Vision Beyond Earth: Synthetic Satellite Data for Neural Perception in Orbit

Marcus G. Müller[1,2], Wout Boerdijk[1,3], Maximilian Ulmer[1,4], Wolfgang Stürzl[1], Abel Gawel[5], Roland Siegwart[2], Rudolph Triebel[1,4], Maximilian Durner[1,3]

[1]German Aerospace Center (DLR)
Institute of Robotics and Mechatronics
Münchner Str. 20, 82234 Wessling, Germany

[2]Swiss Federal Institute of Technology (ETH)
Leonhardstrasse 21
8092 Zurich, Switzerland

[3]Technical University of Munich (TUM)
Boltzmannstrasse 3
85748 Garching, Germany

[4]Karlsruhe Institute of Technology (KIT)
Kaiserstrasse 12
76131 Karlsruhe, Germany

[5]Robotics and AI Institute (RAI)
Elias-Canetti-Strasse 2
8050 Zurich, Switzerland

Contact: Marcus.Mueller@dlr.de

Abstract*—Satellites have become a critical infrastructure pillar for modern life by supporting key services such as communication, navigation or weather forecasting. Yet, the consistent increase of satellites orbiting Earth brings a number of challenges with it, most importantly the steady rise of the chance of collision between orbiting bodies. At the same time, defunct satellites often remain in orbit as space debris, further adding to the number of items circulating Earth. Therefore, not only active debris removal but also means of extending the life span of a satellite - such as (preventive) maintenance and on-orbit servicing - gain increasing importance. For a system performing these tasks in space, perceptive capabilities are of high importance, such as the initial detection of the target satellite object, its successive tracking to visually follow its trajectory, or even satellite pose estimation. Here, automation is highly beneficial, and computer vision algorithms can greatly contribute - especially neural networks have shown vast advances in recent years for object perception tasks. Yet, the extremely harsh conditions in space create additional challenges, and the performance of detectors or pose estimators in industrial or house-hold applications cannot directly be expected for extra-terrestrial scenarios. Most importantly, relevant training data is often not available, and there is a lack of simulators supporting synthetic satellite data generation. In this work, we present Space OAISYS, a major extension of Outdoor Artificial Intelligent SYstems Simulator (OAISYS), supporting vast training data generation for the aforementioned tasks. Space OAISYS simulates satellite missions and creates visual data which can be used in a variety of scenarios. With the simulator one can use an arbitrary satellite object and let it orbit around any kind of celestial body. To use the extension for machine learning task, OAISYS Material Drivers are introduced, which can randomize materials in a great variety. Furthermore, the simulator is extended by sensor moving patterns, which create a more realistic satellite rendering result. Strong emphasis is also placed on precise modeling of visual artifacts in space such as intense reflections or blooming. To ease application, a standardized storage format is integrated. Code is available at https://github.com/DLR-RM/oaisys.

TABLE OF CONTENTS

979-8-3315-7360-7/26/$31.00 ©2026 IEEE

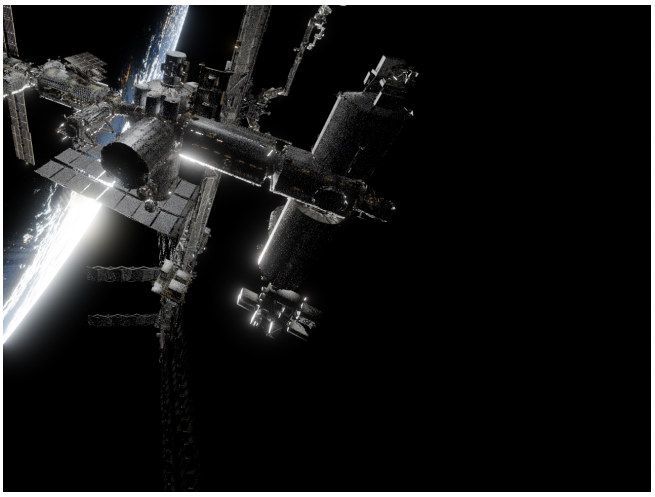


Figure 1: Image of the International Space Station (ISS) orbiting around Earth rendered by Space OAISYS.

1. INTRODUCTION

Since the launch of Sputnik 1 almost seven decades ago, satellites have gained significant importance by becoming vital for a wide range of applications impacting both our daily lives as well as society as a whole: They facilitate global communication and weather forecasting, and are essential for navigation and broadcasting. Today, the number of satellites orbiting Earth exceeds 12,000, and the exponential growth is believed to further continue even into 2030 [1]. This brings increased capacity for the aforementioned applications, but at the same time introduces significant challenges. One of the major problems is the increase of occupied space in Earth's orbits, even more so due to the fact that malfunctioning

satellites often remain in space degrading to space debris. The consequence is a rising risk of in-orbit collisions, not only between active spacecrafts but also with decommissioned satellites and other types of debris. This results in mission concepts for on-orbit servicing, e.g. DARPA's Phoenix program [2] or NASA's Restore-L mission [3]. Startup companies such as Effective Space[2] or Infinite Orbits[3] propose similar solutions for prolonging a satellite's life. Likewise, debris removal scenarios are considered, for instance with the RemoveDEBRIS mission by Surrey Space Center [4].

A key requirement for performing these tasks is knowledge about the target spacecraft's pose - i.e., its position and attitude relative to the servicer spacecraft. Yet, the availability of fiduciary markers cannot be guaranteed, and especially when dealing with debris, the target is incapable of providing information on its state. Hence, also for sake of generality, the localization must be performed on-board the servicer and ideally without a human in the loop.

For autonomous operation, detection and pose estimation / tracking of space crafts are key requirements. These are generally well-studied tasks, and in the house-hold and industrial domain there exists a plethora of datasets and standardized benchmarks (e.g., [5]). Importantly, tools like BlenderProc [6] or Kubric [7] facilitate the creation of photo-realistic datasets with automatic annotations for custom objects, enabling broad applicability of established algorithms on specific tasks in various environments. Yet, the space environment differs vastly from industrial or house-hold settings: illumination conditions, high contrast, specular reflections or specific artifacts like blooming pose additional challenges and need to be considered [8]. While there exist similar benchmarks, associated datasets and designated pose estimators for satellite applications - most notable Speed+ [9] and EagerNet [10] - there is a considerable lack of suitable simulators for realistic, customized satellite data generation with respective annotations, which on top support the harsh conditions mentioned before.

To close this gap we present Space OAISYS. It comprises a major extension to OAISYS [11], tailored specifically for the generation of synthetic training data in the satellite domain. Our enhancements focus on four main aspects: (1) environmental rendering, particularly of celestial bodies like the Earth and the Sun to simulate realistic illumination; (2) the realistic modeling of satellite appearances, including materials and surface characteristics such as multi-layer insulation foils; (3) OAISYS Material Drivers in order to create random material outputs; and (4) support for a wide range of annotations as well as the Benchmark for Object Pose Estimation (BOP) [12] format, facilitating compatibility with existing computer vision frameworks.

This contribution represents to our knowledge the first open-source satellite simulator with a dedicated focus on synthetic data generation for extra-terrestrial vision tasks. The simulator plugin and associated resources are publicly available at https://github.com/DLR-RM/oaisys.

2. Related Work

In this section we review available simulators relevant to our work, and list existing datasets.

[2] https://www.effective.space/
[3] https://www.infiniteorbits.io/

Space/Satellite Simulators

The *Satellite Orbit Simulator* (SOS)[4] is a free-to-use software solving the 4-body problem to determine satellite orbits with Earth, lunar and solar gravity included. Its main intended use is space mission design. Similarly, the NASA Operational Simulator for Small Satellites (NOS3)[5] is a suite of tools intended to be used to validate the functionality and performance of satellite systems before deployment.

Pangu [13] can scatter rocks and craters on randomly generated planetary surfaces, and furthermore supports inclusion of spacecrafts. SurRender [14] is a simulator for planetary terrain generation with a similar feature set. Both can generate various image modalities including height or slope maps and thermal or logarithmic images, but do not support any further annotations of planet surfaces, object parts or poses.

Available Datasets

URSO (Unreal Rendered Spacecraft On-Orbit) [15] is a collection of rendered images of spacecrafts orbiting Earth with accompanying pose annotations. The dataset is created with Unreal Engine 4 and publicly available[6], but the rendering code has not been open-sourced. Cassinis *et al.* [16] investigate reliable monocular pose estimation for near-range maneuvers involving an uncooperative spacecraft. For training a CNN-based detector, they render synthetic 2D images of the Envisat spacecraft with Cinema 4D software. This data is also employed in a subsequent work [17] for relative pose estimation of an inactive spacecraft performed by an active servicer spacecraft. The presented keypoint detector is then evaluated on images recorded from a designated test bed with a 1:25 scaled Envisat spacecraft mockup on a KUKA robotic arm. [18] generate synthetic images of the Northrop Grumman Enhanced Cygnus spacecraft using the physically-based Cycles engine of Blender [19]. Additional augmentations (glare, lens flare and blur) as well as real satellite photos of Earth as background images are used. The *Spacecraft PosE Estimation Dataset* (SPEED)[7] represents the first publicly available dataset for spacecraft pose estimation. It consists of 15,000 synthetic images and 300 real images recorded at the Testbed for Rendezvous and Optical Navigation (TRON), and has been used in the *Kelvins Satellite Pose Estimation Challenge* [20]. The subsequent work *SPEED+* [9] puts further emphasis on the existing domain gap between rendered synthetic images and real-world counterparts: It consists of an increased number of images collected with the TRON facility, with specific emphasis on high-fidelity spaceborne illumination conditions. Specifically, during recording a lightbox and a sunlamp illumination condition are used. The dataset facilitated the second international Satellite Pose Estimation Challenge [21].

Altogether, the lack of available datasets for satellite-related computer vision tasks is generally acknowledged and addressed over the past years: We observe an increase of available data recorded in elaborated test bed facilities, and corresponding benchmarks and challenges being set up. Still, existing datasets feature designated satellite mock-ups and camera parameters, and while taking diverse space-related illuminations into account, the resulting images and annotations are always constrained to specific setups and applications. While there exist a limited number of works on

[4] https://sos-orbital-mechanics.com/
[5] https://github.com/nasa/nos3
[6] https://zenodo.org/records/3279632#.XZIIuHVKg5k
[7] https://purl.stanford.edu/dz692fn7184

synthetic data generation, no code base is publicly available for facilitating data generation in a customized fashion with little manual involvement.

3. SPACE OAISYS

The OAISYS simulator is a photorealistic simulation pipeline designed for unstructured outdoor environments, with a particular focus on planetary robotics. Built on Blender, it enables parametric generation of diverse terrains using procedural mesh deformation, physically-based rendering (PBR) materials, and automatic scattering of objects (e.g., rocks) to create realistic landscapes. It produces high-fidelity multi-level semantic and instance annotations, allowing multiple semantic labels per object and per material. OAISYS supports configurable lighting (HDR or procedural sky), randomized camera poses, and outputs RGB, depth, semantic, and instance segmentation images.

In order to use OAISYS for space scenarios, the simulator needs to be extended by a variety of new components. For one, it has to be extended by celestial bodies, which differ in a few aspects from other mesh assets, for example by being larger than most objects. Furthermore, a specific module must be developed, which is able to make an arbitrary object orbit around another, to simulate a satellite mission. To use the simulator for a high variety of different-looking scenarios and also to use it for machine-learning tasks, a module that can randomize materials has to be developed. Finally, an extension to use sensors in a specific space mission has to be created. In the following, a selection of the main important extensions to the vanilla OAISYS simulator is described.

Celestial Bodies

We further developed OAISYS to use celestial bodies in the simulator. To spawn mesh assets, like planet models, in OAISYS, one can use the *MeshSingle* module. The module simply loads any provided Blender file, places it statically in the scene, and attaches semantic information to it, which can be used later for specific rendering. Although this is sufficient for many situations, it is not enough for celestial bodies. Such objects usually have a certain rotation axis and velocity, which should be simulated as well to yield a realistic output. Therefore, we extended the *MeshSingle* module to also feature the option to let a mesh rotate around a specific axis and velocity. Listing 3 shows an example of a planet configuration.

Listing 1: Example config file for a planet.

```
"name": "planet",
"type": "MeshSingle",
"meshParams": {
  "meshFilePath":"/user/earth.blend",
  "meshInstanceName": "earth",
  "rotation_deg": [23, 0, 0],
  "rotation_deg_step_v": 0.1,
  "rotation_mode": "local",
  "scale": [1.0, 1.0, 1.0],
  "position": [0, 0, 0]
}
```

Here, the parameter *rotation_deg* rotates the planet. The parameter *rotation_mode* controls around which axis the object is rotated, whereas the parameter *rotation_deg_step_v* defines the rotational velocity. Furthermore, some celestial bodies, like planets, are modeled with Volume Shaders, to, for instance, simulate the atmosphere of a given planet. These shaders are not supported in OAISYS and break the operation of the simulator. Therefore, we adapted OAISYS to support these kinds of materials. Figure 2 shows an example of the planet Mars rendered by the simulator.

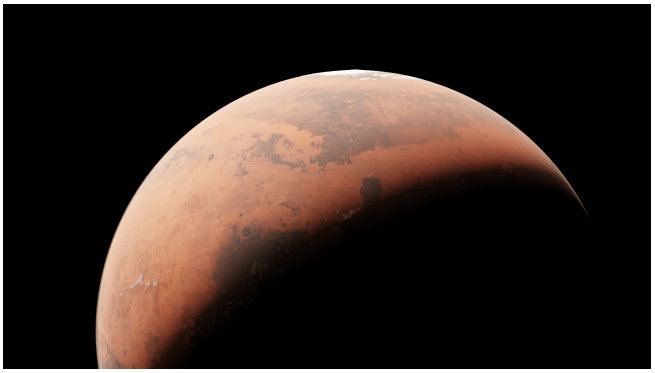


Figure 2: Example of a planet model Mars used in OAISYS.

Figure 3: Example of a possible outer solar system planet model used in OAISYS.

Note that in principle these objects can be any kind of objects, also any kind of planet modeled with any 3D software or downloaded from the internet. The planet in Figure 3, for instance, is an arbitrary model of an alien planet, which probably does not exist in reality. Figure 4 depicts exemplary real planets used in this study.

Satellite Module

Orbit Creation—In the vanilla version of OAISYS, users can define OAISYS modules, which load objects into the simulation. The most basic module, *Mesh_Single_Module*, simply loads any kind of object. These objects are independent of each other, meaning they do not have any information about each other. This, however, presents a problem for a module which spawns a satellite object, since a satellite object orbits around another object. Therefore, the module which spawns the satellite objects needs to have the object data of the module that will be orbited. Since OAISYS does not provide this option, the simulator was extended by a function that lets OAISYS modules share object data with each other.

Each module that spawns meshes now has an extra function called *update_after_initial_mesh_creation(mesh_dict)*. This function is called by OAISYS and provides each module all

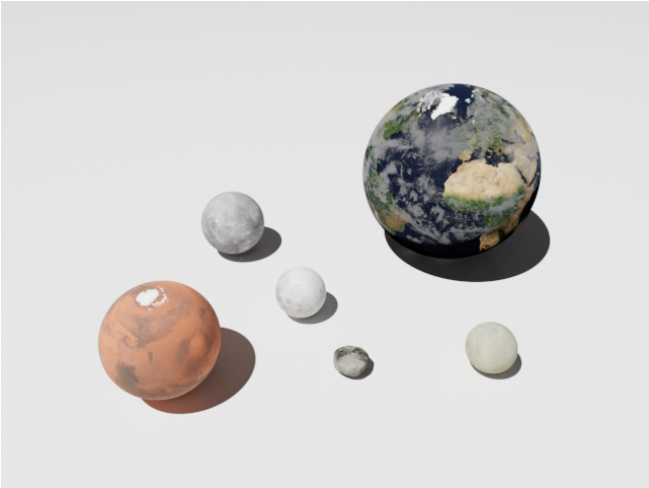


Figure 4: A selection of planets used in this study. From left to right: *Mars*, *Mercury*, *Earth Moon*, *Phobos*, *Earth* and *Europa*. Note that Phobos was scaled by a factor of 100 for better visibility.

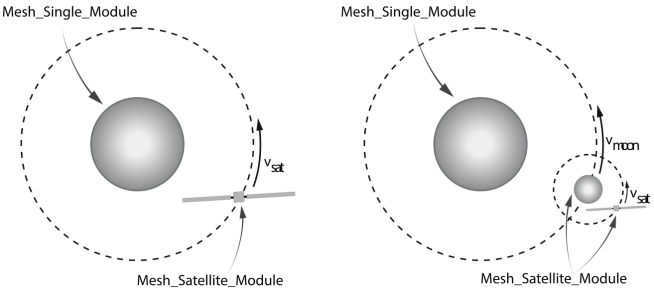


Figure 5: Left: Satellite orbiting around a celestial body. The celestial body is a *Mesh_Single_Module* object, whereas a *Mesh_Satellite_Module* spawns the spacecraft. Right: Example of a hierarchical usage of *Mesh_Satellite_Module*. The spacecraft, as well as the tiny celestial body are defined as *Mesh_Satellite_Module*. The innter celestial body is a standard *Mesh_Single_Module*.

meshes that were created by the simulator via a dictionary. Users can modify this function for their specific tasks. For the newly developed *Mesh_Satellite_Module*, the function is used to retrieve the information of the celestial body and place the satellite in orbit around it accordingly. With this extension, OAISYS modules can build up dependencies with each other and build more complex simulations without bypassing and breaking the overall OAISYS framework.

For simulating a satellite orbiting around a planet, the user can load the planet via *Mesh_Single_Module* and the satellite via *Mesh_Satellite_Module*. However, the simulator is also able to create more complex scenarios - for instance, the stacking of *Mesh_Satellite_Module*. This is particular interesting if one wants to simulate a spacecraft orbit around a moon, which itself orbits around a planet. Figure 5 depicts such orbiting scenarios.

Figure 6 depicts a selection of satellite objects used in this paper, which are all freely available on the Internet. We used

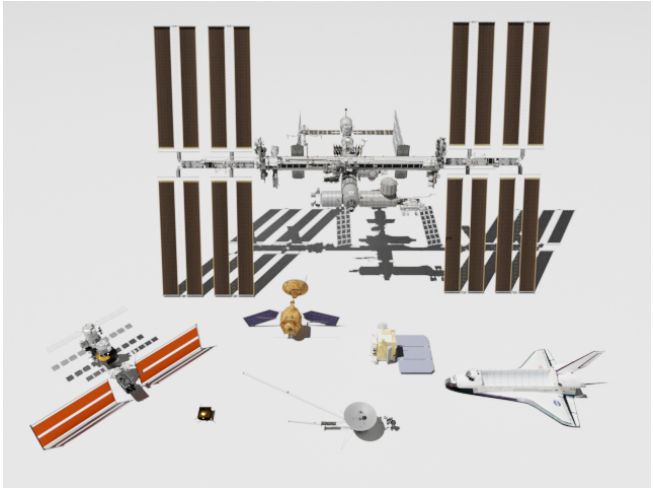


Figure 6: A selection of satellite objects used in this study. From left to right: *Lunar Gateway*, *ISS*, *Tango* (4x scaled), *Mars Reconnaissance Orbiter* (2x scaled), *Voyager* (2x scaled), *Lunar Reconnaissance Orbiter* (4x scaled) and the *Space Shuttle*. Model scalings are for better visibility.

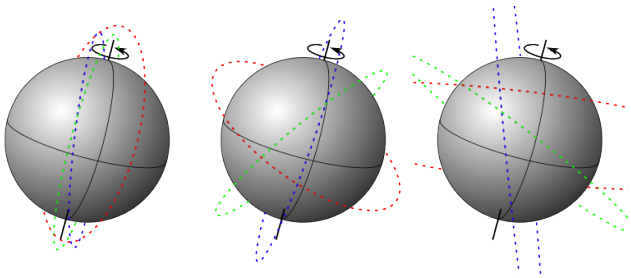


Figure 7: Example orbit trajectories with randomly sampled parameters. From left to right: *Polar*, *LEO* and *MEO* orbits.

models from ESA[8] and NASA[9]. Furthermore, we use freely available textures[10] from celestial bodies to create 3D models.

Orbit Sampling—The *orbit sampling function* takes a predefined orbit type as input, and the xyz-dimensions of the celestial body. It then calculates the necessary scale and rotation parameters for the respective orbit, thereby taking constraints such as altitudes or rotation alignment (in the case of a *Polar* orbit) into account. We provide different orbit types as initial implementations such as Low Earth Orbit (LEO), Medium Earth Orbit (MEO), Geostationary Earth Orbit (GEO) and Polar Orbit (PO). Importantly, a user can easily alter the default parameters (e.g. adjust *LEO* altitudes for a specific purpose), or even provide a completely custom sampling function which is exemplary demonstrated in Listing 2. Example orbits are visualized in Figure 7.

[8] https://gssc.esa.int/education/galileo3d/
[9] https://science.nasa.gov/3d-resources/
[10] https://www.solarsystemscope.com/textures/

Listing 2: Example orbit sampling function for a trajectory around the equator in a specific altitude within LEO.

```
def sample_equator_orbit_parameters(
    ↪ body_radius_xyz=[6378., 6378., 6378.])
    ↪ :
   altitude = np.random.uniform(800., 1200.)
   scale_xyz = (altitude + np.array(
       ↪ body_radius_xyz)) / np.array(
       ↪ body_radius_xyz)
   rotation_rpy = np.array([np.random.uniform
       ↪ (np.deg2rad(-5), np.deg2rad(5)),
       ↪ 0., np.random.uniform(-np.pi, np.pi
       ↪ )])
   return {
      "scale_xyz": scale_xyz,
      "rotation_rpy": rotation_rpy
   }
```

Listing 3 provides an example of the usage of the *Mesh_Satellite_Module*. Here, the dictionary *orbit_params* provides the orbit information. The parameter *orbit_type* lets the user choose the general type of orbit. *parent* defines around which OAISYS object the orbit is placed and linked. The parameter *v_step* defines the velocity of the satellite along the orbital track. *orbit_start* defines the starting positing of the satellite on the orbit.

Listing 3: Example of *Mesh_Satellite_Module* config file.

```
"name": "iss",
"type": "MeshSatelliteObject",
"meshParams": {
  "meshFilePath":"user/iss.blend",
  "meshInstanceName": "iss",
  "rotation_deg": [30., 0., 0.],
  "scale": [0.0000169332079,
            0.0000169332079,
            0.0000169332079],
  "rotation_deg_step_v": 0.1,
  "orbit_params": {
    "type": "orbit_database",
    "orbit_type": "LEO",
    "parent": "planet",
    "v_step": 0.01,
    "orbit_start": 0.3
  }
}
```

Alternatively, the user can also provide a specific desired orbit by providing direct values. Listing 4 shows an example of such a config file. Here, the shape of the circular orbit can be determine by the *scale* parameter. *rotation_deg* defines the rotation of the orbit.

OAISYS Material Driver

To develop and evaluate robust methods, introducing random variables in the simulation are crucial. This is particular true for data generated in order to train neural networks. Usually, textures and materials of the provided models are static. In Blender (and hence OAISYS), materials are defined as graphs, called node trees. Adjusting particular nodes in the graph can result in a variety of the material. For instance, Figure 8 shows the same material with different values of the material scale, which leads to different sizes of the textures.

Listing 4: Example of *Mesh_Satellite_Module* config file with a direct user defined orbit.

```
    "name": "iss",
    "type": "MeshSatelliteObject",
    "meshParams": {
      "meshFilePath":"user/iss.blend",
      "meshInstanceName": "iss",
      "rotation_deg": [30., 0., 0.],
      "scale": [0.0000169332079,
      0.0000169332079,
      0.0000169332079],
      "rotation_deg_step_v": 0.1,
      "orbit_params": {
        "type": "circular",
        "parent": "planet",
        "scale": [1.06396989,
        1.06396989,
        1.06396989],
        "rotation_deg": [30., 30., 0.],
        "v_step": 0.01,
        "orbit_start": 0.3
      }
    }
```

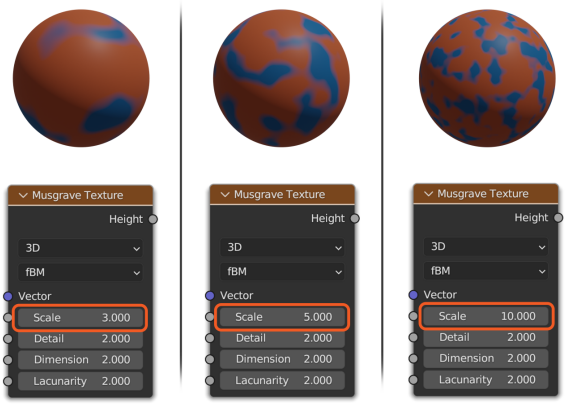


Figure 8: The figure shows a sphere with a Musgrave noise texture and the corresponding noise node. The *scale* value changes the size of the noise texture. The illustrations shows three different values for scale: 3, 5, 10 (form left to right).

In order to adjust these parameters in OAISYS, we introduce *OAISYS_Material_Drivers*. With these drivers, it is possible to extend the materials of a model by nodes, which can later be used by OAISYS to change their values at runtime. Users can attach a driver to any scalar value of the node tree by creating a scalar node that begins with the keyword *OAISYS_*, for example *OAISYS_texture_scale*. In the config file, users can then provide the random value information for the specific driver. The simulator crawls through the entire node tree and searches for nodes containing the keyword, then adjusts the values according to the config file. Values can be adjusted for every scene once, or also for every rendering step. Figure 9 shows the node tree from the previous example with its attached OAISYS Material driver.

Figure 10 illustrates a model of the Mars Reconnaissance Orbiter (MRO) for which the MLI material is changed by a material driver. The corresponding Blender node tree with drivers is illustrated in Figure 11 and the section of the config file in Listing 5.

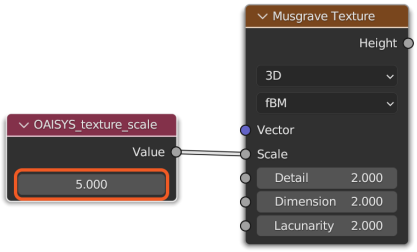


Figure 9: Adjusting the texture scale for a Musgrave noise texture, which can be done by the simulator.

Each driver consists of four parameters: *clip_min*, *clip_max*, *ideal*, *jitter_variance*. The *ideal* value defines the default value of the node. The parameter *jitter_variance* states the variance of the random value which is added to the *ideal* value. Since some nodes desire a minimum and maximum value, the driver also contains clipping values via *clip_min* and *clip_max*. The overall variance of the applied random jitter values can be controlled via the parameter *jitter_strength*. Each *jitter_variance* value is multiplied by *jitter_strength*. If *jitter_strength* is set to zero, the driver is set to the *ideal* value.

Listing 5: Example OAISYS Material Driver for adapting location, scale, and rotation information of a texture.

```
"material_drivers": {
  "drivers": {
    "scale": {
      "clip_min": 0.1,
      "clip_max": 10.0,
      "ideal": 1.0,
      "jitter_variance": 10.0
    },
    "location": {
      "clip_min": 0.0,
      "clip_max": 1000.0,
      "ideal": 0,
      "jitter_variance": 1000.0
    },
    "rotation": {
      "clip_min": 0.0,
      "clip_max": 1000.0,
      "ideal": 0,
      "jitter_variance": 1000.0
    }
  },
  "jitter_strength": 1.0,
  "step_interval": 1
}
```

With drivers it is not only possible to drive simple scaling operations of textures, but also create more advanced behaviors as illustrated in Figure 12, where the clouds are deformed through a driver.

Sensor Module for Space Missions

We extended OAISYS to track a given object with the sensor setup. For this, the sensor module of the simulation needs position data of the object to be tracked. The original version of the simulator does not provide any possibility to obtain that information. Therefore, we introduced a new function to the general sensor modules, called *set_oaisys_objs*. This function provides the sensor module with the all meshes spawn by OAISYS and can be used to update the position of the sensor accordingly to the tracked object.

Additionally, we introduce a variety of sensor pose sampling methods, which can be chosen by the user. The sensor module can either use random sampling around the satellite object or trajectory sampling. In random sampling, positions are sampled on a sphere with origin at the satellites position. Since the position of the satellite is moving with every simulation step, the sampled position on the sphere has to be updated as well. The radius of the sphere is given by a parameter, which also can be randomized. Therefore, this option is well suited to create a high variety of poses for machine learning tasks. An example config for random sampling is shown in Listing 6.

Listing 6: Example config to use random sampling for sensor poses.

```
"GENERAL": {
  "sensorMovementType":"orbit_object",
  "orbit_mesh": "iss",
},
"SENSORS": [{
  "type": "SensorCameraRGBD",
  "sensorParams": {
    "outputBaseName": "sensor_1",
    "imageResolution": [640, 480],
    "KMatrix": [541.14, 0, 320,
                0, 541.14, 240,
                0, 0, 1],
    "transformation": [0,0,0,1,0,0,0],
    "triggerInterval": 1,
    "renderPasses": {
      "RGBDPass": {
        "activationSlot":[1],
        "DepthEnabled": true
      },
      "SemanticPass": {
        "activationSlot":[1,1]
      },
      "InstancePass": {
        "activationSlot":[1]
      }
    }
  }
}]
```

For creating more realistic movements of an approaching spacecraft, we also developed a trajectory sampling module. The space version of OAISYS supports various kinds of pre-defined camera trajectories, which are visualized in Figure 13:

Linear Trajectory—A linear trajectory indicates straight motion from an (approaching) spacecraft towards the object of interest. Given pre-calculated satellite positions, a user can optionally specify relative start and end poses of the trajectory and a relative start and end velocity to further emulate a realistic approach (i.e., slowing down closer to the satellite). Then, n camera positions are sampled across the given trajectory.

Spiral Trajectory—Here, the camera circulates around the object of interest in a spiral fashion on a given plane. A user

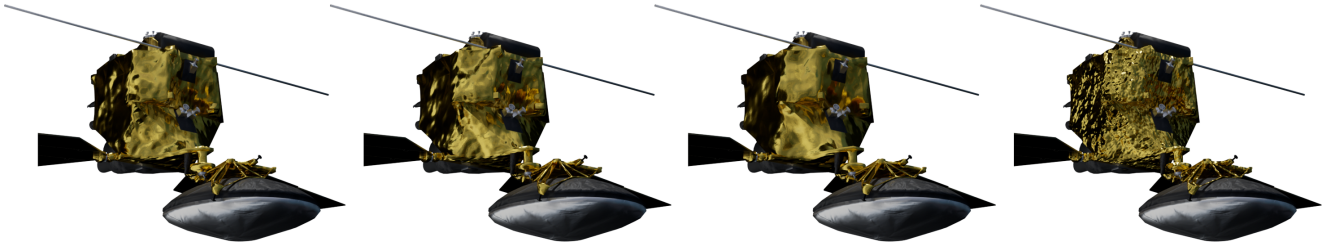


Figure 10: Different noise levels of a MLI material of the MRO.

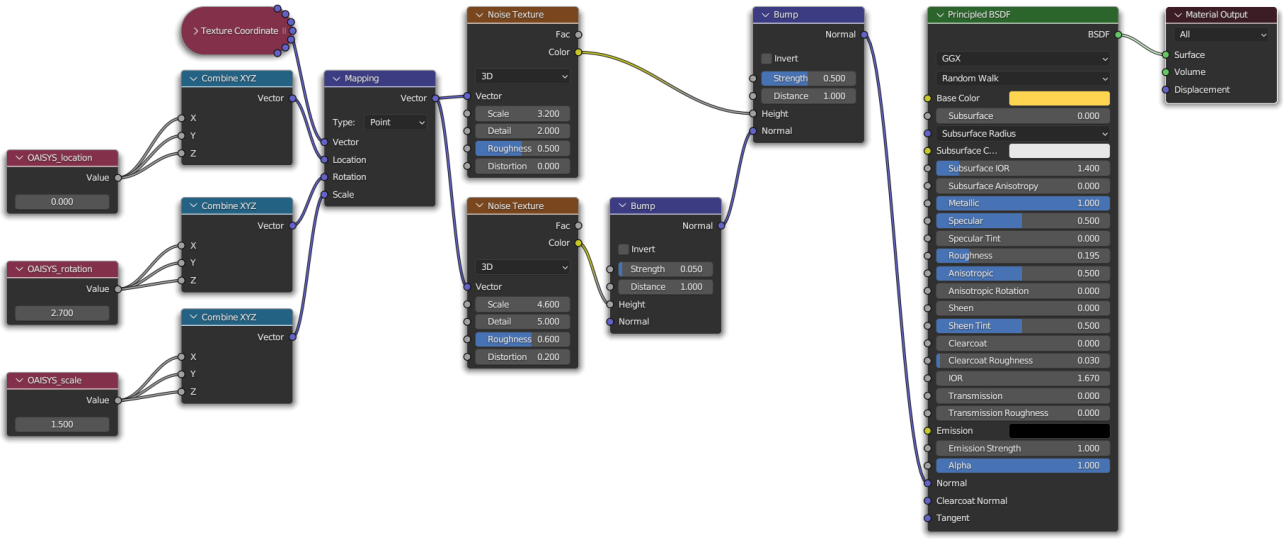


Figure 11: Blender node tree with OAISYS drivers for procedural material manipulation for a MLI texture.

Figure 12: Different cloud noise levels with OAISYS Material Drivers.

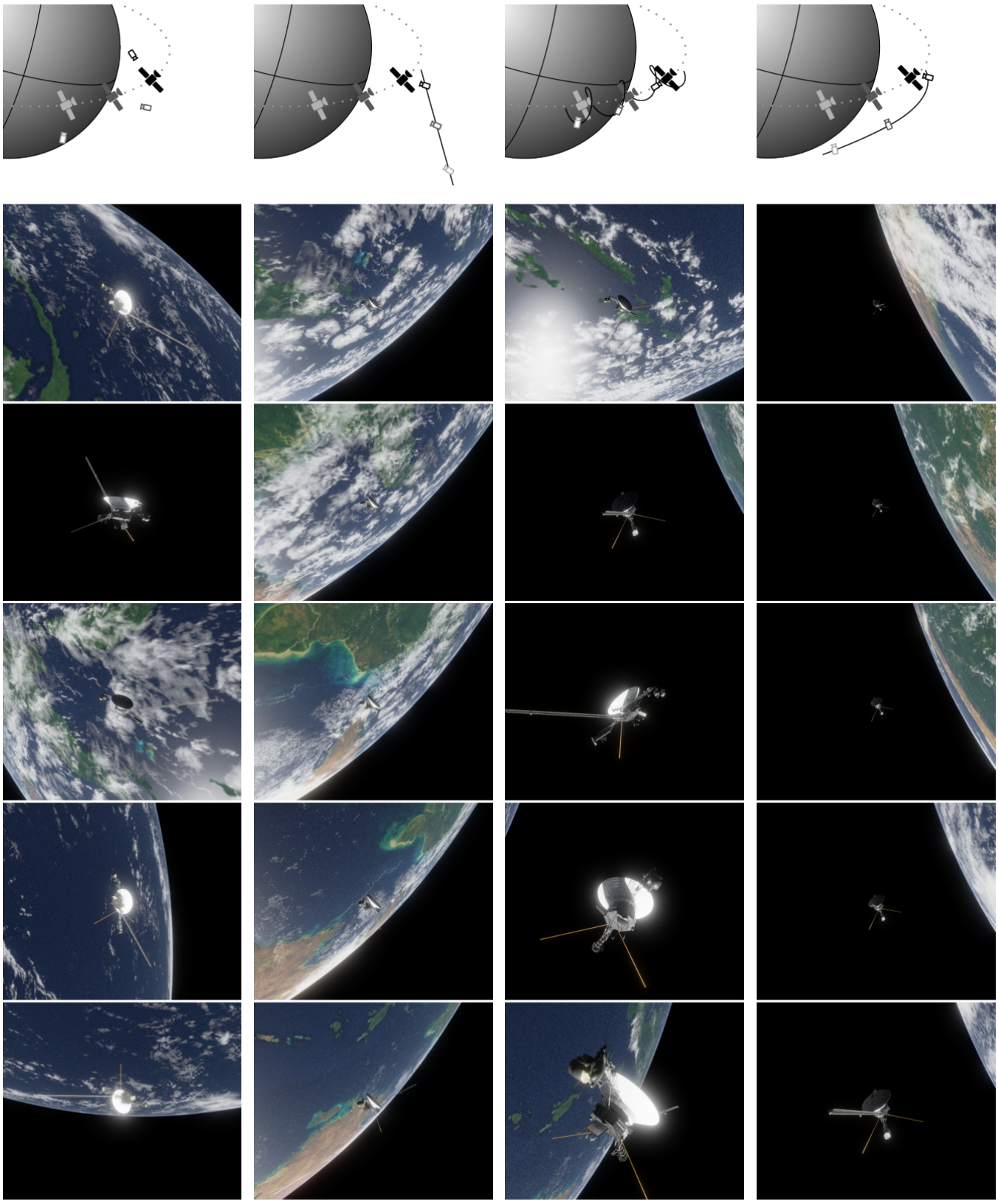
can specify a starting position and radius, and an optional end radius to emulate a spiral approach trajectory. Furthermore, the total number of rotations around the object can be specified. The exemplary config template is shown in Listing 7.

Flyby Trajectory—Given a rotation number smaller than 1 (e.g. 0.25) and a declining radius, the spiral trajectory type can directly be utilized to emulate a polynomial flyby path.

Custom Trajectories—Aside the wide range of parameters to be specified for each of the above trajectory types, a user can create a custom sampling function and specify this as camera pose generator, similar to the orbit sampling function (see Section 3).

Listing 7: Example config to use a spiral approach trajectory for sensor poses.

```
"GENERAL": {
  "sensorMovementType":"
     approach_trajectory",
  "orbit_mesh": "sat",
  "orbit_radius": 0.01,
  "orbit_radius_noise": 0,
  "approach_moving_type": "spiral",
  "approach_moving_params": {
    "r_start": 0.01,
    "r_end": 0.0001,
    "plane":  "xy",
    "rotations": 0.25},
}
```



(a) Random camera poses (b) Linear trajectory (c) Spiral trajectory (d) Flyby trajectory

Figure 13: Different types of camera placements with a sketch illustration at the top. In the pictogram, the grayscale colors of the satellite and camera pictograms simulates the different timesteps. The subsequent images showcase examples of the respective trajectory.

4. SELECTED USE-CASE EXAMPLES

In the following, we showcase exemplary scenarios to highlight the broad applicability of Space OAISYS.

Variety of Scenarios

Space OAISYS is capable of rendering any orbital object and planetary body in highly realistic fashion. Figure 14 and Figure 15 depict qualitative examples for a variety of space crafts (*ISS*, *Lunar Gateway*, *LRO*, *MRO*, *Space Shuttle*, *Tango* and *Voyager*) and six different celestial bodies (*Earth*, *Mars*, *Mercury*, *Earth Moon*, *Europa* and *Phobos*).

6D Pose Annotations

In our previous work [22] we integrated the option to retrieve additional metadata of the rendered scene and assets; specifically, the 6D poses of camera and asteroid. This can readily be applied to satellite meshes, and Figure 16 illustrates this by overlaying 6D satellite poses of Voyager during an approach trajectory of a space craft over Earth.

Semantic Annotations

In Computer Vision, image segmentation is the process of assigning class labels on a pixel-level. The material-based rendering process of OAISYS enables segmentation annotations in various formats, and gives a user fine-grained control of different labels. These promote a multitude of Computer Vision tasks, and exemplary annotations are explained in more detail in the following. A graphical overview is shown in Figure 17.

Semantic (Part-) Segmentation—Semantic segmentation divides an image into semantically different regions. Importantly, objects of the same semantic category are assigned the same class label. Part-wise segmentation further splits semantics up into individual object-related parts. For instance, in the case of a satellite, one could further sub-categorize the object into a body part, a dish, an antenna, and solar panels.

Instance Segmentation—In contrast to semantic segmentation, instance segmentation separates on an instance level - i.e., multiple objects belonging to the same semantic class each receive a unique, instance-specific id. This is particularly helpful when interacting with the environment on a sub-category hierarchy. As an example, consider the task of solar panel identification for e.g. preventive maintenance or cleaning. Here, one might not be interested in the overall solar array, but rather detect specific panels in order to carry out the aforementioned tasks.

Miscellaneous Render Demonstration

Last but not least, we showcase an example of the diverse rendering outputs in Figure 18.

Simulator Hardware Performance

Rendering performance depends heavily not only on the used computational resources, but also on the scene which is rendered. For instance, the rendering time will significantly increase with high resolution textures of planets or models with a high vertices count. It is also effected by which models are in view of the rendered image. Therefore, it is difficult to give a general performance statement of the simulator since it is scene depended. However, to give a reference we include the rendering information of the image in Figure Figure 15, which shows Voyager orbiting Phobos. This image was rendered on an Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz, 32GB CPU RAM, NVIDIA GeForce RTX 2080 with 8GB VRAM, and took 2.5 seconds and used 822 MB of VRAM.

Although all samples are rendered with Blender's built-in rendering engine Cycles, OAISYS can also be extended to use Eevee (another Blender rendering engine for real-time rendering) or in principle other fast rendering engines, if render speed is more important than quality. In general, to decrease rendering time, it is useful to reduce the model complexity of all used meshes and their texture sizes.

5. SUMMARY

In this work we present Space OAISYS, a major extension to the OAISYS Simulator specifically designed for orbital perception of space objects circulating around planetary bodies. The main focus is set on both easy usage and full customization to support a broad applicability: For instance, realistic trajectories are readily integrated, but a user can seemingly extend these with custom-written approaches. Dedicated material drivers allow randomization of materials of celestial bodies in a great variety for highly realistic renderings. We have showcased the multitude of annotations including 6D poses and various semantics. Future work might include the incorporation of faster rendering engines like Eevee, and the application to a diverse range of downstream tasks like satellite detection and pose estimation. Altogether, we believe that Space OAISYS is well suited for paving the way for future work on neural perception in orbit.

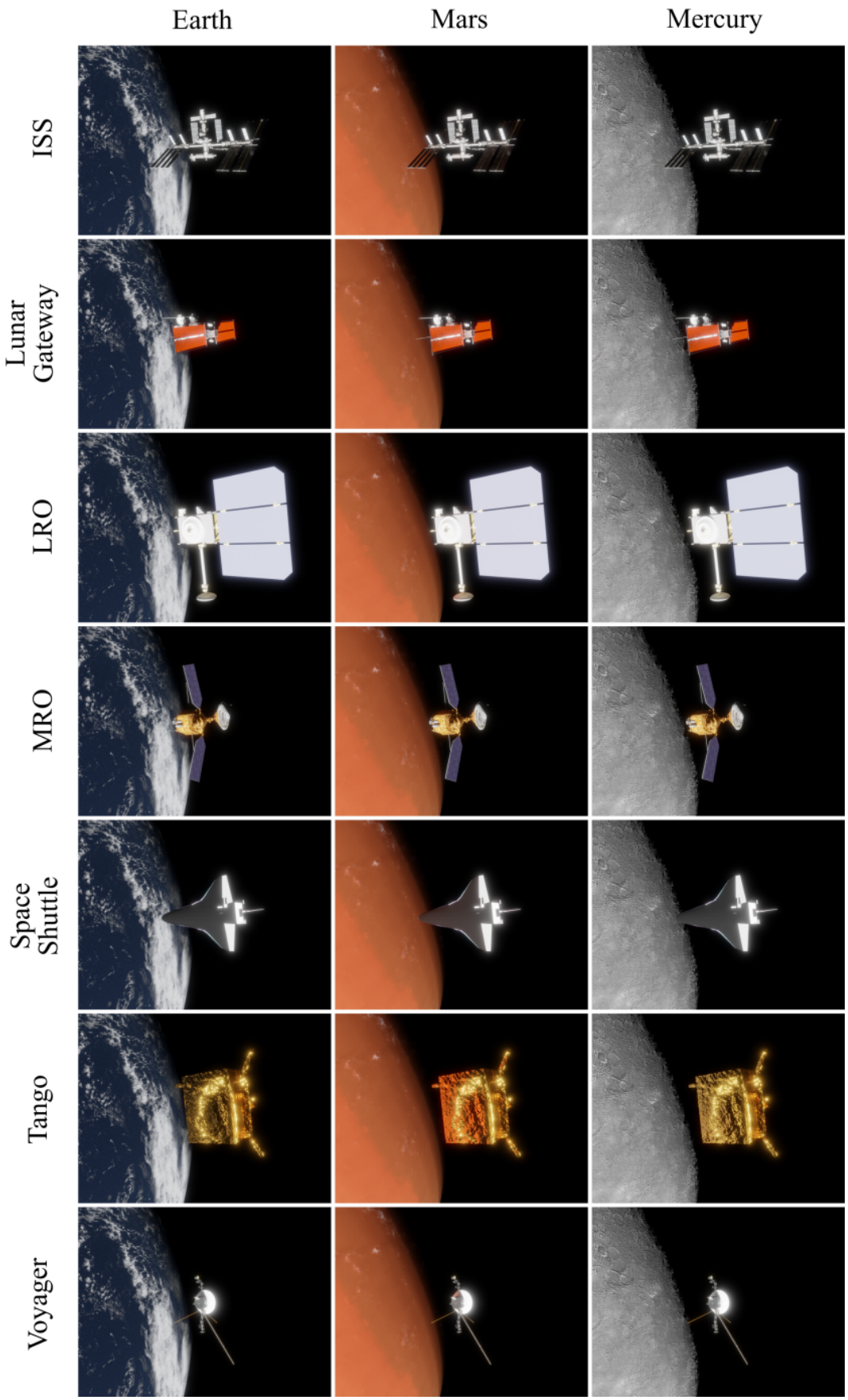


Figure 14: Seven different space crafts orbiting Earth, Mars and Mercury.

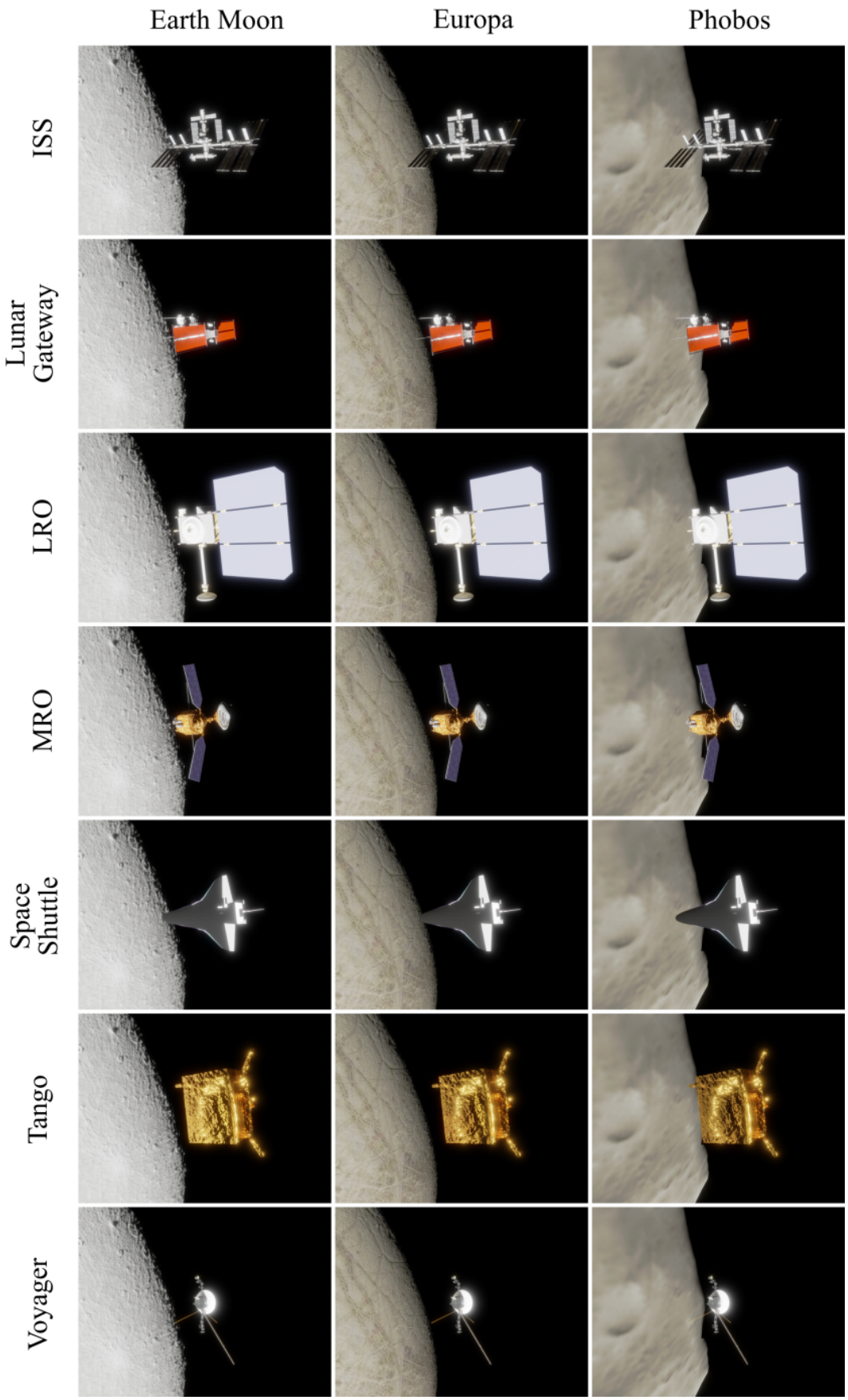


Figure 15: Seven different space crafts orbiting Earth Moon, Europa and Phobos. Note that the visual artifacts on Phobos are due to the limited resolution of the texture.

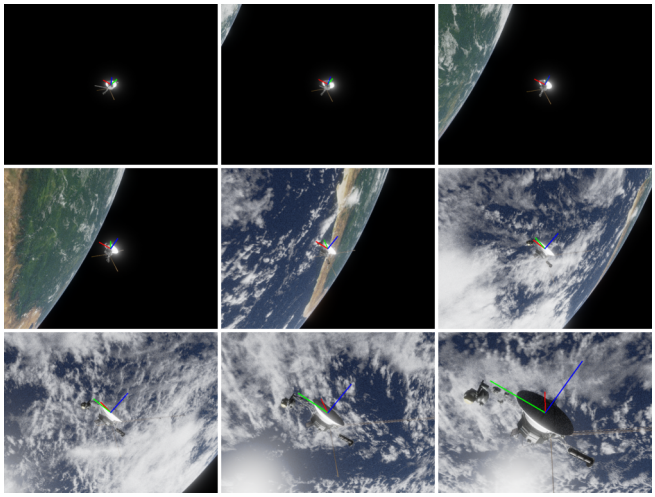


Figure 16: Rendered overlays of 6D Voyager poses during an approach flight over Earth. The red, green and blue lines denote the x, y and z coordinate axes, respectively.

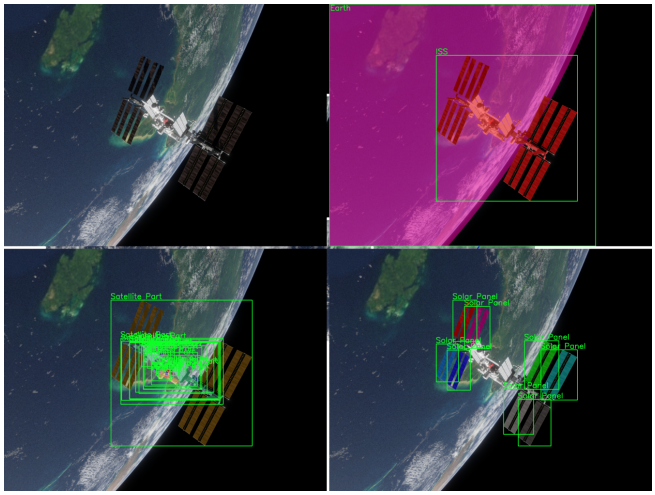


Figure 17: View of the ISS above Earth with different types of annotations. From left to right, top to bottom: Raw RGB image, Semantic Segmentation, Part Segmentation, and Instance Segmentation of the main solar panels. Bounding boxes are added for enhanced visualization. Colors are assigned randomly.

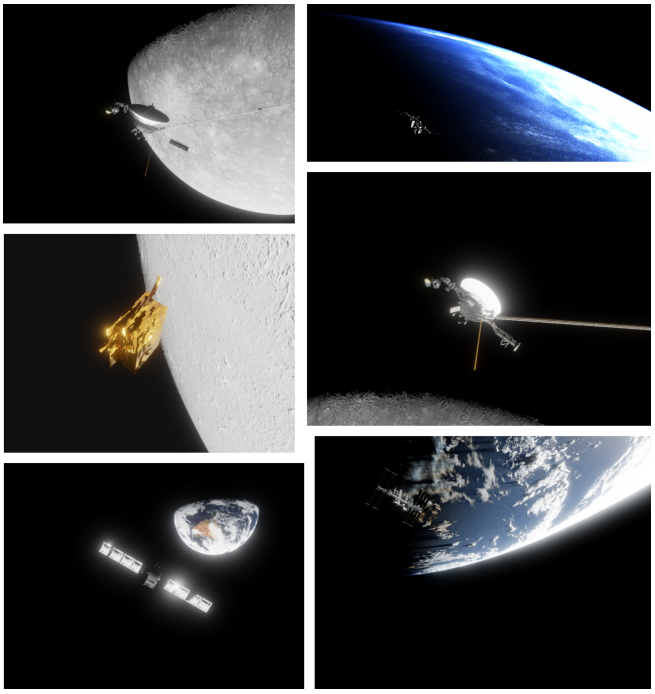


Figure 18: Exemplary renderings of various orbital objects and celestial bodies.

REFERENCES

[1] C. Daehnick, J. Gang, and I. Rozenkopf, "Space launch: Are we heading for oversupply or a shortfall?" 2023, accessed: 2025-09-12. [Online]. Available: https://www.mckinsey.com/industries/aerospace-and-defense/our-insights/space-launch-are-we-heading-for-oversupply-or-a-shortfall

[2] B. Sullivan, D. Barnhart, L. Hill, P. Oppenheimer, B. L. Benedict, G. V. Ommering, L. Chappell, J. Ratti, and P. Will, *DARPA Phoenix Payload Orbital Delivery System (PODs): "FedEx to GEO"*. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2013-5484

[3] B. B. Reed, R. C. Smith, B. J. Naasz, J. F. Pellegrino, and C. E. Bacon, "The restore-l servicing mission," in *AIAA Space 2016 Conference and Exposition*. American Institute of Aeronautics and Astronautics, 2016.

[4] J. L. Forshaw, G. S. Aglietti, N. Navarathinam, H. Kadhem, T. Salmon, A. Pisseloup, E. Joffre, T. Chabot, I. Retat, R. Axthelm, S. Barraclough, A. Ratcliffe, C. Bernal, F. Chaumette, A. Pollini, and W. H. Steyn, "Removedebris: An in-orbit active debris removal demonstration mission," *Acta Astronautica*, vol. 127, pp. 448–463, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S009457651530117X

[5] V. N. Nguyen, S. Tyree, A. Guo, M. Fourmy, A. Gouda, T. Lee, S. Moon, H. Son, L. Ranftl, J. Tremblay, E. Brachmann, B. Drost, V. Lepetit, C. Rother, S. Birchfield, J. Matas, Y. Labbé, M. Sundermeyer, and T. Hodaň, "BOP Challenge 2024 on Model-Based and Model-Free 6D Object Pose Estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, ser. CV4MR, Nashville, TN, USA, 2025, cVPRW 2025.

[6] M. Denninger, D. Winkelbauer, M. Sundermeyer, W. Boerdijk, M. Knauer, K. H. Strobl, M. Humt, and R. Triebel, "Blenderproc2: A procedural pipeline for photorealistic rendering," *Journal of Open Source Software*, vol. 8, no. 82, p. 4901, 2023. [Online]. Available: https://doi.org/10.21105/joss.04901

[7] K. Greff, F. Belletti, L. Beyer, C. Doersch, Y. Du, D. Duckworth, D. J. Fleet, D. Gnanapragasam, F. Golemo, C. Herrmann, T. Kipf, A. Kundu, D. Lagun, I. Laradji, H.-T. D. Liu, H. Meyer, Y. Miao, D. Nowrouzezahrai, C. Oztireli, E. Pot, N. Radwan, D. Rebain, S. Sabour, M. S. M. Sajjadi, M. Sela, V. Sitzmann, A. Stone, D. Sun, S. Vora, Z. Wang, T. Wu, K. M. Yi, F. Zhong, and A. Tagliasacchi, "Kubric: a scalable dataset generator," 2022.

[8] M. Ulmer, L. Klüpfel, M. Durner, and R. Triebel, "How important are data augmentations to close the domain gap for object detection in orbit?" in *2025 IEEE Aerospace Conference, AERO 2025*. IEEE, July 2025, pp. 1–12. [Online]. Available: https://elib.dlr.de/220493/

[9] T. H. Park, M. Märtens, G. Lecuyer, D. Izzo, and S. D'Amico, "Speed+: Next-generation dataset for spacecraft pose estimation across domain gap," in *2022 IEEE aerospace conference (AERO)*. IEEE, 2022, pp. 1–15.

[10] M. Ulmer, M. Durner, M. Sundermeyer, M. Stoiber, and R. Triebel, "6d object pose estimation from approximate 3d models for orbital robotics," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2023*, December 2023, preprint at https://arxiv.org/abs/2303.13241. [Online]. Available: https://elib.dlr.de/197430/

[11] M. G. Müller, M. Durner, A. Gawel, W. Stürzl, R. Triebel, and R. Siegwart, "A Photorealistic Terrain Simulation Pipeline for Unstructured Outdoor Environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2021.

[12] T. Hodaň, M. Sundermeyer, B. Drost, Y. Labbé, E. Brachmann, F. Michel, C. Rother, and J. Matas, "BOP challenge 2020 on 6D object localization," *European Conference on Computer Vision Workshops (ECCVW)*, 2020.

[13] S. Parkes, I. Martin, M. Dunstan, and D. Matthews, "Planet surface simulation with pangu," 05 2004.

[14] R. Brochard, J. Lebreton, C. Robin, K. Kanani, G. Jonniaux, A. Masson, N. Despré, and A. Berjaoui, "Scientific image rendering for space scenes with the surrender software," *arXiv preprint arXiv:1810.01423*, 2018.

[15] P. F. Proença and Y. Gao, "Deep learning for spacecraft pose estimation from photorealistic rendering," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 6007–6013.

[16] L. P. Cassinis, R. Fonod, E. Gill, I. Ahrns, and J. G. Fernandez, *CNN-Based Pose Estimation System for Close-Proximity Operations Around Uncooperative Spacecraft*. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2020-1457

[17] L. Pasqualetto Cassinis, A. Menicucci, E. Gill, I. Ahrns, and M. Sanchez-Gestido, "On-ground validation of a cnn-based monocular pose estimation system for uncooperative spacecraft: Bridging domain shift in rendezvous scenarios," *Acta Astronautica*, vol. 196, p. 123–138, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0094576522001515

[18] K. Black, S. Shankar, D. Fonseka, J. Deutsch, A. Dhir, and M. R. Akella, "Real-time, flight-ready, non-cooperative spacecraft pose estimation using monocular imagery," *arXiv preprint arXiv:2101.09553*, 2021.

[19] B. O. Community, *Blender - a 3D modelling and rendering package*, Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. [Online]. Available: http://www.blender.org

[20] M. Kisantal, S. Sharma, T. H. Park, D. Izzo, M. Märtens, and S. D'Amico, "Satellite pose estimation challenge: Dataset, competition design, and results," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 56, no. 5, pp. 4083–4098, 2020.

[21] T. H. Park, M. Märtens, M. Jawaid, Z. Wang, B. Chen, T.-J. Chin, D. Izzo, and S. D'Amico, "Satellite pose estimation competition 2021: Results and analyses," *Acta Astronautica*, 2023.

[22] M. G. Müller, W. Boerdijk, A. G. Hernandez, L. Klüpfel, M. Durner, and R. Triebel, "Bridging the data gap of asteroid exploration: Oaisys extention for synthetic asteriods creation," in *2025 IEEE Aerospace Conference*, 2025, pp. 1–13.

BIOGRAPHY

Marcus G. Müller is a researcher in the department of "Perception and Cognition" at the German Aerospace Center (DLR) since 2016 and Ph.D. student at ETH Zurich. He is the leader of the MAV Exploration Team at the Institute of Robotics and Mechatronics (DLR-RM), where he is working on autonomous navigation algorithms for MAVs. Before joining DLR he conducted research at the Jet Propulsion Laboratory (JPL) of NASA in Pasadena, USA, where he worked on visual inertial navigation for MAVs and on radar signal processing. Marcus received his Master's and Bachelor's degree in Electrical Engineering from the University of Siegen, Germany.

Wout Boerdijk is a PhD student at the Technical University of Munich (TUM) and a research scientist at the German Aerospace Center (DLR), where he is part of the Perception and Cognition department in the Institute of Robotics and Mechatronics. Before joining DLR in 2019, he received his Master's degree in Robotics, Cognition and Intelligence at TUM. His research interests include computer vision methods for learning of and interacting with objects, and their applicability in the space context.

Maximilian Ulmer received his B.Sc. and M.Sc. in Electrical Engineering and Information Technology from the Technical University of Munich (TUM). He is currently a Ph.D. student at the Karlsruhe Institute of Technology (KIT) and a research scientist at the German Aerospace Center (DLR) in the department for Perception and Cognition at the Institute of Robotics and Mechatronics. His research focuses on closing the domain gap for orbital perception, 3D vision, and object-centric computer vision for robotic manipulation.

Wolfgang Stürzl is a senior research scientist in the department of "Perception and Cognition" at the Institute of Robotics and Mechatronics of the German Aerospace Center (DLR). His research interests include computer vision for mobile robots, in particular using multi-camera and wide-angle imaging systems, and bio-inspired visual navigation of flying systems.

Abel Gawel is currently Research Project lead at the Robotics and AI (RAI) institute. Before that, he was Principal Researcher in Computer Vision and Machine Learning with Huawei Zurich, and Senior Scientist at the Autonomous Systems Lab of ETH Zurich. He received the PhD from ETH Zurich in 2018 and was a visiting Postdoctoral Fellow in the CRI group at NTU Singapore in 2019. His research interests include world models, semantic scene understanding, and SLAM with application in field robotics. Prior to joining ETH in 2014, he worked for Bosch Corporate Research and the BMW group.

Roland Siegwart Roland Siegwart (1959) is full Professor of Autonomous Systems at ETH Zurich since July 2006 and Founding Co-Director of the Wyss Zurich. From January 2010 to December 2014, he took office as Vice President Research and Corporate Relations in the ETH Executive Board. He is member of the board of directors of various companies, including Komax and NZZ. He received his Diploma in Mechanical Engineering in 1983 and his Doctoral Degree in 1989 from ETH Zurich. He brought up a spin-off company, spent ten years as professor at EPFL Lausanne (1996-2006) and held visiting positions at Stanford University and NASA Ames.

Rudolph Triebel received his PhD in 2007 from the University of Freiburg in Germany. The title of his PhD thesis is "Three-dimensional Perception for Mobile Robots". From 2007 to 2011, he was a postdoctoral researcher at ETH Zurich, where he worked on machine learning algorithms for robot perception within several EU-funded projects. Then, from 2011 to 2013 he worked in the Mobile Robotics Group at the University of Oxford, where he developed unsupervised and online learning techniques for detection and classification applications in mobile robotics and autonomous driving. From 2013 to 2023, Rudolph worked as a lecturer at TU Munich, where he teached master level courses in the area of Machine Learning for Computer Vision. In 2015, he was appointed as leader of the Department of Perception and Cognition at the Robotics Institute of DLR, and in 2023 he was appointed as a university professor at Karlsruhe Institute of Technology (KIT) in "Intelligent Robot Perception".

Maximilian Durner is a research fellow at the Institute of Robotics and Mechatronics, German Aerospace Center (DLR) since 2016 and a Ph.D. student at the Technical University of Munich (TUM). Before, he studied Electrical Engineering at the TUM, partially studying at the Politecnico di Torino, Italy, and the Universidad Nacional de Bogota, Colombia. Currently, he is leading a research group on semantic scene analysis in the Perception and Cognition Department at DLR, which is working on robotic perception providing semantic information to interact with the surroundings. This includes known, unseen, and unknown object detection, pose estimation, and tracking. In this context, Maximilian focuses on robust and continuously adapting object-centric perception for mobile manipulation.