changing lanes. For this purpose, radar and laser sensors are installed on the vehicle.

1.3.1. Physical sensors operation principles

Radar (radio detection and ranging) emits electromagnetic waves to the object and receives a reflected signal – an echo; and the distance to the object is calculated based on the time of echo return. Radars are characterized by long range, wide field of view, high resolution, and ability to work in difficult weather conditions. However, due to their high price, radars are usually used in cars of the upper market segment.

Lidar (LiDAR – light detection and ranging) is a sensor that is completely similar to radar, except that an infrared laser beam is used instead of radio waves.

According to experts from the Taiwan Association for the Development of Industry and Technology in Photonics (PIDA), the scope of LIDAR applications will grow rapidly, and business opportunities in the automotive industry are promising.

Like radars, LIDARs are characterized by long range (150–200 m), wide field of view, high accuracy (1%), resolution (0.1 m), and significant resistance to pollution and poor lighting conditions.

The principle of operation of the optoelectronic measuring technology of the LIDAR is based on calculating the distance by measuring the time required for the laser beam to reach the target and return back to the receiver. The sensor sends a short light impulse in the infrared frequency range. The impulse reflected from the target is recorded by the sensor and transmitted to the associated electronic unit. The on-board computer performs its ranking, that is, calculates the distance, speed and relative position of the vehicle in front of the controlled car. Based on the data obtained, it is possible to automatically adapt the car's movement to the conditions of rapidly changing traffic by using the braking system or by changing the engine control parameters.

1.3.2. Ray sensor mathematical model

The implemented mathematical model of the ray sensor allows the user to expand their general knowledge on the active scanning sensor. It is not limited to the specifics of a particular technology, which is applicable, for example, only for LIDARs or only for laser scanners. It implements the general principles of operation of such devices and, therefore, it can be used to verify the standard technical characteristics of any active scanning sensor at the system level.

The ray sensor model specification is based on simple concepts, such as search direction, field of view, detection method and the number of emitted rays. More complex concepts, such as range measurement accuracy, range resolution, angular accuracy, angular resolution or transparency of the environment and others are not included in the model.

1.3.3. Configuring ray sensors

Configuring ray sensors is similar to configuring GPS sensors (see 1.2.5). We now consider the configuration steps focusing on the specific details of the ray sensors.

For adding a ray sensor to a vehicle model in **UM Simulation** use the **Tools | Sensors** and open the **Sensor Settings** dialog, see Figure 1.16. The tools available in this window are similar to those used to configure GPS sensors.

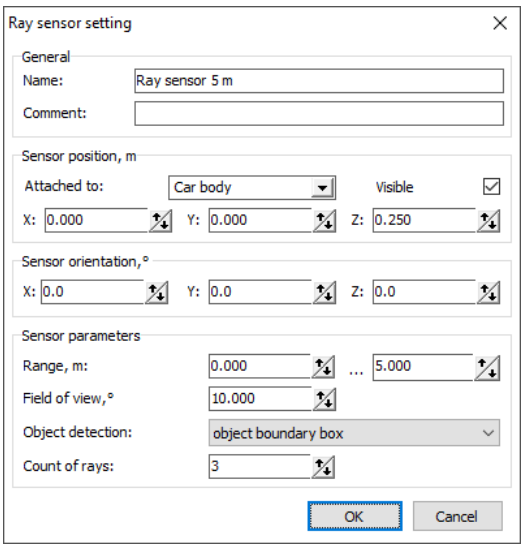

Figure 1.16. Ray sensor settings

In the group box **Sensor orientation** the direction of the sensor rays is specified by the rotation angles relative to the coordinate axes (see Figure 1.17).

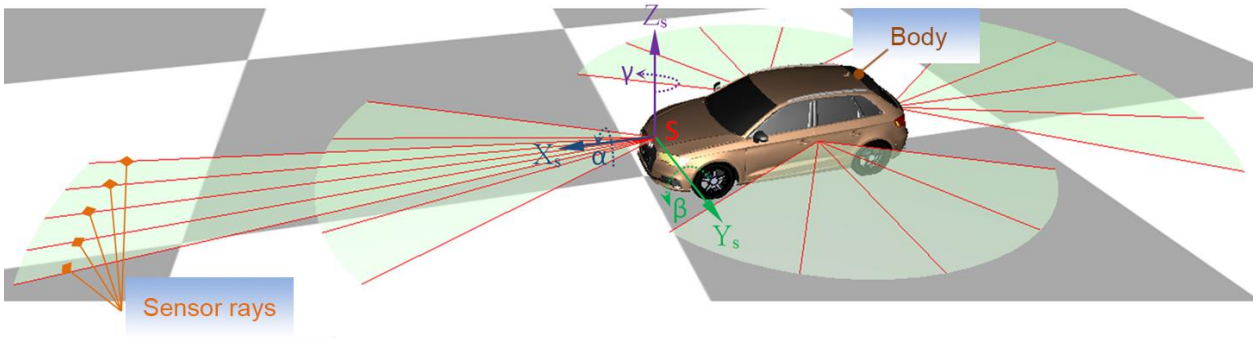

Figure 1.17. Setting ray sensor scanning direction

The area where the sensor can detect the target is determined by the parameters **Range** (minimum and maximum distance to the target in meters) and **Field of view** (in angular degrees) relative to the specified target search direction. These fields are combined in the group box **Sensor Parameters.** This group box also contains the fields for selecting the target detection method and the number of rays of the sensor.

In the **Object detection** field you can specify the target detection method, depending on the simulation goals, see Figure 1.18. The differences between the three object detection methods for detecting objects are shown in Figure 1.19.

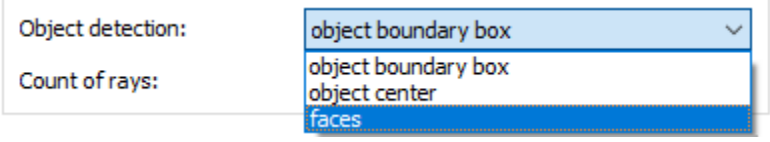

Figure 1.18. Specifying the target detection method

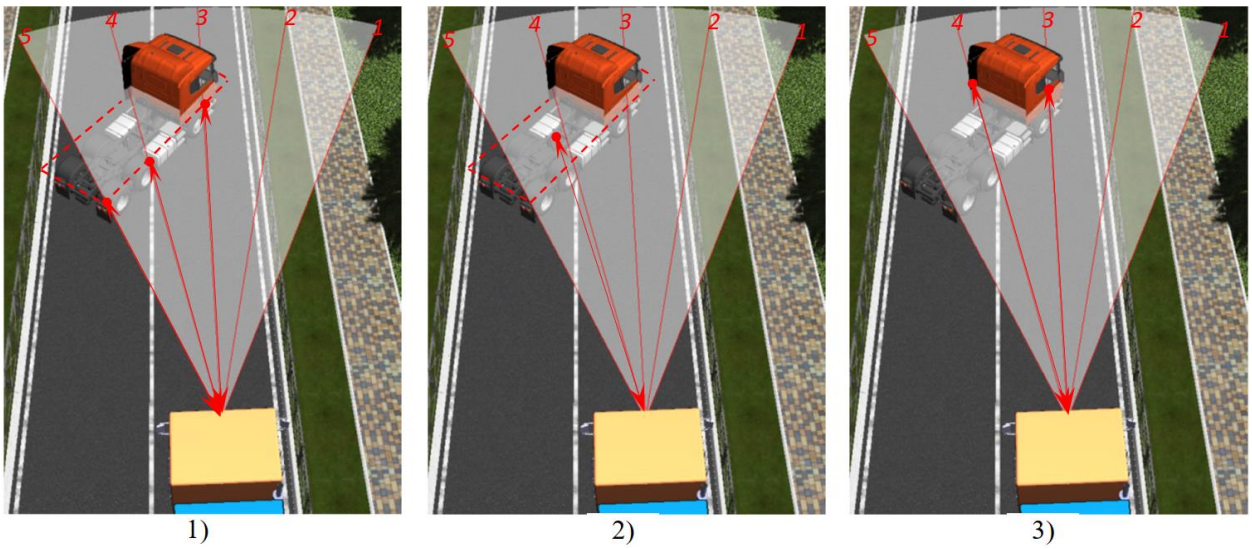

Figure 1.19. Objects detection methods.
1) by object dimensions; 2) to the object center; 3) by details.
Numbers of sensor rays are shown in red

In the first case (object detection by its **boundary box**) each ray returns the distance from the point of its intersection with the object dimensions to the point of the sensor position. These are rays 3, 4 and 5.

In the second case (**object center**) rays 3, 4 and 5 return the distance from the geometric center of the object to the point of the sensor position.

In the third case (**faces**) each ray returns the distance from the point of its intersection with a specific part of a complex object to the point of the sensor position. These are rays 3 and 4.

Available outputs for every ray of the ray sensor are shown in Figure 1.20. Counter clockwise rotation is used for numeration of rays, see in Figure 1.19.

- [**ObjectDetected**] is the object detection flag. The flag is **1** if any object detected and it is **0** if not. If a ray does not detect any object all ray sensor outputs will be set to zero too.
- [**Range**] is the distance from the sensor to the detected object. It is **0** if the ray does not detect any object.
- [**Azimuth**] is the azimuthal ray angle in degrees. Every single ray has the constant azimuthal angle in the sensor's frame of reference.
- [**Elevation**] is the elevation ray angle. In the current implementation of ray sensors all rays are located in the XY plane of the sensor's frame of reference and have zero elevation. In fact, it is reserved for the future use.
- [**DopplerVelocity**] is the Doppler velocity. It is a projection of the vector of relative velocity of the sensor and the detected object (point) to the line between them. Doppler velocity has the positive value if distance between objects decreases.
- [**WorldPositionX, Y, Z**] is the position of the intersection point between every single ray and the detected object in the global frame of reference.

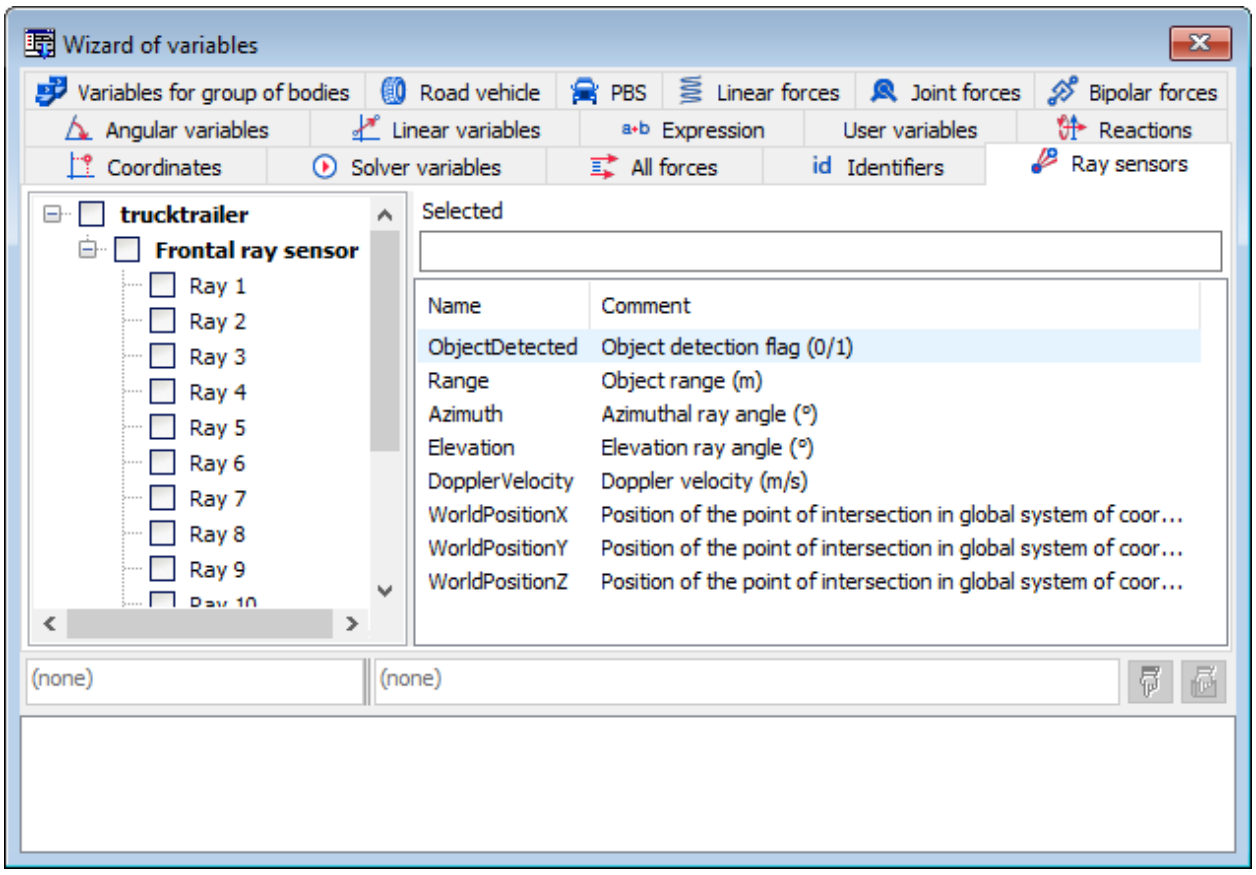

Figure 1.20. Variables for ray sensors

1.4. Range, azimuth and elevation

Range (*R*), azimuth (α) and elevation (β) are shown in Figure 1.21.

Azimuth is the angle between X axis to the projection of the line from the sensor to the object on the XY-plane in the sensor-fixed frame of reference. Positive azimuth corresponds to rotation of the X-axis clockwise if to watch from the positive end of the Z-axis.

Elevation is the angle between line to the object and its projection on XY-plane in the sensor-fixed frame of reference. Positive elevation corresponds to the shortest rotation of the X-axis to the Z-axis.

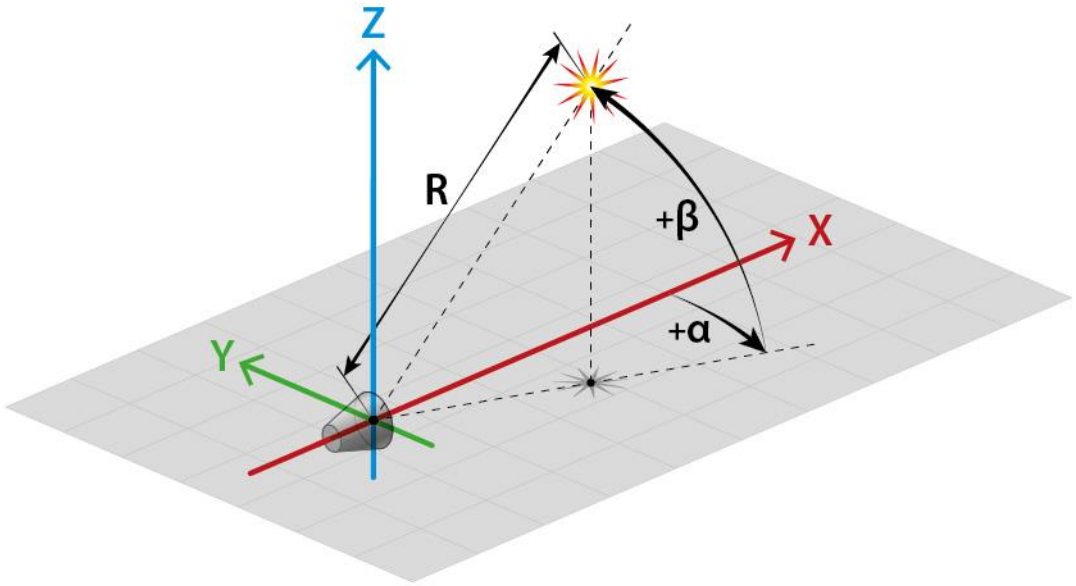

Figure 1.21. Range, azimuth and elevation

1.5. Beacon

A beacon sensor provides a quick, simple, universal and technology-independent solution for detecting an object.

Sensors of this type can be used for scenarios in which a simple sensor is required. Typical examples of its application: tuning controllers and testing decision-making algorithms. Experiments are considered “simple” when there is no obvious need to generate raw information from sensors; only top-level readings are generated, such as range and angle information. However, beacons provide accurate information. If more sophisticated data is needed or the sensor should be used in accordance with some scanning principle, then other types of sensors should be used.

For the beacon you can adjust the position, orientation, viewing angle and target range. The beacon settings are similar to those used for the ray sensor described above.

In the main menu select item **Tools | Sensors** and open the **Sensor configuration** window, see figure 1.14. For adding or deleting beacons use the window menu or right-click and use the drop-down menu.

For configuring the beacon left double-click on it. The **Beacon settings** window appears, see Figure 1.22.

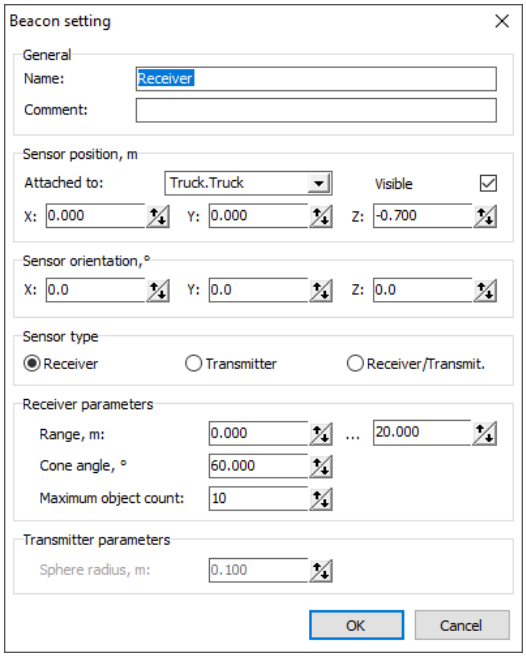

Figure 1.22. Beacon settings

The beacon identifier is specified in the **Name** field. In the **Comment** field, you can enter additional information that describes the sensor. Next, the object the sensor is attached to is selected, and its position is indicated. The **Sensor orientation** is determined by the rotation angles around the X, Y, Z coordinate axes. In this direction, the zone, where the sensor can detect the target is determined by the **Field of view** and the **Range** of distances to the target.

The beacon is an accurate sensor. Each beacon might be is a receiver, a transmitter or both receiver and transmitter at the same time. A scene containing a collection of visible objects for beacons is created. For making the object detectable, a transmitter must be installed on it. The sensor is attached to the object in a certain position and with a given orientation, which determines the direction of the beam. The beacon has a detection zone, determined by the **Range** and **Field of view**. The beam is cone-shaped.

A receiver can detect up to *N* transmitters in the detection cone, where *N* is the **Maximum object count** setting. If in the detection cone there are more transmitters than specified in the **Maximum object count** then closest transmitters will be detected.

The occlusion is not taken into account, and the sensor can detect "blocked" objects. The beacon ray passes through the obstacles blocking an object. An example of such a situation is shown in Figure 1.23, where object 5 is visible, despite object 3 blocks it. Objects 6 and 7 are not visible, because they are beyond the field of view of the beacon 1.

Sphere radius for transmitters is the radius of the sphere that is highlighted the exact position of the transmitter during the simulation. Sphere radius is used for the visualization only and does not effect on the transmitter detection.

Available outputs for every receiver are listed below.

- [**ObjectDetected**] is the object detection flag. The flag is **1** if any transmitter is detected and it is **0** if not. If no transmitters are detected all other sensor outputs will be set to zero too.

- [**Range**] is the distance from the sensor to the detected object (transmitter). It is **0** if the ray does not detect any object.
- [**Azimuth**] is the azimuthal angle to the detected object in degrees in the receiver frame of reference.
- [**Elevation**] is the elevation angle to the detected object in degrees in the receiver frame of reference.
- [**DopplerVelocity**] is the Doppler velocity. It is a projection of the vector of relative velocity of the sensor and the detected object (transmitter) to the line between them. Doppler velocity has the positive value if distance between objects decreases.
- [**WorldPositionX, Y, Z**] is the position of the detected object in the global frame of reference.

The results of these calculations can be displayed in the form of plots. When preparing the plots, the following conventions are used.

If several objects are in the field of view of the beacon, the parameters of the nearest object are displayed. If the beacon does not detect any objects, the value of the displayed parameter is zero.

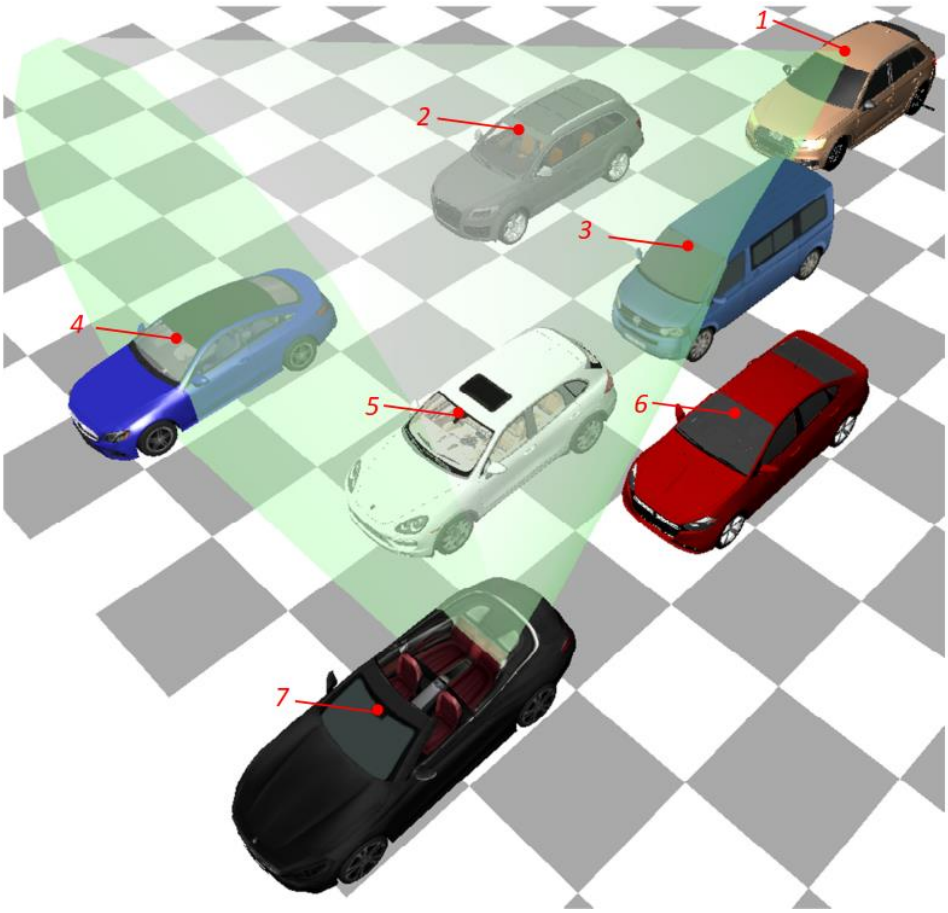

Figure 1.23. Objects detection using beacon 1:
objects 2, 3, 4 and 5 are visible; objects 6 and 7 are invisible

1.6. Ray sensors and beacons monitor

Objects detected by ray sensors and beacons at each moment of time are displayed in the monitor windows. We consider the monitoring of ray sensors and beacons using the example of a beacon.

To open the beacon monitor use the menu item **Tools** | **Beacon monitor** or **Ray sensor monitor**, see Figure 1.24. The beacon (ray sensor) monitor window opens, see Figure 1.25. The graphic field (1) and the projection of the cone of the beacon ray (2) are displayed on the left side of the monitor window. Blue markers (3) depict the objects in the beacon's field of view. Red marker (4) in the origin represents the selected sensor. The right part of the window contains a list of available beacons (5). Any of them can be set active by selecting it in the list.

Note If your model contains several ray sensors or beacons, and you want to monitor them at the same time, you have to open several monitor windows and select a sensor for monitoring in each window.

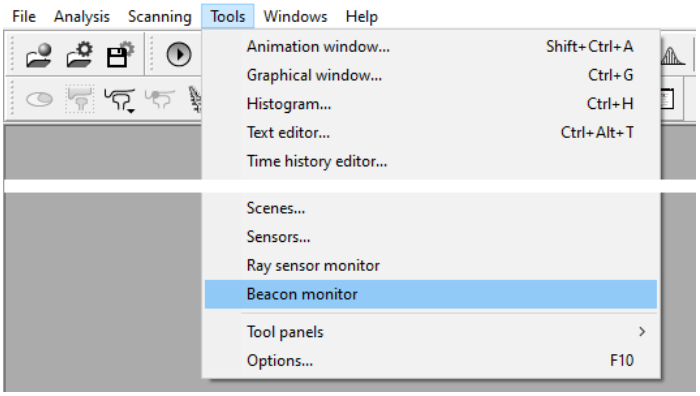

Figure 1.24. Displaying beacons monitor window

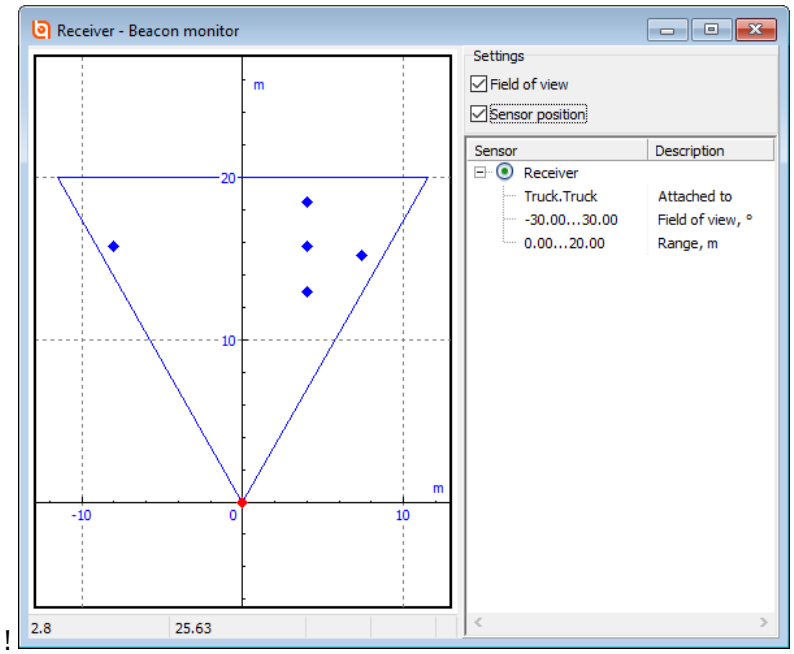

Figure 1.25. Beacons monitor window

1.7. Camera sensor

Camera sensor is a virtual video camera which can be installed on vehicle models created in UM. This type of sensor is the fourth sensor component of ADAS. Virtual camera generates a sequence of images that can be seen from the camera installation point, with a given time step. The sequence of images can be saved in a video file on disk or transferred to third-party applications using TCP/IP. The camera sensor can be installed on any model developed in UM and, therefore, it is not limited to use in the automotive applications. The camera sensor can be used in any problems requiring visual control of the operation of mechanical systems, such as problems of robotics systems, as well as for preparing videos to illustrate the operation of mechanisms.

1.7.1. Camera sensor settings

To create new camera sensor open **Tools | Sensors** in main menu and use **Wizard of sensors** item (Figure 1.26). Right click on the form or left click on the button with a symbol "+" and then select **Camera**.

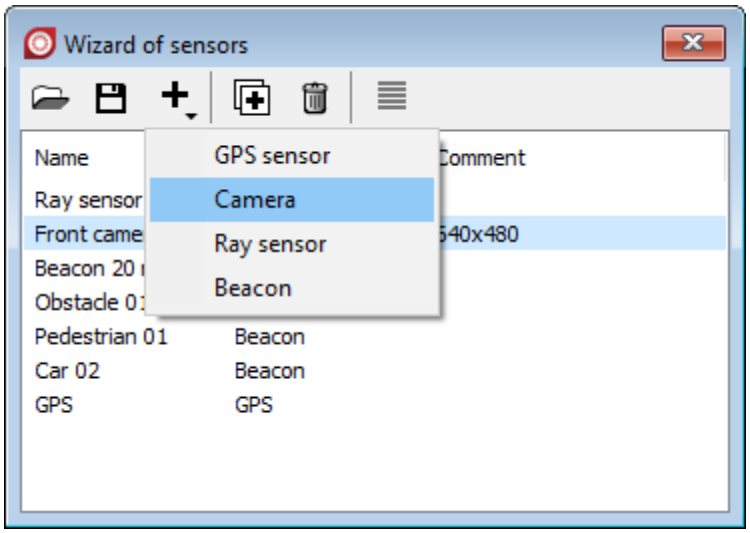

Figure 1.26. Adding new camera sensor

Camera sensor setting dialog has three tabs: **General**, **Extended** and **Video export** (Figure 1.27, Figure 1.28 and Figure 1.29, respectively). We now consider only camera specific parameters.

In the tab **General** the following camera settings can be configured.

- **Camera settings.**
  - **Image type:** select image type.
    Color: 8-bit color, RGB; **Monochrome:** grayscale, 8 bit color.
  - **Time step:** time interval between frames in seconds. Time step must be a multiple of the step of presenting the results, which is set in the object modeling inspector. Recommended value: 0.04 s; this value corresponds to the frame rate of 25 frames per second (FPS). The FPS is displayed below the time step edit box for reference purposes.

The tab **Extended** describes the image sensor parameters.

- **Image size:** pixel dimensions (width and height).
- **Sensor size:** selection from typical sensor sizes of industrial cameras or user size.
- **User sensor size:** width and height of sensor in millimeters.
- **Focal length:** focal length of camera in millimeters.
- **Field of view:** horizontal and vertical field of view in angular degrees.

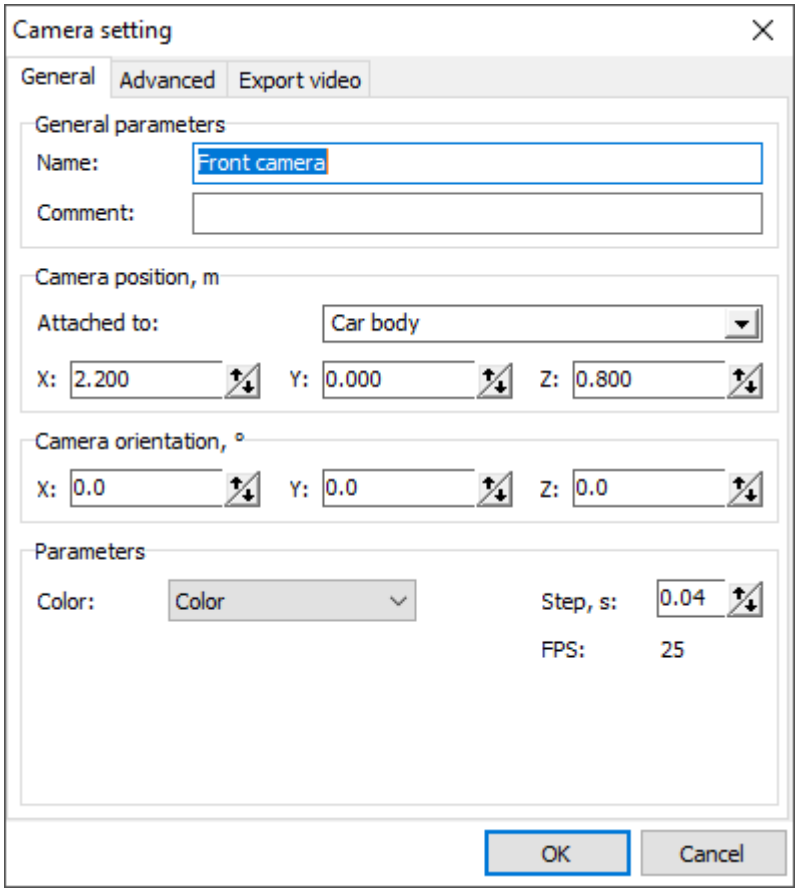

Figure 1.27. Camera sensor setting: **General** tab

The sensor dimensions and the focal length are related to the fields of view as follows:

$$Horizontal\ \ field\ of\ \ view = 2 \cdot arctg\left(\frac{Sensor\ \ width}{2 \cdot Focal\ length}\right) \cdot \frac{180}{\pi},$$

$$Vertical\ \ field\ of\ \ view = 2 \cdot arctg\left(\frac{Sensor\ height}{2 \cdot Focal\ length}\right) \cdot \frac{180}{\pi}.$$

Default image sizes are listed below:

$$Image\ width = 640,$$

$$Image\ height = Image\ width \cdot \left(\frac{Sensor\ height}{Sensor\ \ width}\right).$$

Thus, the desired field of view of the camera can be achieved in two ways. The first way is to specify the physical dimensions of the sensor and the focal length. In this case, the horizontal and vertical fields of view will be calculated automatically using the relationships given above. The second way is to set the fields of view directly; then the sensor dimensions will be calculated

automatically, whereas the focal length will not change. In both cases the width and height of the image in pixels will be proportional to the width and height of the sensor, respectively. The image size set first is leading; the second one will be calculated automatically based on the aspect ratio of the frame. Changing the focal length will also affect the fields of view; other settings will retain their values.

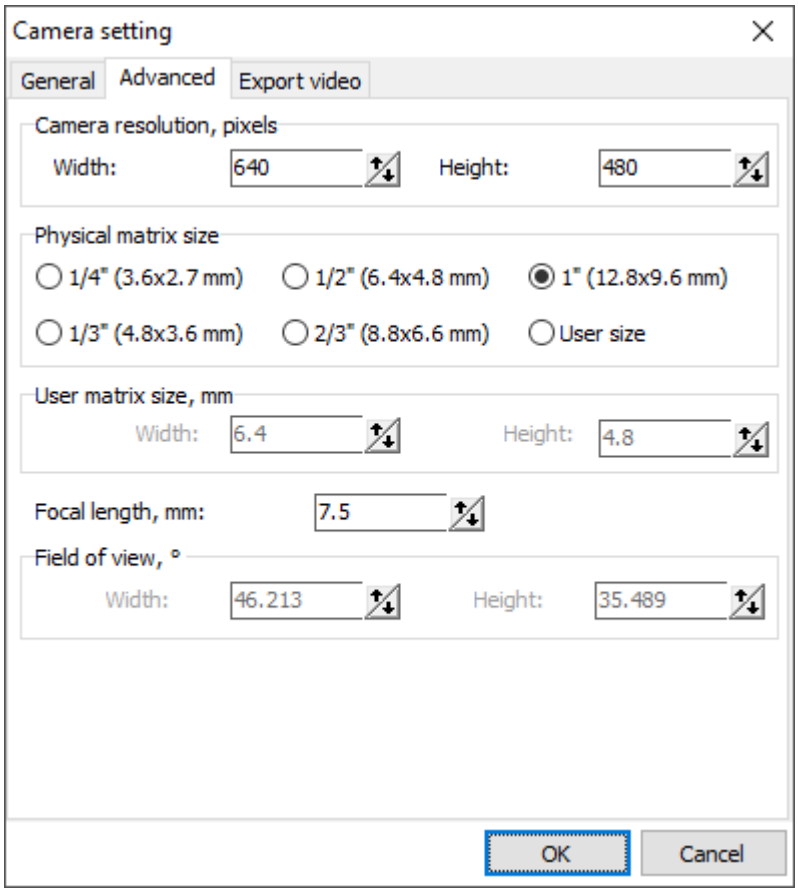

Figure 1.28. Camera sensor setting: *Extended*

Tab **Video export** describes the TCP/IP connection settings.

- **Connection settings.**
  - **Export video:** when checked, video export will begin when the simulation process begins.
  - **TCP/IP host:** network address of the receiving computer. The address 127.0.0.1 (the default value) allows you to establish a connection and transfer data to the server application (receiver) locally, that is, to the same computer.
  - **Port:** port number (default value: 20011). The port must be free and not used by other applications.
- **Connect:** connection to the server application will be attempted. The server application must be started in advance and the receive mode must be activated. The **Connect** button is intended for test purpose. If the **Export video** checkbox is checked and the connection settings are specified correctly, the connection will be established when the simulation process begins.
- **Disconnect:** an attempt will be made to disconnect from the server application.

- **Send frame:** one frame from the selected camera sensor will be sent to the server application. The server application must be started in advance.
- **OK:** saves the settings and closes the camera sensor setting dialog box.
- **Cancel:** closes the camera sensor setting dialog box without saving.

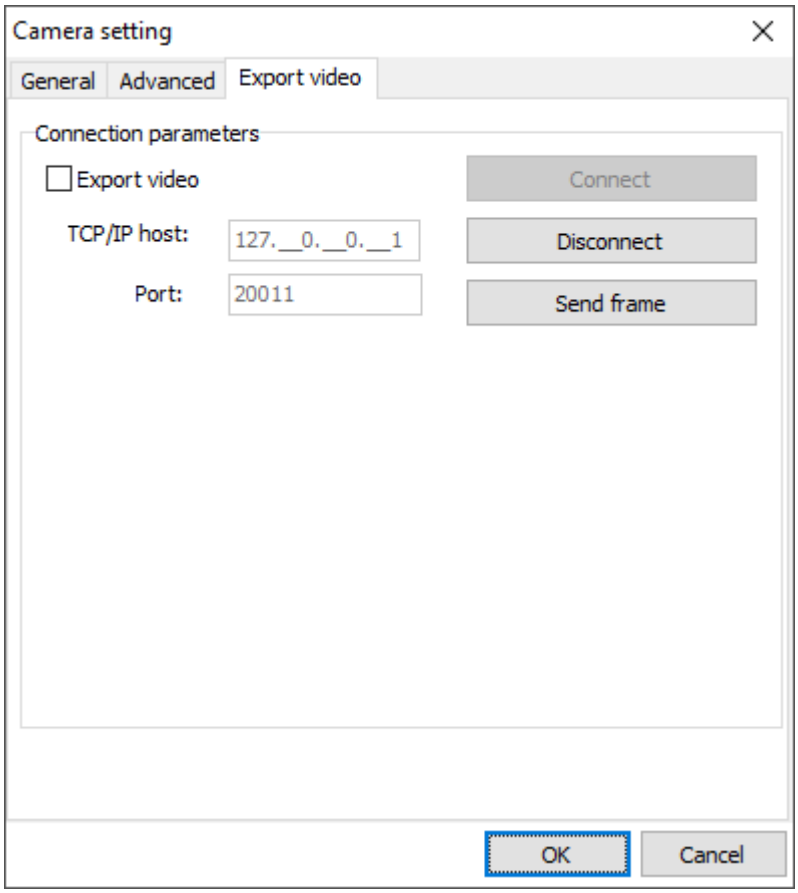

Figure 1.29. Camera sensor setting: *Video export*

If the camera sensor has been created and configured correctly, the **Export video** checkbox is checked and server application is in receive mode, the camera sensor will start sending video frames when the simulation process begins. Server application can be developed in MATLAB/Simulink, SimInTech and other simulation systems. It is assumed that the server applications analyze the received sequence of images in real-time or offline mode and recognize road signs, detect and track road lanes, vehicles, pedestrians and road obstructions and use these data for ADAS and self-driving cars.

1.7.2. Using camera monitor

Camera monitor helps controlling camera sensors. For enabling camera monitor right-click on the selected camera in the camera sensor settings and select the **Show sensor monitor**, see Figure 1.30.

In case if the model contains several camera sensors, you can switch between them by selecting the camera from the list (Figure 1.31).

The camera monitor window provides the following features, see Figure 1.31:

- copying image to the clipboard;
- saving image to a file;
- saving animation to a file.

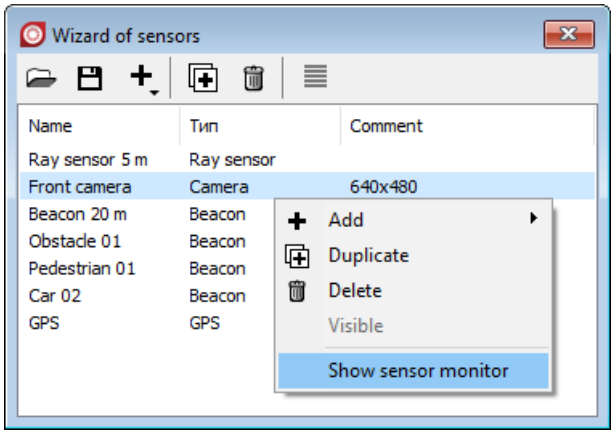

Figure 1.30. Wizard of sensors: show camera monitor window

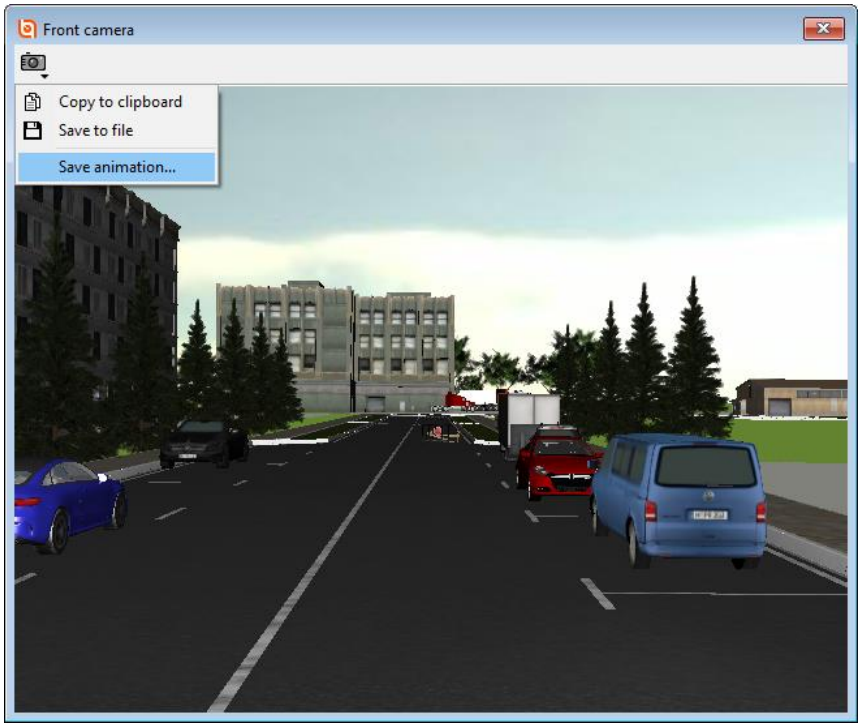

Figure 1.31. Camera monitor window

Selecting **Save animation** opens the recording settings dialog box, see Figure 1.32.

- **Save animation:** if checked, recording will begin when simulation process begins.
- **Copy step:** time interval between frames in seconds.
- **File name:** the video file name.
- **Time scale:** a factor that allows you to compress or expand the duration of an animation stored in the file relative to real time.
- **Codec:** select a video codec from the list. Currently available: no compression, Microsoft Video 1, Lagarith Lossless Codec, TechSmith Screen Capture Codec.

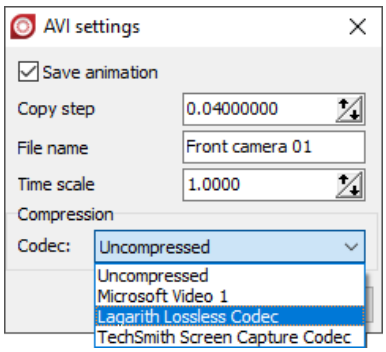

Figure 1.32. Configuring animation recording

1.7.3. Organizing video flow reception

Let us consider the reception of a video sequence using an example of a simple server application (receiver) implemented in MATLAB (script file UMVideoFlowTestReceiver.m). The script must be run before exporting video from UM.

After deleting all objects from the project, freeing system memory and clearing the command window, a TCP/IP object is created:

```
clear all; close all; clc;

t = tcpip('127.0.0.1', 20011, 'NetworkRole', 'server', 'Timeout', 600);
```

The receiving host and port number should be the same as specified in the camera sensor settings in UM. The timeout value sets the time in seconds that the script will wait for reception. If reception does not start or is interrupted, the connection will be disconnected after this time.

Buffer size is set: the number of bytes that can be simultaneously delivered to the receive queue. The size of the buffer is arbitrary; if the value is too large, MATLAB will warn you about it. The default value ensures the delivery of 84 bytes of control data and 1200*900*3*2 = 6480000 bytes for receiving two color images 1200×900.

```
t.InputBufferSize = 84 + 1200*900*3*2;
```

TCP/IP object opens:

```
fopen(t);
```

From now on, the script is ready to receive data from UM. When connected, the control sequence of 21 int32 numbers will be transferred from UM and used to initialize the parameters. After receiving, the data is deleted from the buffer:

```
indata = fread(t,21,'int32');
flushinput(t);
```

The initializing data form the following structure:

```
% Coordinates of camera sensor in the object's reference frame
f1 = 'CameraPosition';  v1 = indata(1:3);

% Dimensions of the car body along coordinate axes X, Y, Z, respectively
f2 = 'CarBodySize';     v2 = indata(4:6);
```

```
f3 = 'FrameRate';       v3 = indata(7); % Frames per second

f4 = 'Channels';        v4 = indata(8); % 0 - one camera (monoscopic vision);
                                        % 1 - two cameras (stereoscopic
                                        % vision), currently not available

f5 = 'Colors';          v5 = indata(9); % 1 - RGB image, 0 - grayscale image

% RGB channels intensity for conversion from RGB to grayscale:
f6 = 'IntensityFactors';  v6 = indata(10:12).*100;

f7 = 'StereoBase';        v7 = indata(13); % Stereobase, mm,
                                           % currently not used

f8 = 'FocalLength';       v8 = indata(14);  % Focal length, mm

f9 = 'SensorType';        v9 = indata(15); % identifier for selecting from
                                           % typical sensor sizes
                                           % of industrial cameras

% Width and height of sensor in millimeters. Real values are multiplied
% by 100 before transmission, transferred as integers and divided
% by 100 after reception
f10 = 'SensorSize';       v10 = indata(16:17).*100;

% Horizontal and vertical image sizes:
f11 = 'ImageResolution';  v11 = indata(18:19);

% Horizontal and vertical fields of view, Real values are multiplied
% by 100000 before transmission, transferred as integers and divided
% by 100000 after reception
f12 = 'FieldOfViewDeg';   v12 = indata(20:21).*100000;

% Filling structure fields
CameraInitData                                                              =
struct(f1,v1,f2,v2,f3,v3,f4,v4,f5,v5,f6,v6,f7,v7,f8,v8,f9,v9,f10,v10,f11,v11,
f12,v12);
clear indata; % Принятые данные удаляются из памяти
```

The frame sizes in pixels are set:

```
w = v11(1,1); h = v11(2,1);
```

To correctly complete the reception of data and close the video file, you can use, for example, the frame counter. Knowing the model time and the time step for capturing video, you can calculate the number of frames that will be received and interrupt the reception loop:

```
MaxFrames = 1000;
```

Memory allocation for a single RGB frame with resolution w×h:

```
RGBMatrix = zeros(h,w,3);
```

There is an option of recording received frames in a video file from the receiver program. To do this, create a video object:

```
writerObj = VideoWriter('d:\ReceivedVideo.avi');
writerObj.FrameRate = v3;  % frame rate
writerObj.Quality = 85;    % compression quality, from 0 to 100
open(writerObj);
```

Further work of the receiver program is reduced to the sequential reception of integer arrays that form a video sequence:

```
if v5 == 1      % color images are sent
   index = 1;
   while keep_receiving   % images reception loop
      indata = fread(t, h*w*3, 'uint8'); % reception of a single frame
      count = count + 1;  % received frames counter
      for i = 1:h         % loop for lines
         for j = 1:w      % loop for columns
           RGBMatrix(h-i+1,j,1) = indata(index+2)/255; % forming a 3-D array
           RGBMatrix(h-i+1,j,2) = indata(index+1)/255; % of a dimension
           RGBMatrix(h-i+1,j,3) = indata(index)/255;   % w*h*3
           index = index+3;
         end
      end
      index = 1;
      frame = im2frame(RGBMatrix);  % conversion of a received image
                                    % into a frame
      writeVideo(writerObj, frame); % frame is written into a video object
      imshow(RGBMatrix);            % received frame is displayed
      flushinput(t);                % data are removed from the buffer
      if count > MaxFrames   % checking for the received frames number
         keep_receiving = 0; % stop frames reception
      end
   end
   close(writerObj); % close video object
end
```

The reception of monochromatic images from UM can be organized as follows:

```
if v5 == 0     % monochromatic images are sent
    GrayscaleImage = zeros(h,w); % memory allocation for a single frame
    index = 1;
    while keep_receiving
       indata = fread(t, h*w, 'uint8');
       count = count + 1;
       for i = 1:h
          for j = 1:w
              GrayscaleImage(i,j) = indata(index);
              index = index + 1;
           end
       end
       flushinput(t);
       if count > MaxFrames
          keep_receiving = 0;
       end
    end
end
```

The server application considered above can be modified or written from scratch according to the user needs.

Further work with the received images can be performed, for example, using the MATLAB Automated Driving Toolbox, which provides the user with a rich toolkit for image analysis in relation to automated driver assistance systems (ADAS). A detailed description is available in the MATLAB documentation; see, for example, https://www.mathworks.com/help/driving/.