FLANDERS MAKE

DRIVING INNOVATION IN MANUFACTURING

University of Antwerp

Cosys-lab | Co-Design of Cyber Physical Systems

Tutorial on Tools and Applications using Cosys-AirSim: A Real-Time Simulation Framework Expanded for Complex Industrial Applications

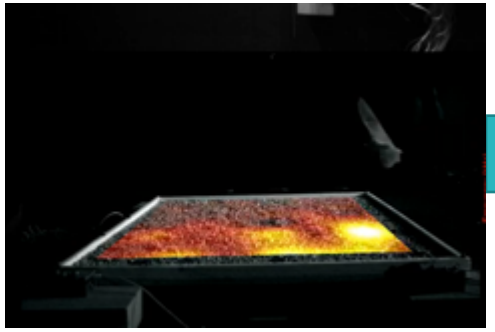
Flanders Make - Driving Innovation in Manufacturing

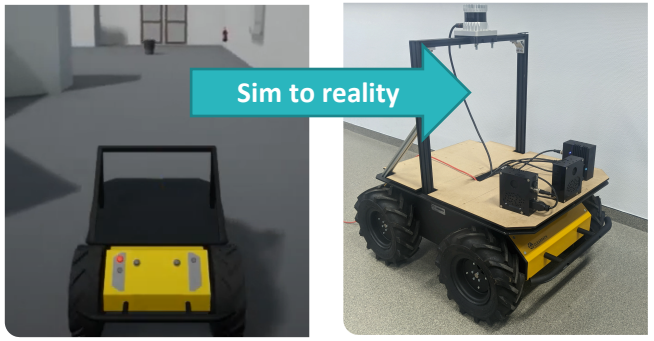
- Research center in Flanders, Belgium
- Innovation and technological development in manufacturing industry
- Specialize in AI, robotics, mechatronics, and more
- Work with companies of all sizes to overcome challenges, streamline operations, and create innovative products
- State-of-the-art facilities and research
- Collaborative approach to drive innovation in manufacturing

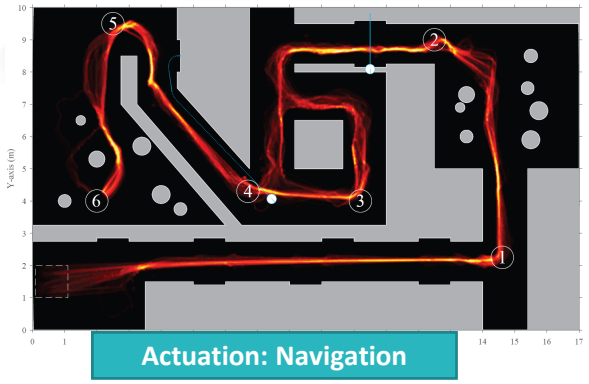
FLANDERS MAKE
DRIVING INNOVATION IN MANUFACTURING

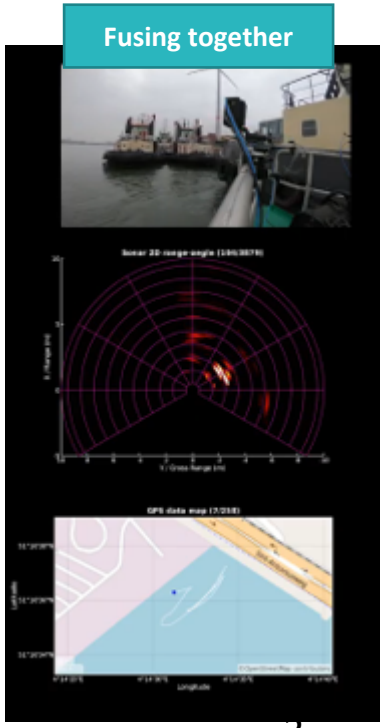
Cosys-Lab - Learn from nature for sensing and actuating

- Research group, Applied Engineering, University of Antwerp, Belgium
- Co-Design for Cyber-Physical Systems
- Development of novel sensor techniques and embedded systems
  - Industrial sensing in complex environments
  - Inspired by nature









University of Antwerp
Cosys-lab | Co-Design of Cyber Physical Systems

Overview

Tutorial on **Tools** and **Applications** using **Cosys-AirSim**: A Real-Time Simulation Framework Expanded for Complex Industrial Applications

CONTEXT

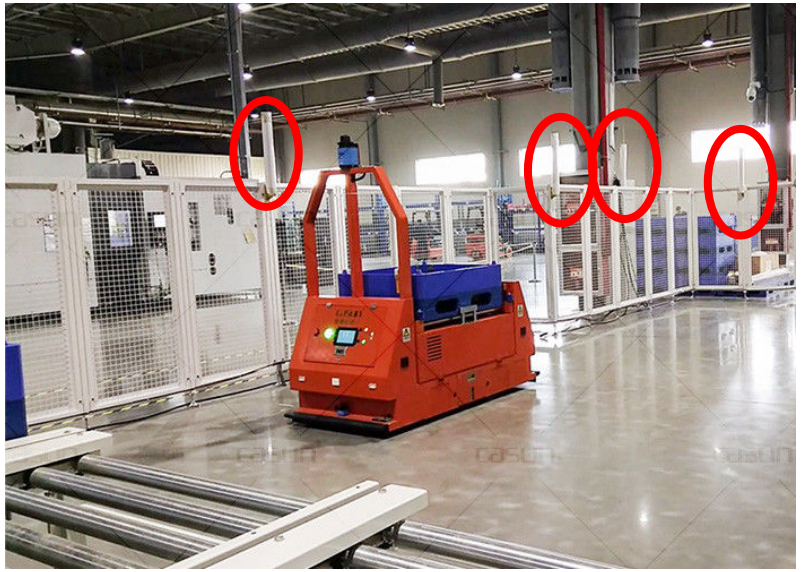
Evolution of indoor localization

Most common localization techniques

Flexibility

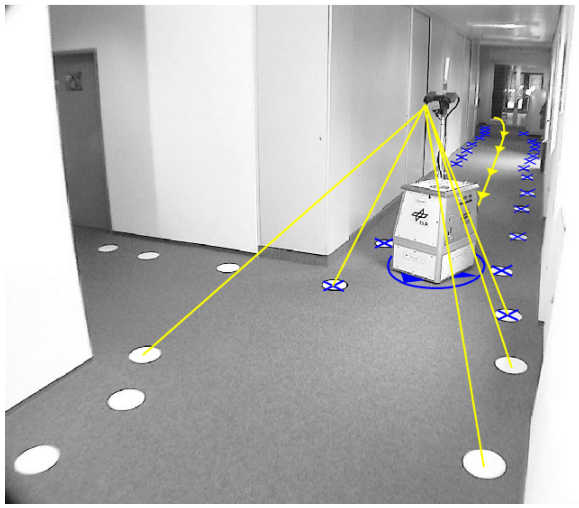
1950-1990

- **Active inductive track guidance**



1990-2010

- **Laser-based navigation (or magnets)**



~2010-…

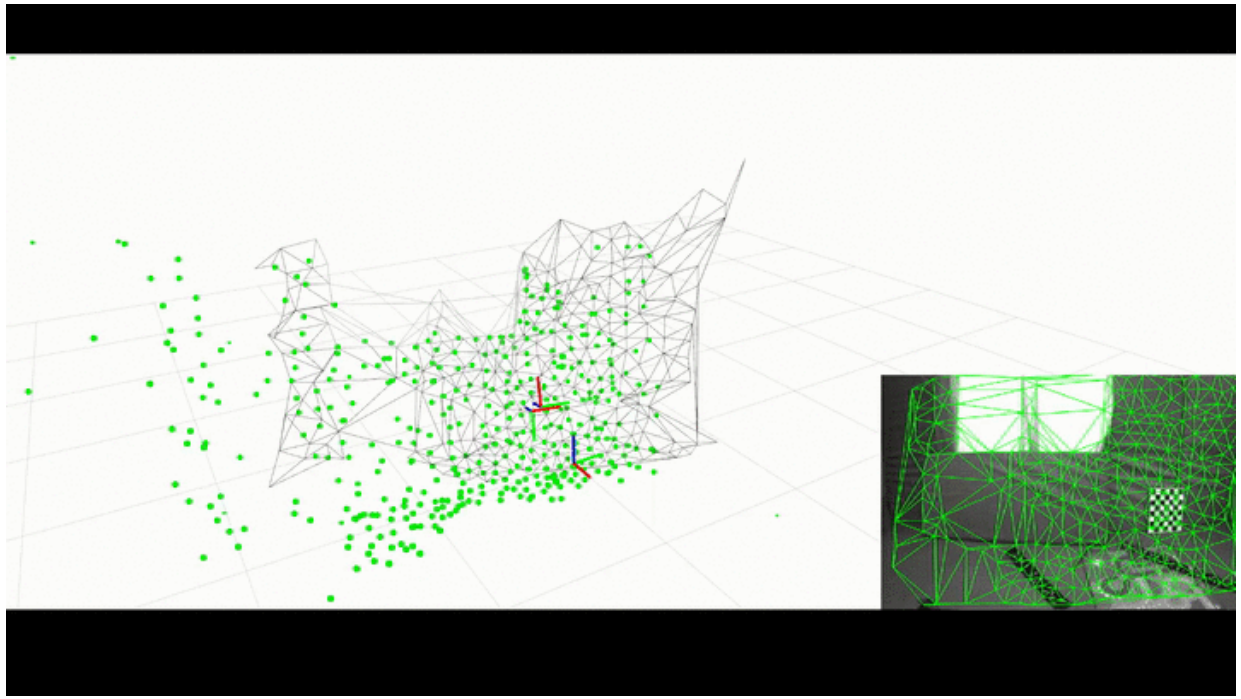
- **SLAM (Simultaneous Localization And Mapping)**

Time

Robustness becomes more challenging over the years

What is SLAM?

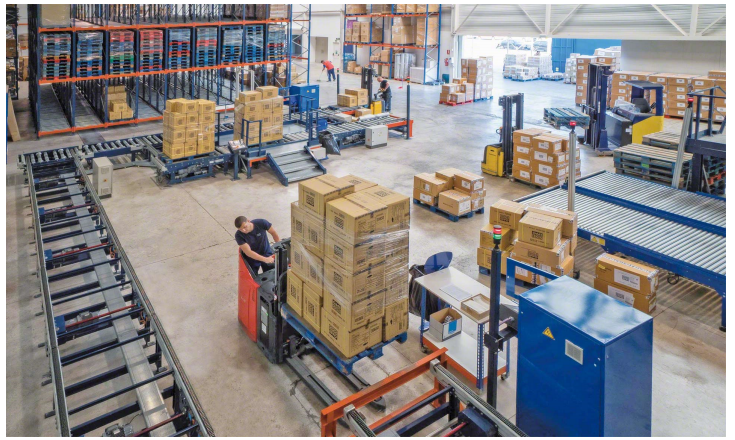
- **S**imultatenous **L**ocalization **A**nd **M**apping
- “Using onboard sensors, map the environment and localize yourself within this map”



Kimera: An Open-Source Library for Real-Time Metric-Semantic Localization and Mapping by Rosinol et al

Why use simulation ?

- Very complex algorithms that require frequent testing during development
- Multiplicity of environments (logistic, industrial, outdoor)
- Large number of scenarios and vehicles to be tested
- AI models require a large amount of annotated data for training
- Difficulty to reproduce scenario and get location ground truth in real life



Challenges: Large dataset generation

Problem

- Visual odometry can be tricked by the motion of surrounding object

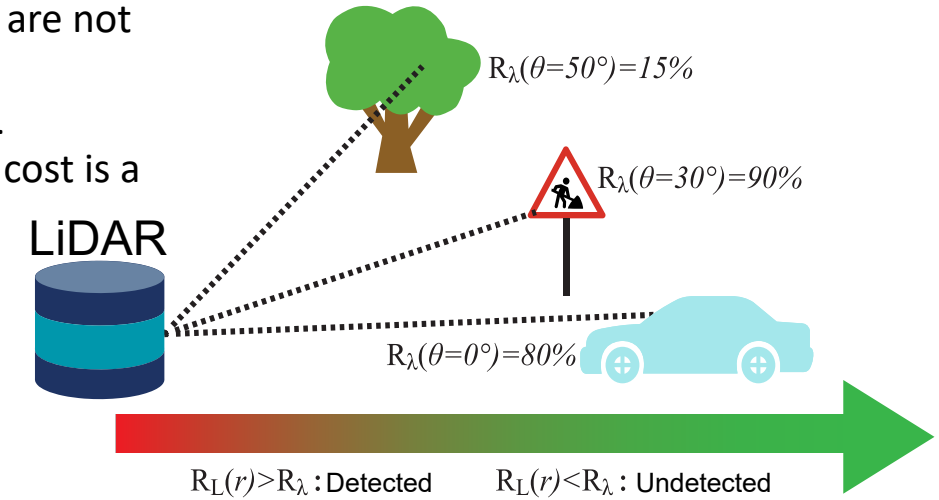
Solution

- Use AI model to distinguish static objects from movable object and rule them out for localization
- Generate synthetic data to train the AI model in representative environment

Challenges: Multi sensor modelling

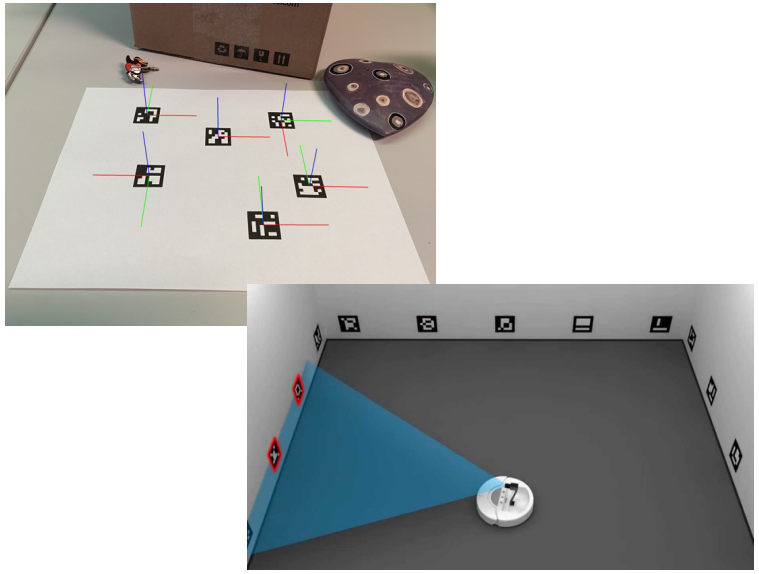
Problem

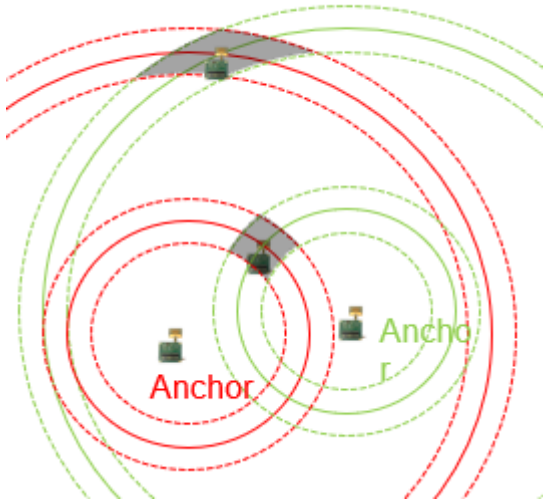
- Ideal localization relies on multiple type of input sensor and information which are not always available in simulation:
  - Ultra Wide Band (UWB), WiFi, Fiducial markers, Cameras, Lidars, Radars, Pulse Echo…
- Placing these sensors correctly to maximize localization accuracy and minimize cost is a challenge



Solution

- Testing multiple sensor configuration easily and rapidly
- Represent the different sensor behavior in simulation
- Generate maps for ideal placements





Delusion of precision

Challenges: Sensor realism

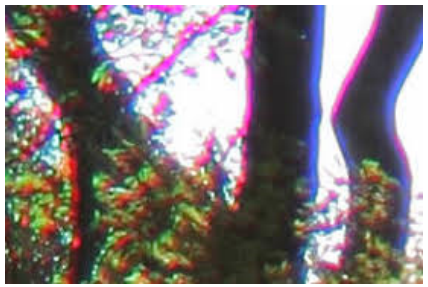
Problem

- Simulation is too perfect, not physically accurate
- Perfect sensor results leads to more accurate visual odometry which is not representative

Solution

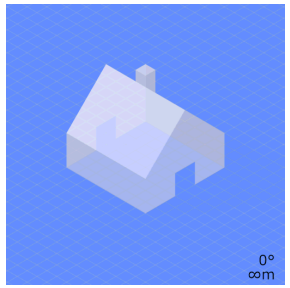
- Bring imperfection to the simulation
- Easily represent multiple type of imperfect and allow tuning based on real sensor characteristics

Chromatic aberration



https://www.camerastuffreview.com

Radial lens distortion

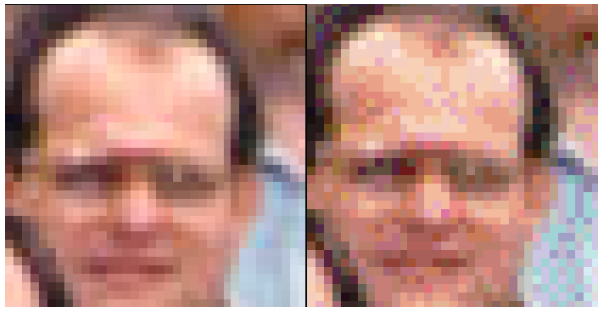


SharkD, Wikimedia, CC BY-SA 3.0

Motion Blur

Yoichi Sato

Compression noise



Overexposure

Kate Mereand, Wikimedia, CC BY-SA 2.0

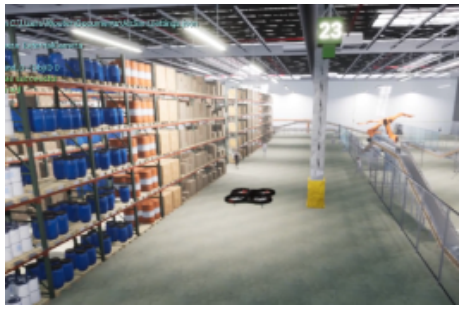
Challenges: Multi vehicle modelling

Problem

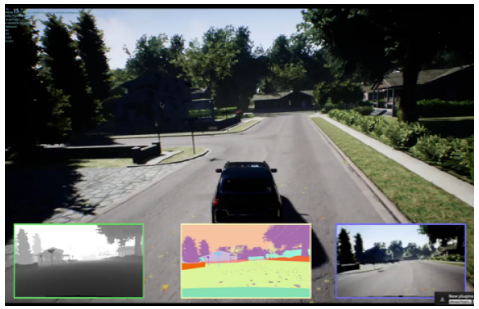
- Multiple types of vehicle with different dynamics are needed and not available in simulation

Solution

- Create new physical model representative of existing vehicle
- Integration of these model in the different simulation environments



Drones



Car

Differential drive

Skid drive

Tractor digital twin

Challenges: High resolution simulation

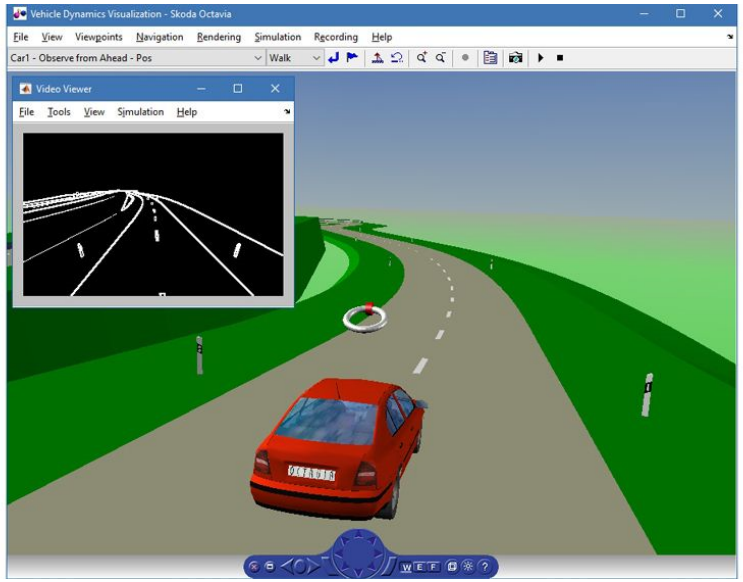
Problem

- Many available simulation platforms are ‘low-resolution’
- This does not allow for transfer learning (learning ML models on simulated data)

Solution

- Choose/create a simulation framework using a high-resolution renderer

Gazebo http://gazebosim.org/



MATLAB Simulink renderer (old-version), Mathworks https://nl.mathworks.com/products/3d-animation.html

TOOLS

Choice of simulation engine

Commercial

- Siemens Prescan
- ANSYS Avxcelerate
- Vector DYNA4
- VIRES VTD
- MathWorks MATLAB/Simulink
- NVIDIA DriveWorks
- rFpro
- dSPACE AURELION
- Microsoft Project AirSim

Open-source

- CARLA
- (Cyberbotics) Webots
- Gazebo
- (LG) LGSVL
- **(Microsoft) AirSim**

Conclusions

- Heavy focus on ADAS/AD
- Not always high-resolution or dynamic
- Licensing models not ideal
- Rise of gaming engine renderers
- Commercial not extendable or modular
- Open-source is not forever supported

Simulation framework presentation

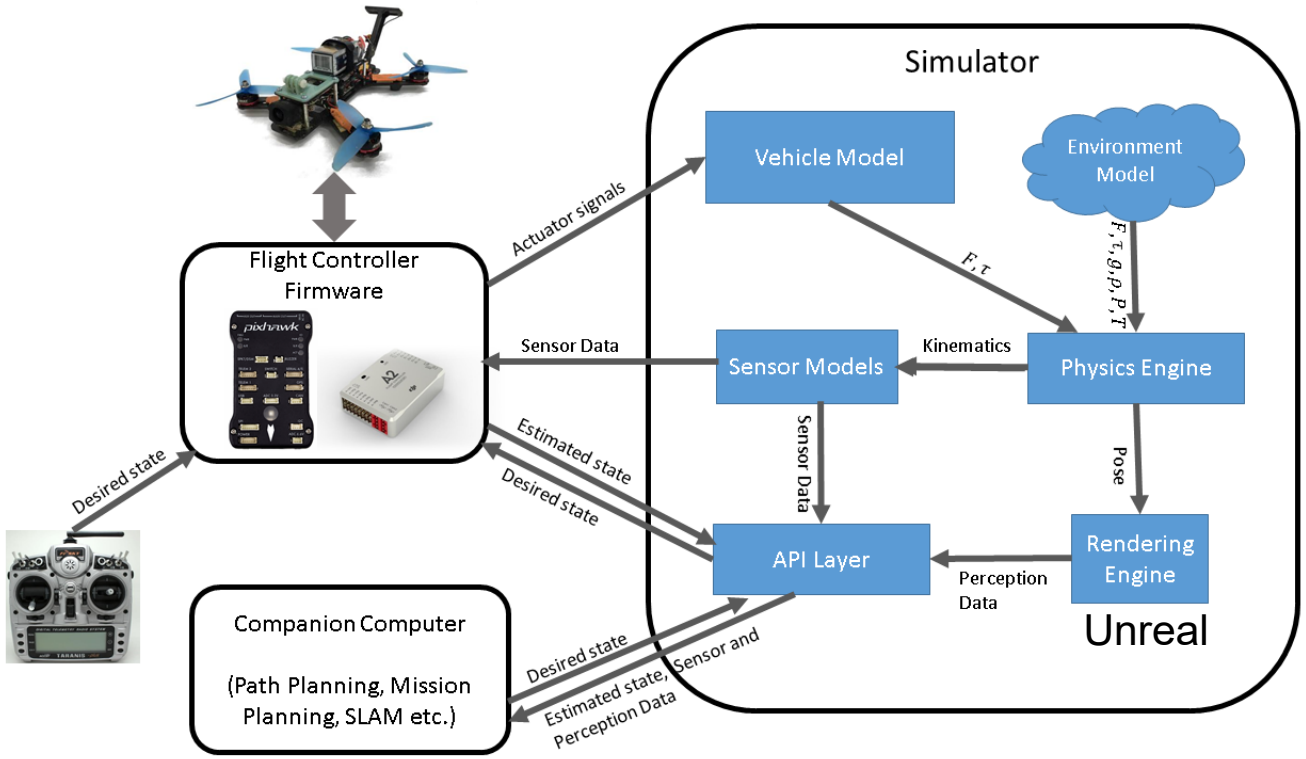
Unreal Engine

- Opensource and user friendly with large community
- Exceptional visual rendering quality
- Extensible and compatible platform (Airsim, Ros…)
- Robust physics engine with object interactions, collisions, and dynamics
- Downside: no fixed-step simulation = all simulation must be real-time

Simulation framework presentation

Microsoft AirSim

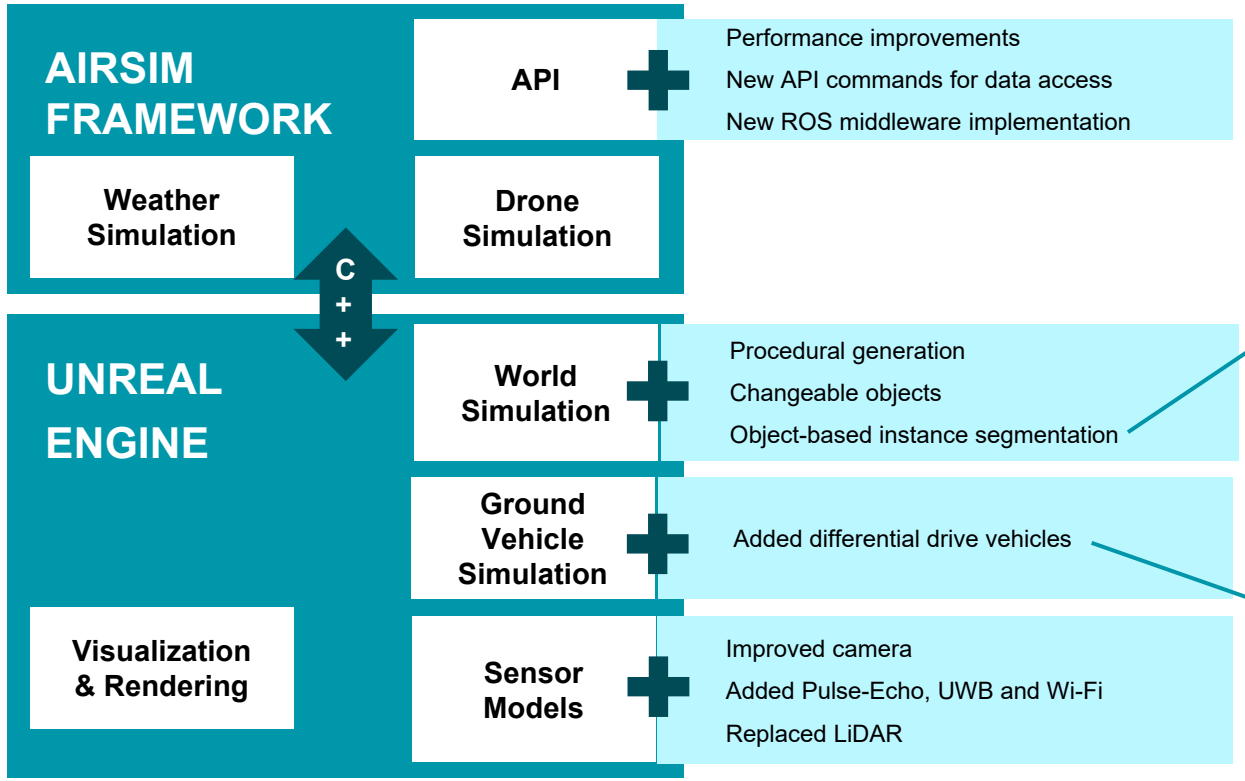
- Physically and visually realistic simulations
- Physics engine that can operate at a high frequency for real-time hardware-in-the-loop (HITL)
- Support for popular protocols and APIs
- Built-in sensor simulation allowing for realistic perception and sensor data generation
- Open source
- Extensible, modular design, various vehicle types

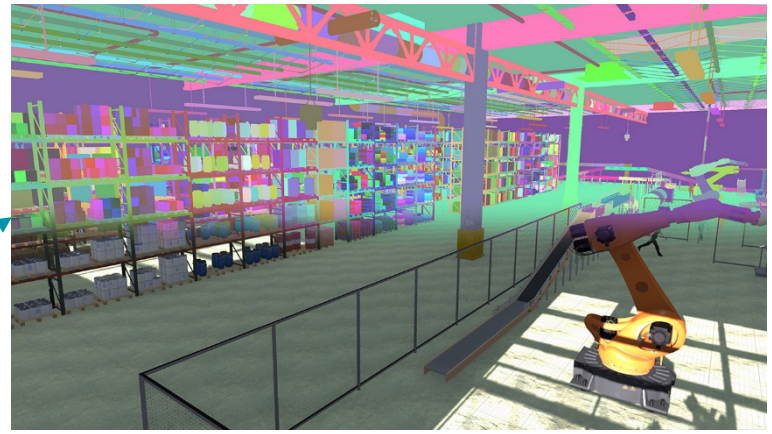


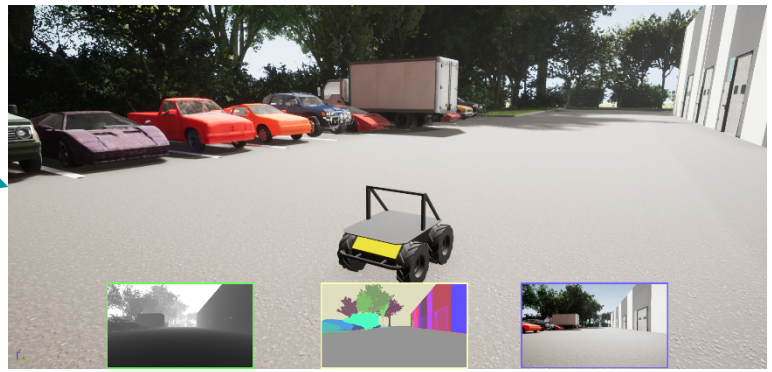
Shital et al 2017, AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles

Cosys-Lab contributions to the simulation framework

- Heavily extended the open-source **AirSim framework**
- High detail, real-time simulation of
  - Vehicles
  - sensors
  - environments







Tools:

Installation & usage

Installation of the Cosys AirSim simulator

Available at github.com/Cosys-Lab/Cosys-AirSim

Linux	Windows (recommended)
Native support of ROS environment for data recording and API control	Requires installation and use of WSL (Windows Subsystem for Linux) to use ROS
Less Unreal features and plugin availability	All plugins available
Easier setup (but less stable)	Visual Studio 2019 is required for building (see link below)
https://github.com/Cosys-Lab/Cosys-AirSim/blob/main/docs/build_linux.md	https://github.com/Cosys-Lab/Cosys-AirSim/blob/main/docs/build_windows.md

Accessing Unreal Engine source code on GitHub: https://www.unrealengine.com/en-US/ue-on-github

Visual studio **2019**:
https://my.visualstudio.com/Downloads?q=visual%20studio%202019&wt.mc_id=o~msft~vscom~older-downloads (visual studio community 2019)

CLICK HERE

CLICK HERE

Setting up blocks environment and configuring settings file

CLICK HERE

Settings and setup

Setup environment:
https://github.com/Cosys-Lab/Cosys-AirSim/blob/main/docs/unreal_proj.md

Settings README:
https://github.com/Cosys-Lab/Cosys-AirSim/blob/main/docs/settings.md

Example settings.json file:
https://github.com/Cosys-Lab/Cosys-AirSim/blob/main/docs/settings.json

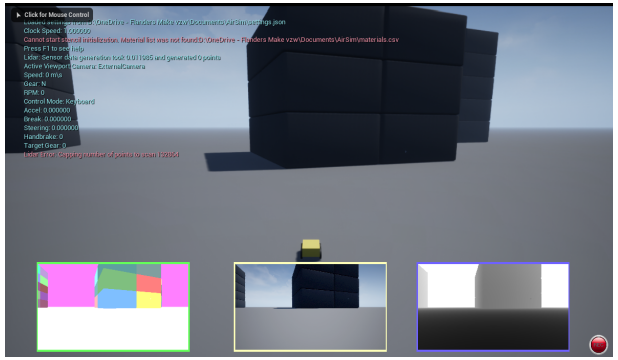
ROS launch files associated to this settings.json file:
https://github.com/Cosys-Lab/Cosys-AirSim/tree/main/ros/python_ws/src/airsim/launch

WSL setup README:
https://github.com/Cosys-Lab/Cosys-AirSim/blob/main/docs/ROS_on_Windows_with_WSL.md

Vehicles settings.json file

"SimMode": "Car",

"VehicleType": "BoxCar",

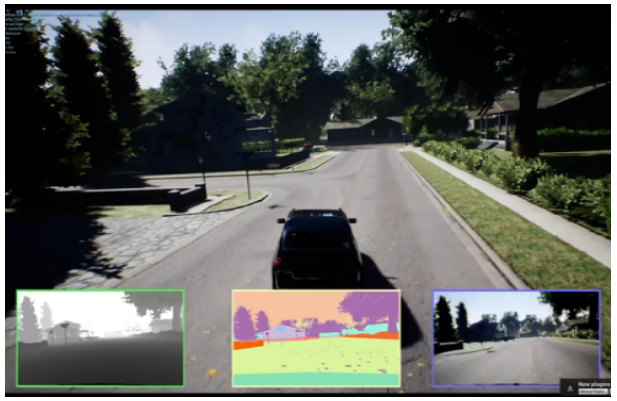


"SimMode": "SkidVehicle",

"VehicleType": "Pioneer",

"SimMode": "Car",

"VehicleType": "PhysXCar,

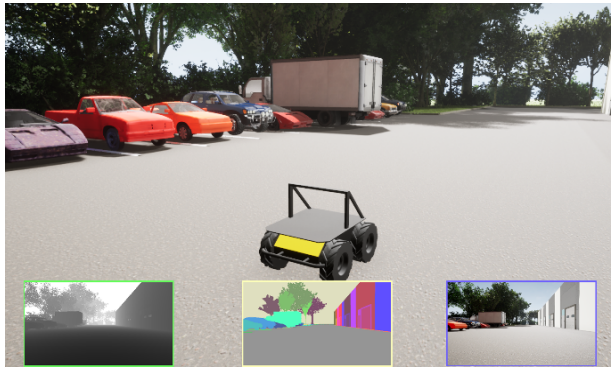


"SimMode": "Multirotor",

"VehicleType": "SimpleFlight",

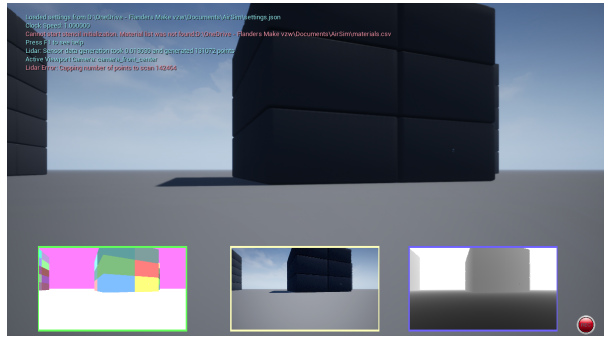
"SimMode": "SkidVehicle",

"VehicleType": "CPHusky",



"SimMode": "ComputerVision",

"VehicleType": "ComputerVision",



AirSim Camera names (settings.json)

Available Cameras

These are the default cameras already available in each vehicle. Apart from these, you can add more cameras to the vehicles and external cameras which are not attached to any vehicle through the settings.

Car

The cameras on car can be accessed by following names in API calls: `front_center`, `front_right`, `front_left`, `fpv` and `back_center`. Here FPV camera is driver's head position in the car.

Multirotor

The cameras on the drone can be accessed by following names in API calls: `front_center`, `front_right`, `front_left`, `bottom_center` and `back_center`.

Computer Vision Mode

Camera names are same as in multirotor.

Backward compatibility for camera names

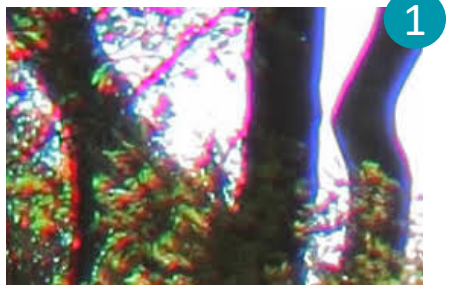
Before AirSim v1.2, cameras were accessed using ID numbers instead of names. For backward compatibility you can still use following ID numbers for above camera names in same order as above: `"0"`, `"1"`, `"2"`, `"3"`, `"4"`. In addition, camera name `""` is also available to access the default camera which is generally the camera `"0"`.

Tools:

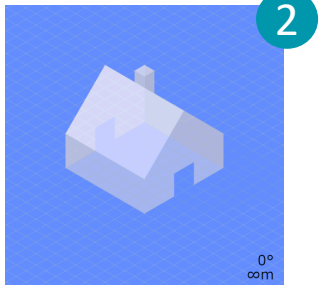
Sensor models

Sensor models

- Existing Sensors
  - Camera, Barometer, IMU, GPS, Magnetometer, LiDAR
- Improved sensors
  - LiDAR
  - Camera:
    1. Chromatic aberration
    2. Radial lens distortion
    3. Motion blur
    4. Compression noise
    5. Overexposure



https://www.camerastuffreview.com



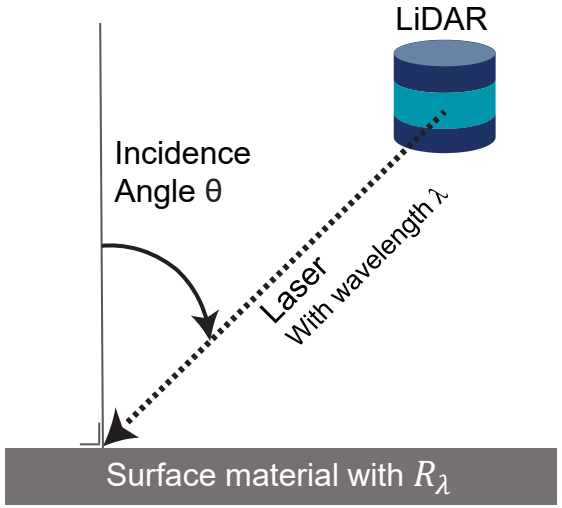
SharkD, Wikimedia, CC BY-SA 3.0

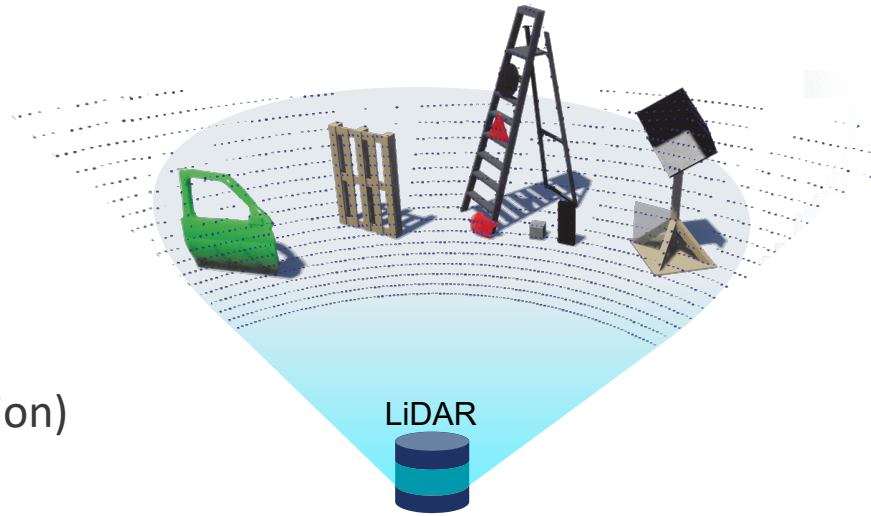
Yoichi Sato

- Added Sensors
  - Echo (RADAR/SONAR)
  - UWB & Wi-Fi

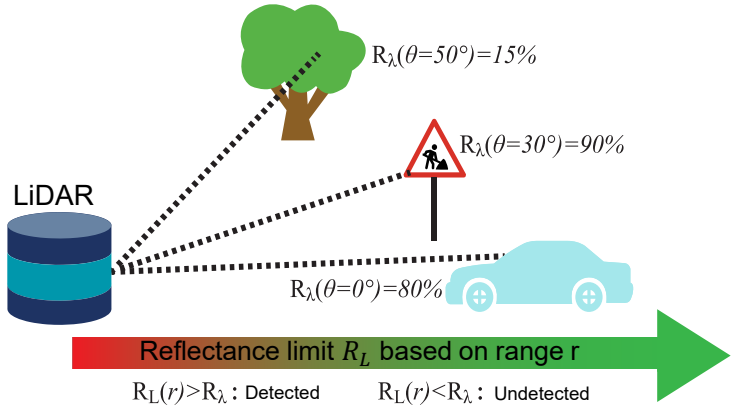
LiDAR sensor

- Influence of surface material and incidence angle on returned reflection power





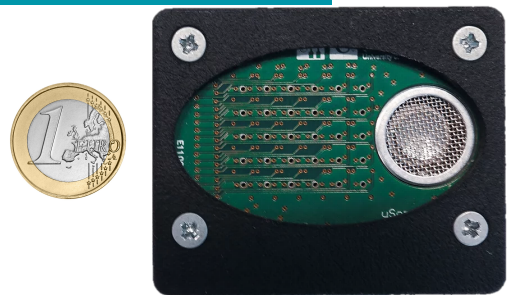
- Influence of capability of LiDAR sensor (based on real datasheet information)



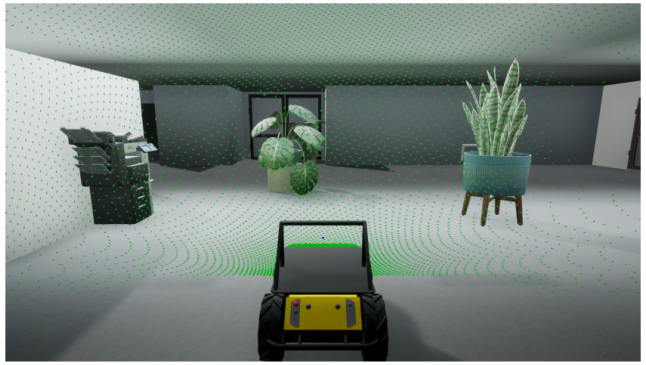
- Influence of rain on accuracy/precision and returned reflection power

Pulse echo sensor

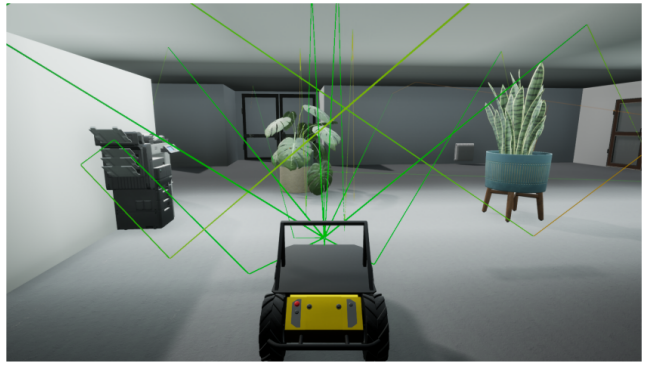
- Biological echolocation
- RADAR/SONAR
- Physical model that works in real-time
- Using collision ray-tracing



www.3dsonar.eu

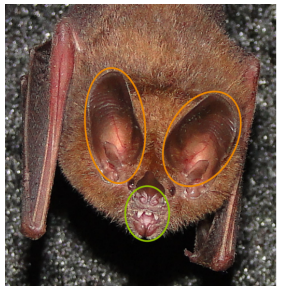


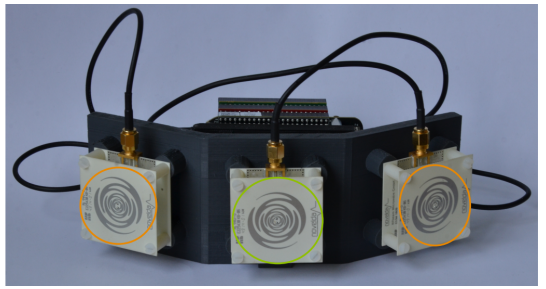
(a)



(b)

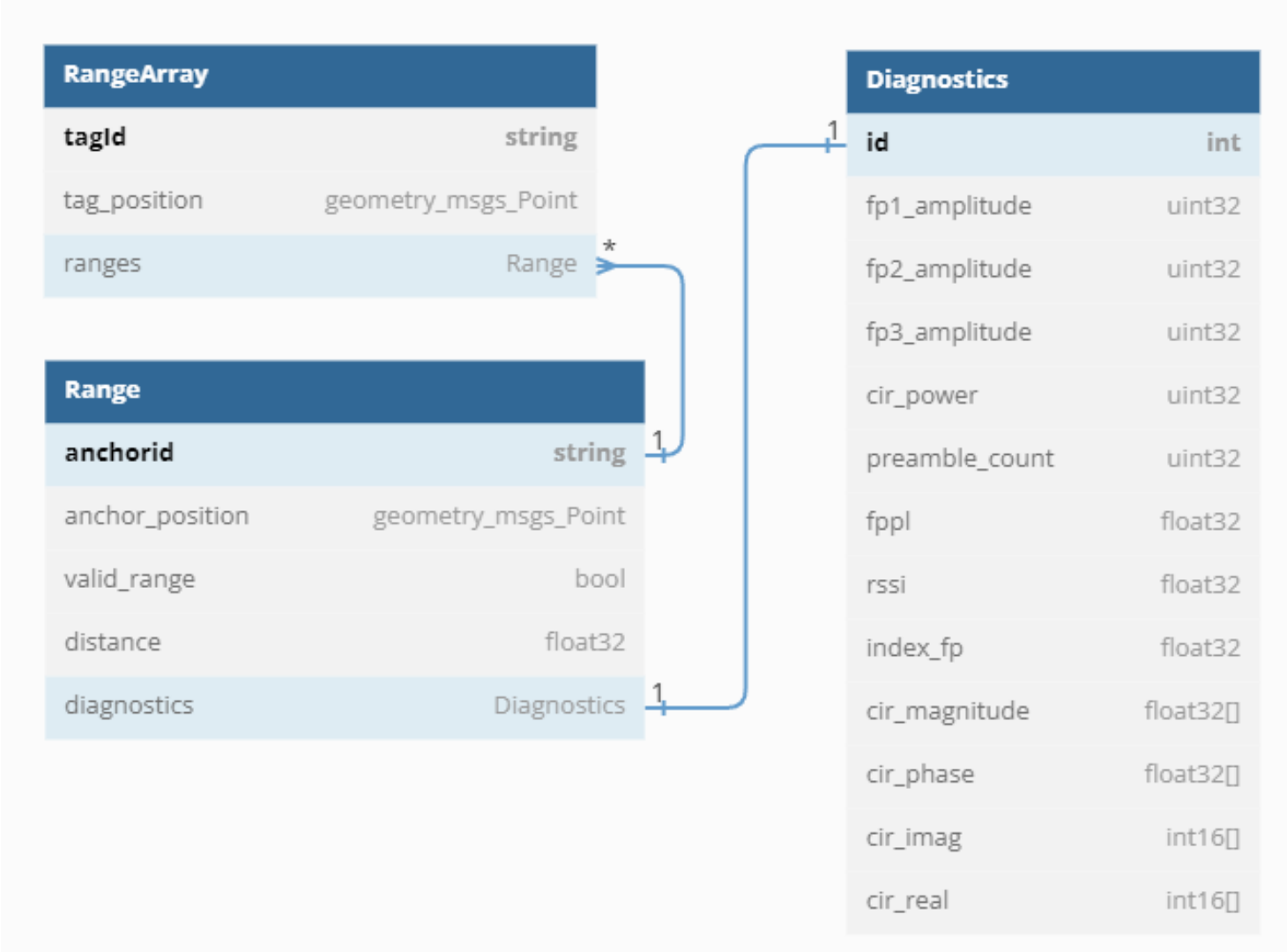
(c)





UWB & Wi-Fi

- Beacons/Anchors
  - Static
- Sensor/Tag
  - On the vehicle
- Localization
  - ToF
  - RSSI
- Ranges
  - Via API
  - Available in ROS

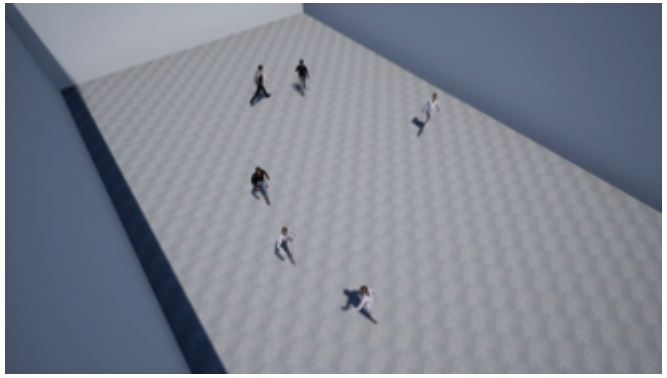


Tools:

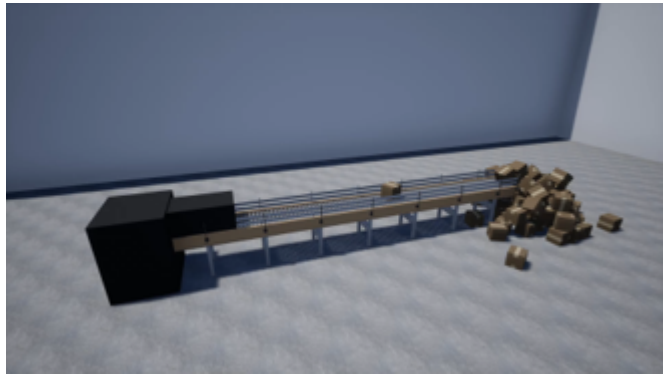
World modeling & API

Realistic environments

Changeable objects

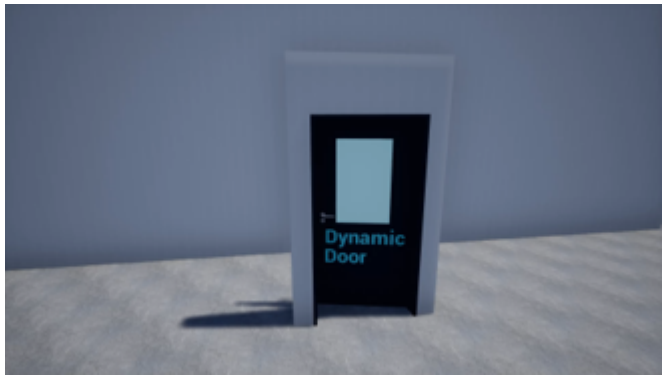


AI characters

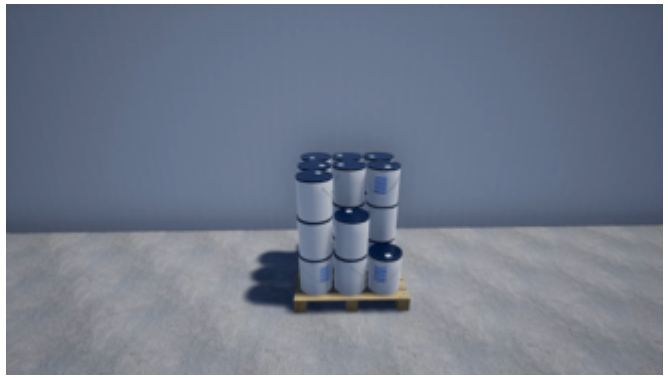


Conveyor belts

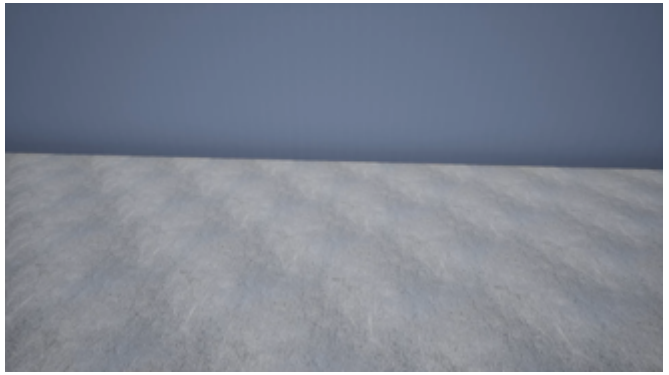
Robotic arms



Dynamic object states



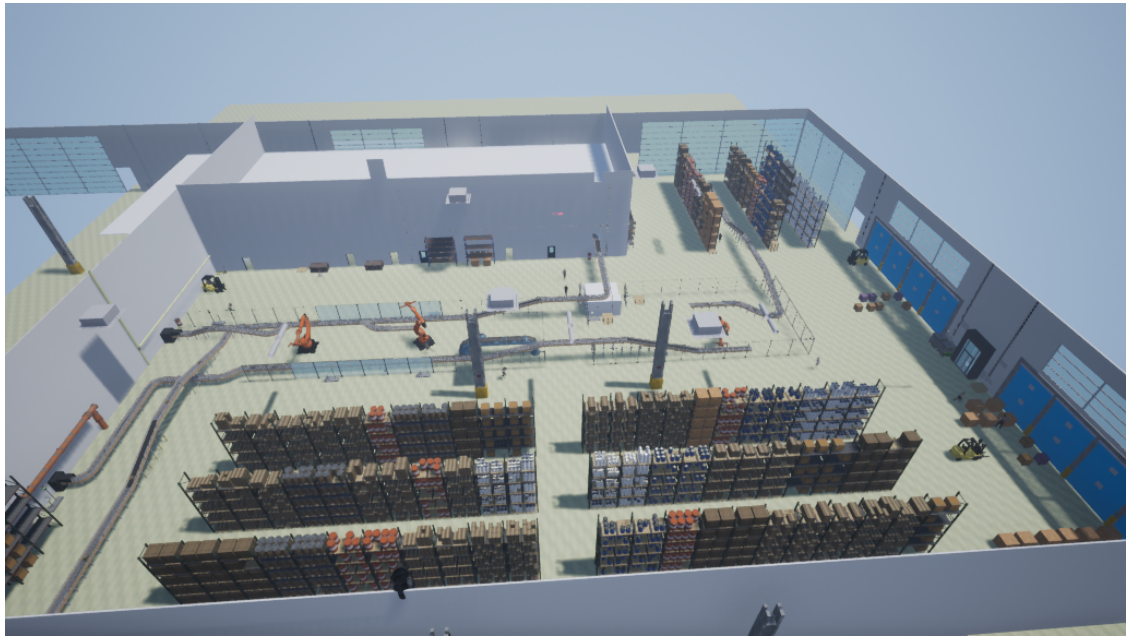
Dynamic pallets

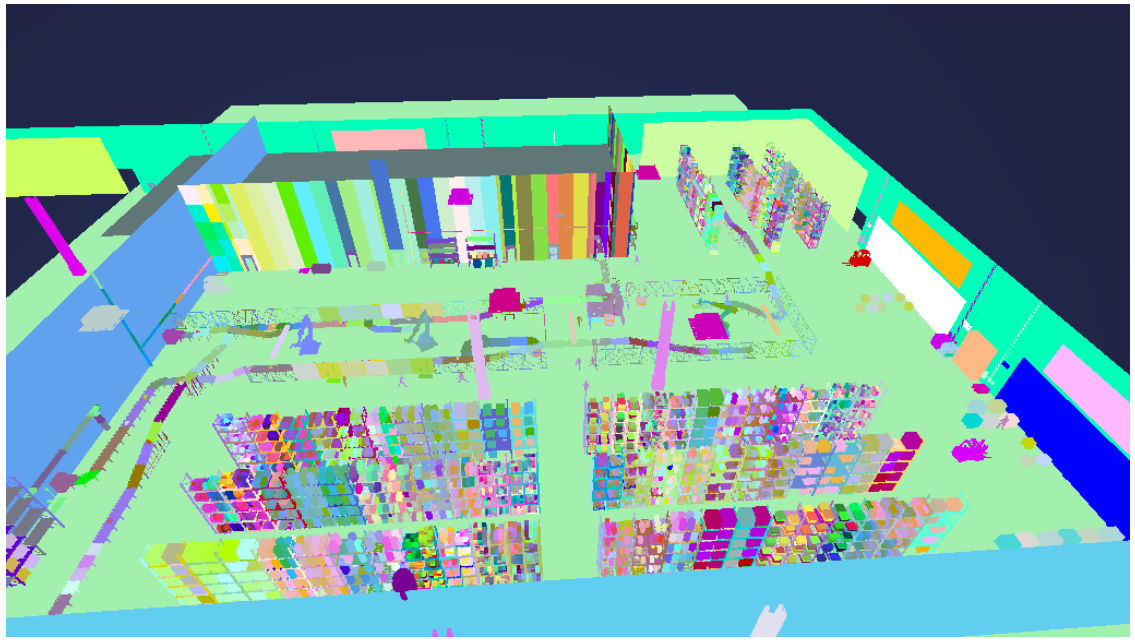


Dynamic object creation

Instance segmentation

- Created custom fork of Unreal Engine (v4.24.4) to make it work
- 255 colors, class-limited (original AirSim) ⇔ 2744000 colors (our AirSim)
- For both camera and LiDAR sensor types



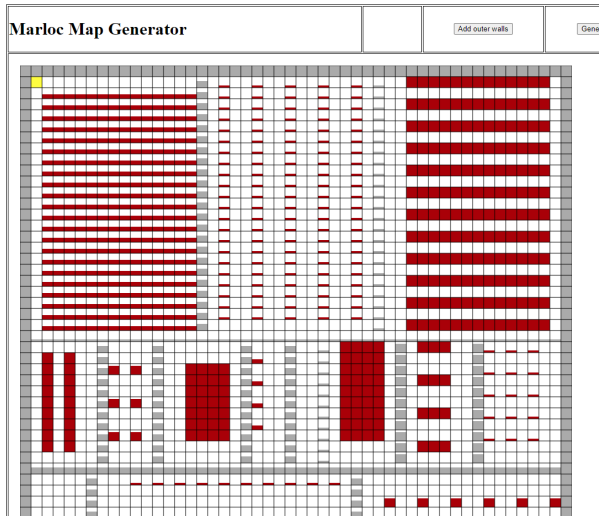


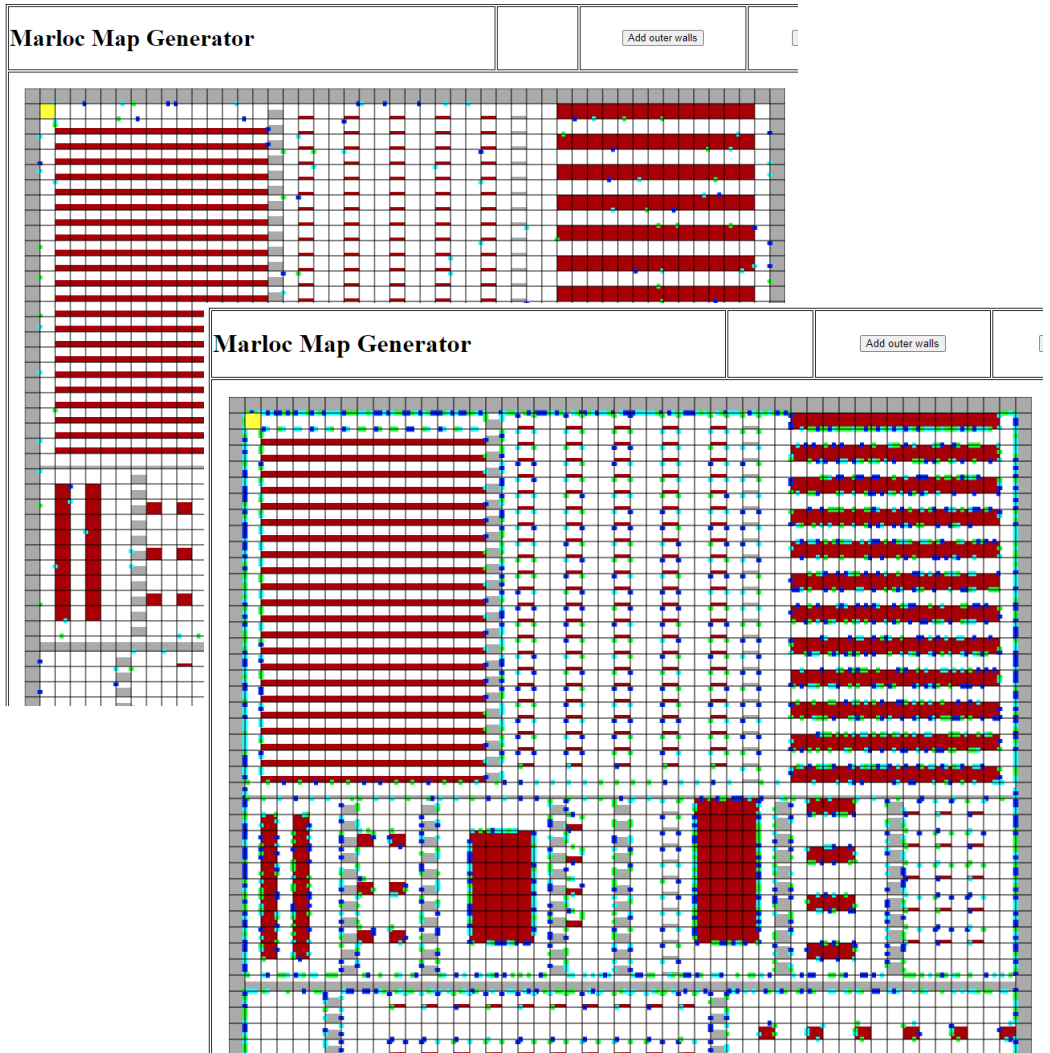
Map Generation

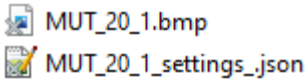
Generate Map
- Manual (HTML Tool)
- Third party software
- …

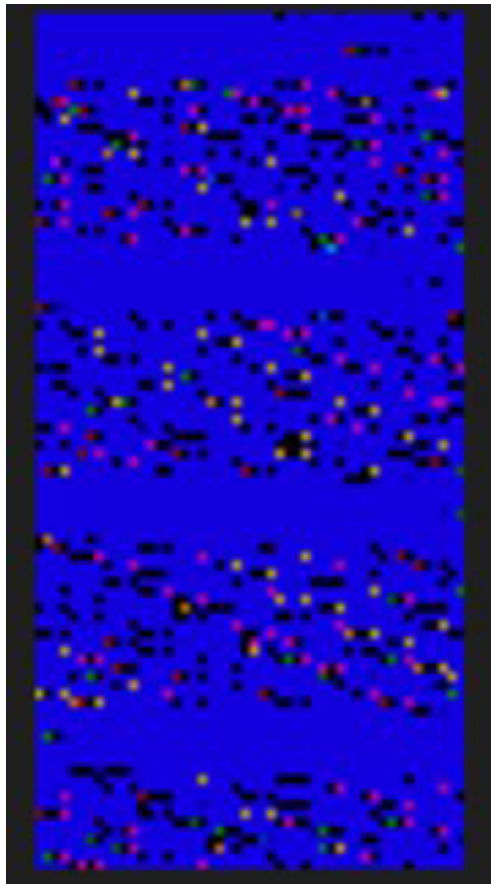
Adding Anchors (UWB, Wifi, Aruco)
- Third party software
- Varying fill levels

Output
- Image (NN Input)
- Settings (AirSim Input)









API

- RPC based API
  - C++ and python support
  - Sensor data queries
  - Vehicle data queries
  - Environment data queries
- Complete rewrite of ROS Python API
  - Subscribe/publish messaging
  - Sensor, vehicle and environmental data capture

ROS API, Sensors, Instance Segmentation, Changeable objects: demo

CLICK HERE

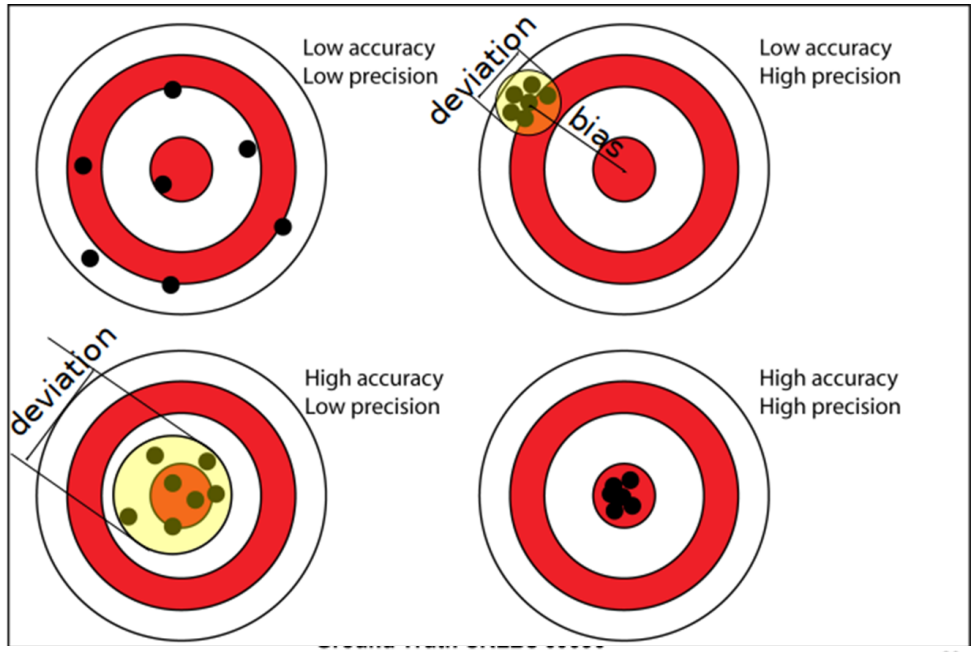
APPLICATIONS

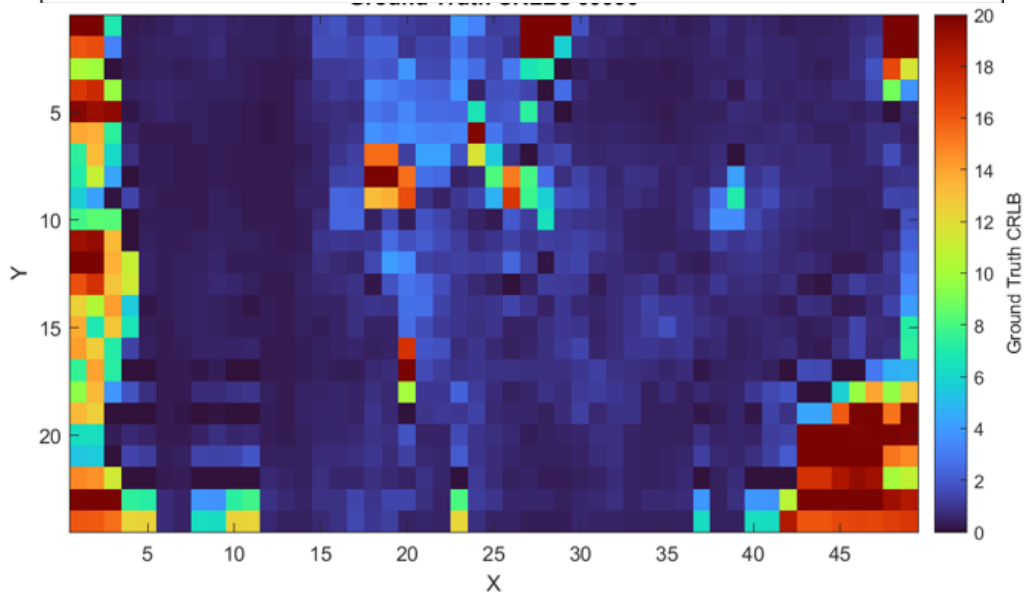
Application:

Ideal sensor placement

Ideal sensor/beacon placements

- Optimize sensor/beacon placement
  ➡ Evaluate CRLB
- Lengthy calculation



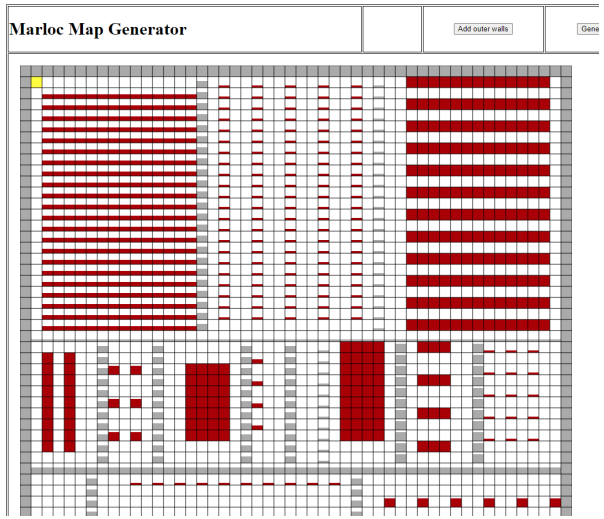


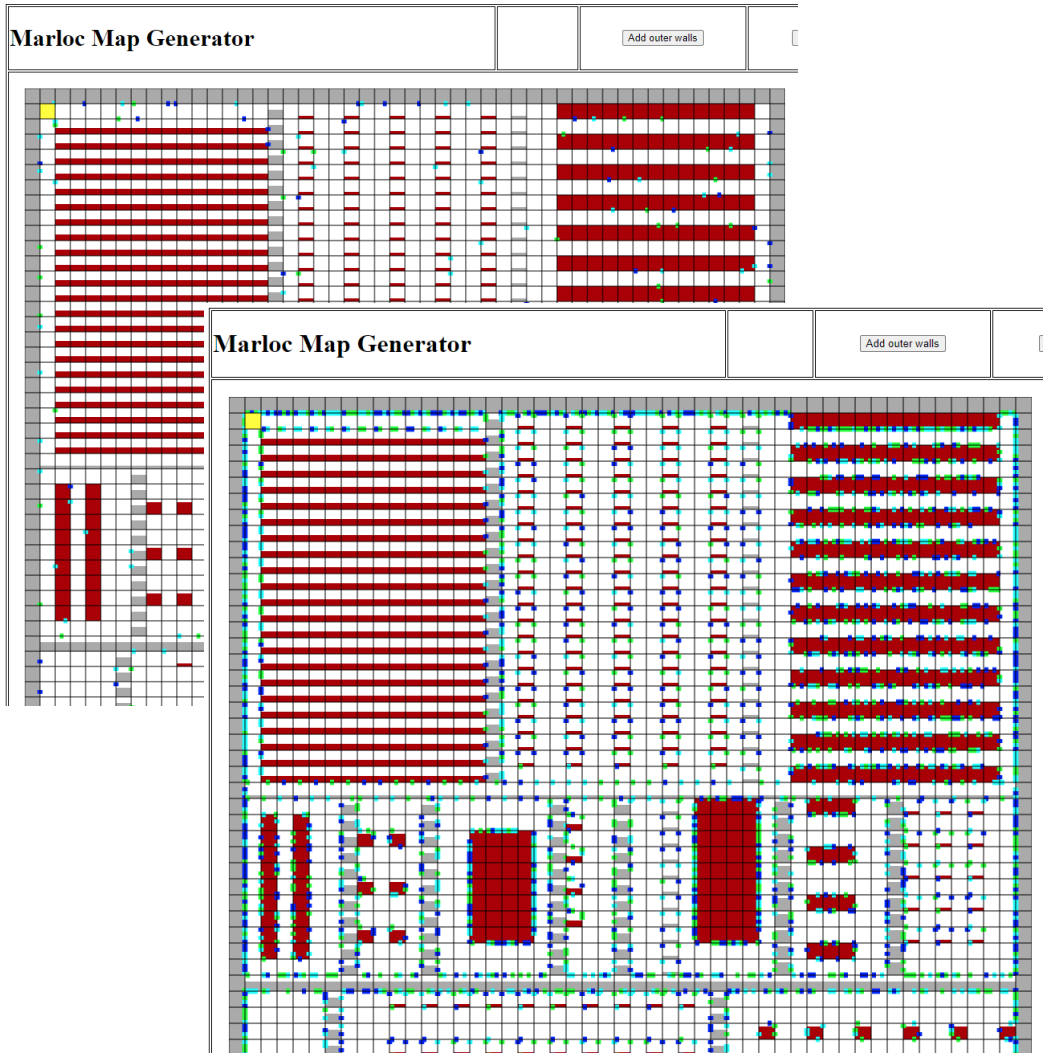
Ideal sensor/beacon placements

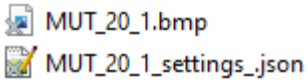
Generate Map
- Manual (HTML Tool)
- Matlab
- …

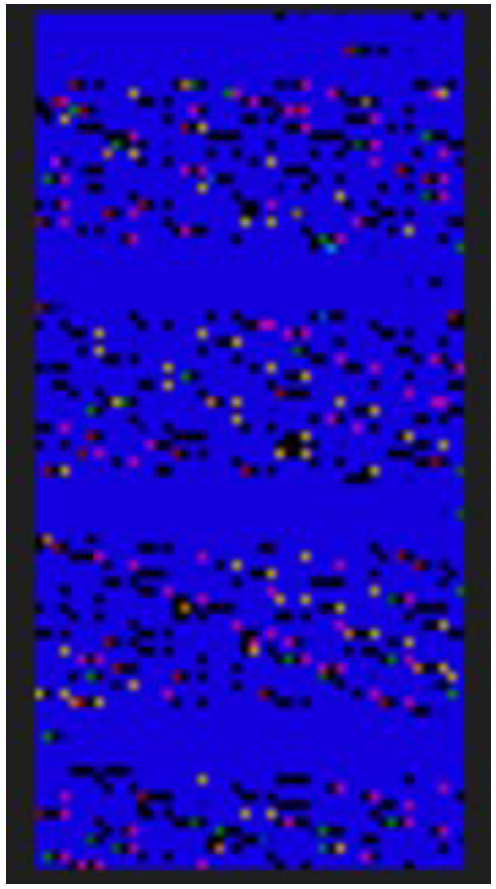
Adding Anchors (UWB, Wifi, Aruco)
- Matlab
- Varying fill levels

Output
- Image (NN Input)
- Settings (AirSim Input)

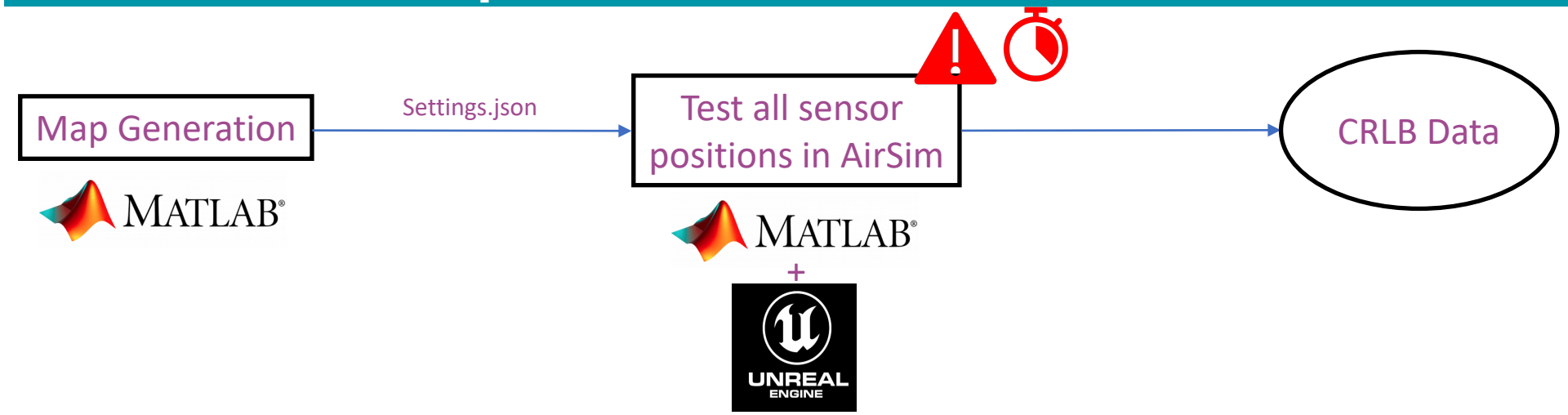






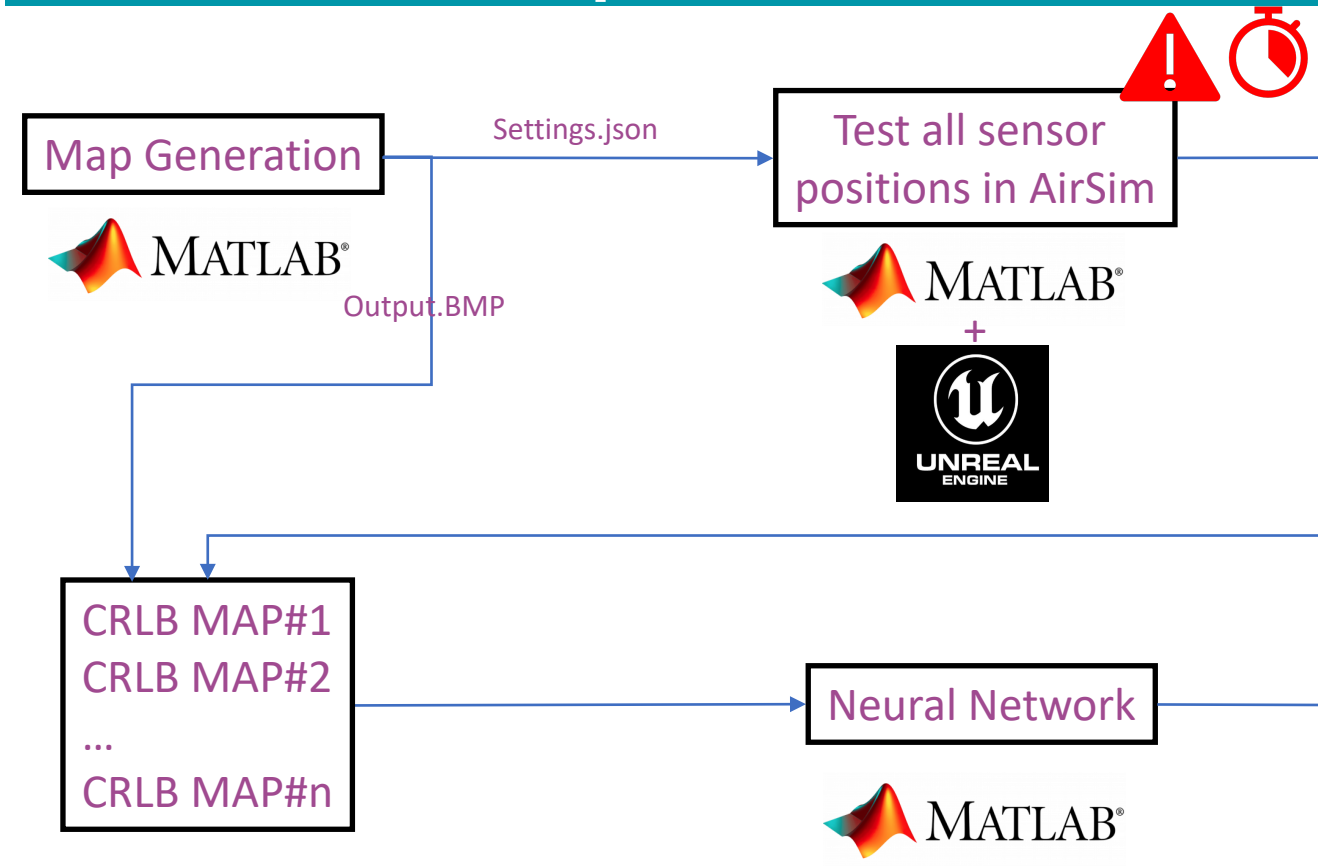


Ideal sensor/beacon placements



Step 1:
Takes a long time to calculate CRLB
5-30 minutes per map

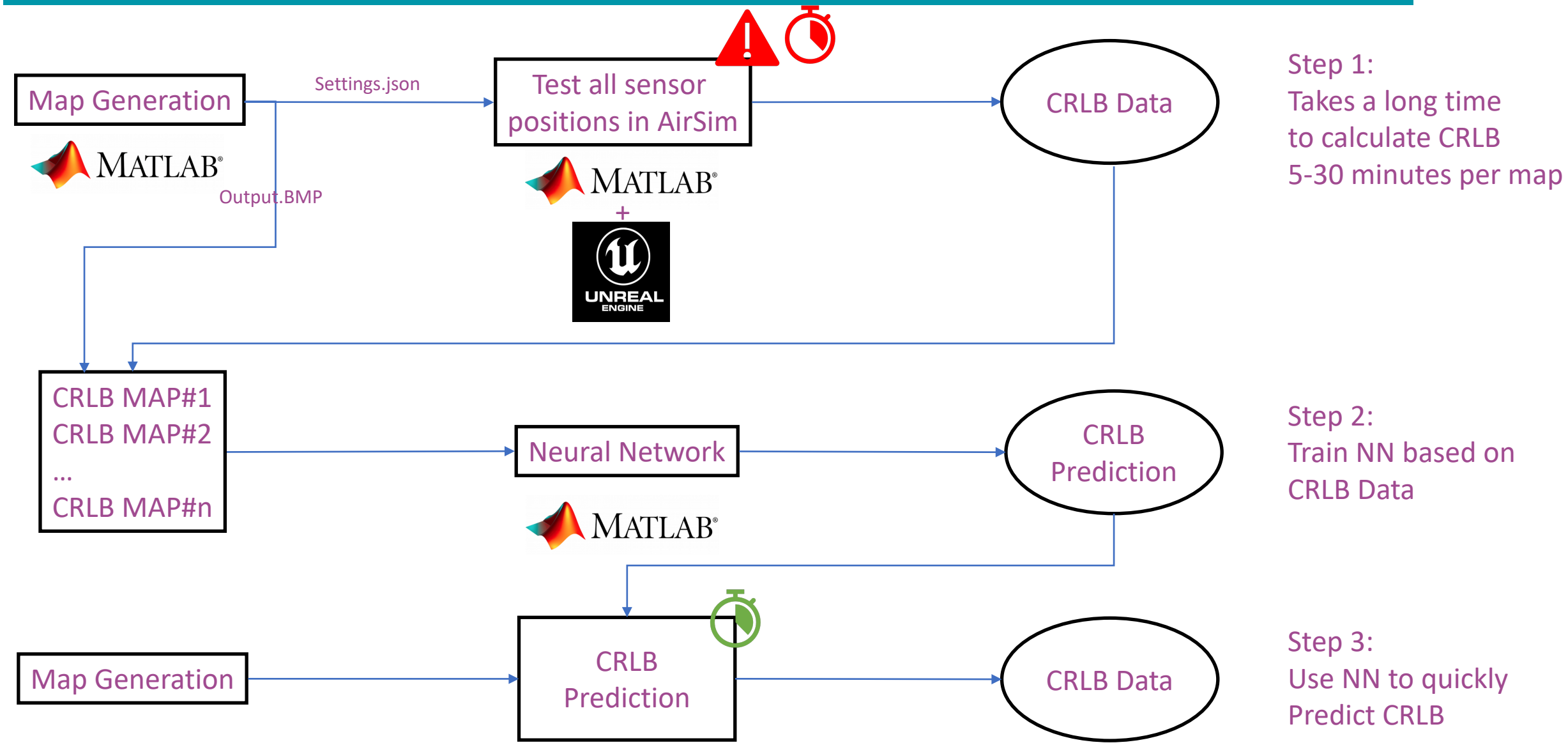
Ideal sensor/beacon placements



Step 1:
Takes a long time to calculate CRLB
5-30 minutes per map

Step 2:
Train NN based on CRLB Data

Ideal sensor/beacon placements



Ideal sensor/beacon placements: Demo

CLICK HERE

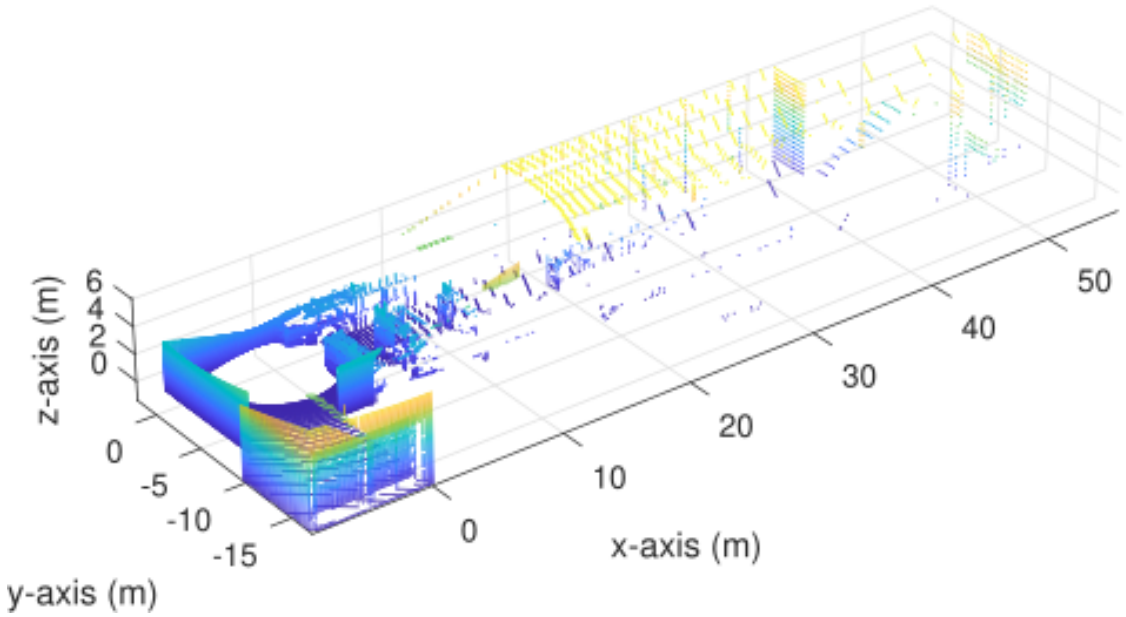
Application:

Lidar object segmentation

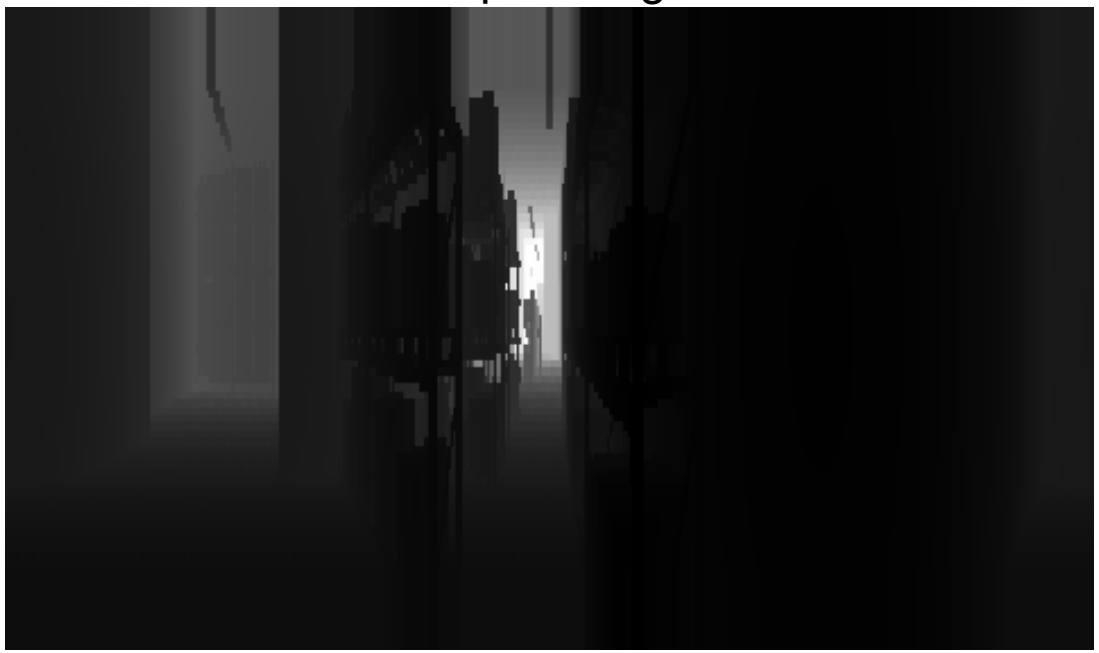
LiDAR Object Segmentation

- Goal: Segmentation of objects in indoor environments based on LIDAR point clouds
- Approach: Point Cloud -> Depth Image

Point Cloud



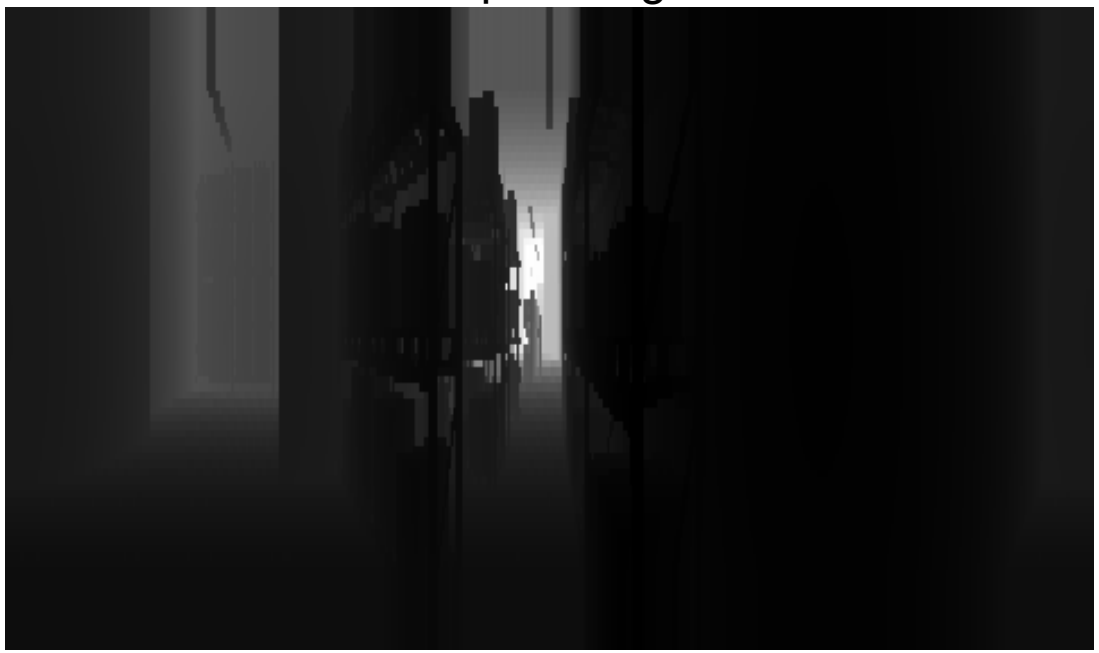
Depth Image



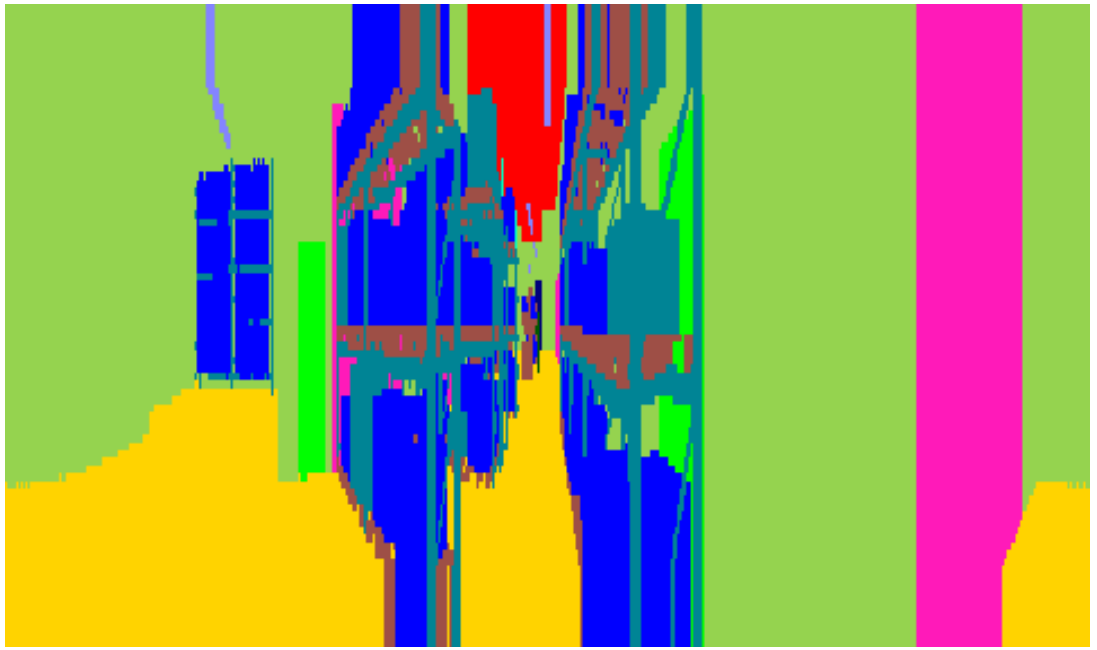
LiDAR Object Segmentation - Approach

Train CNN using depth images and corresponding label images

Depth Image



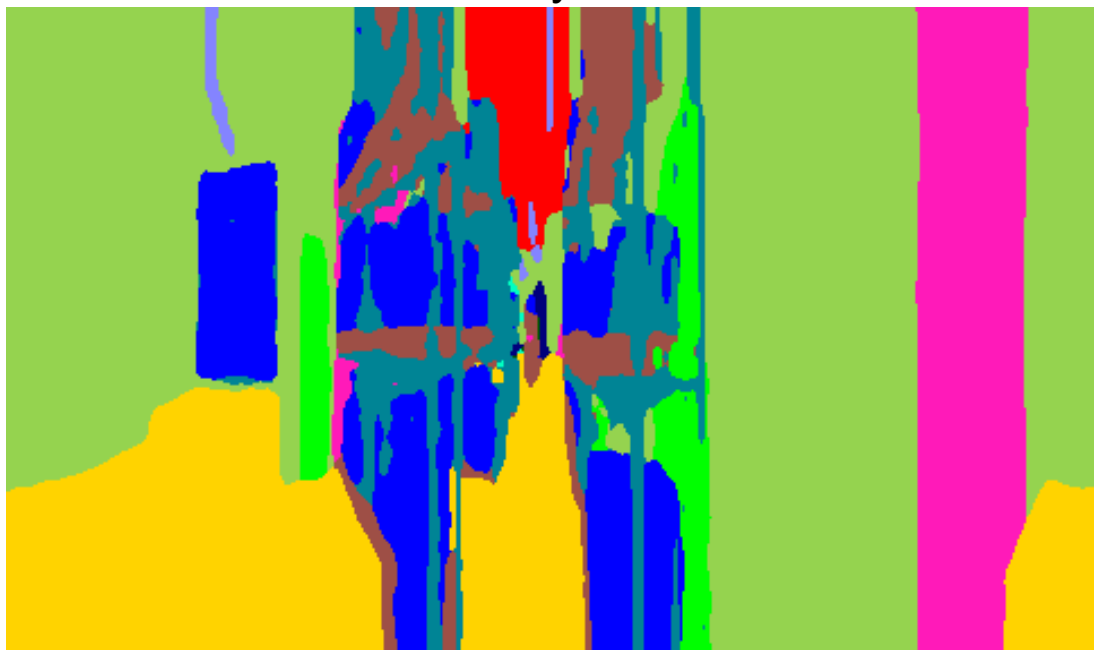
Labelled Depth Image

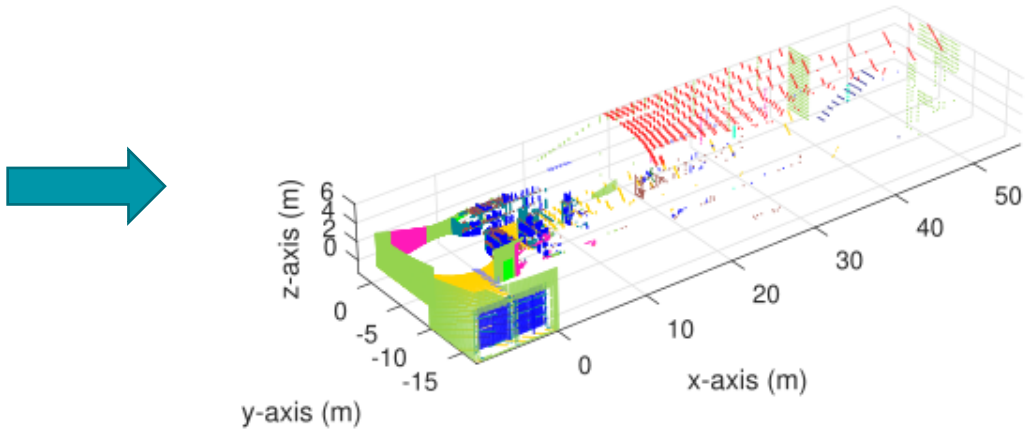


LiDAR Object Segmentation - Approach

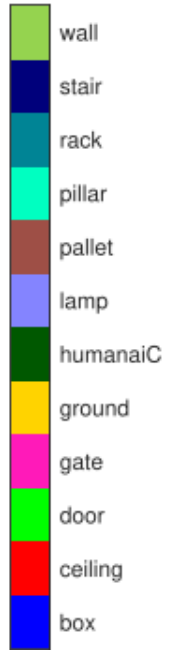
Label LiDAR data with the CNN and label the Point cloud

Labeled By CNN





Labeled Point Cloud

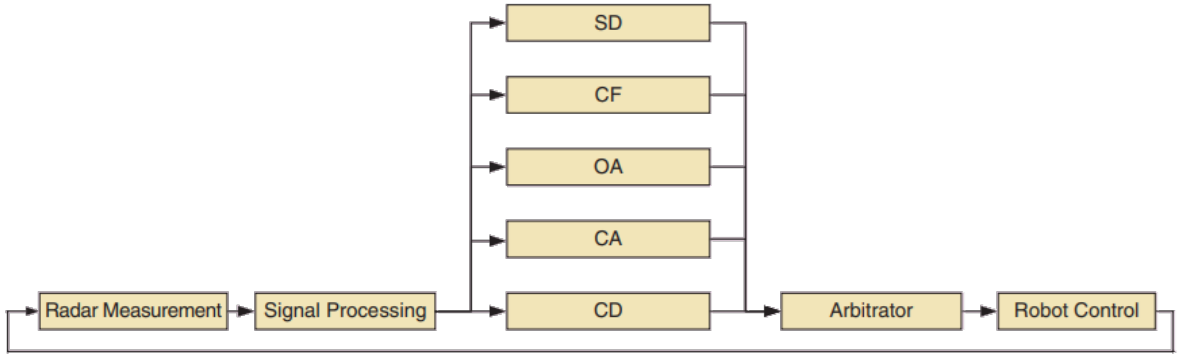


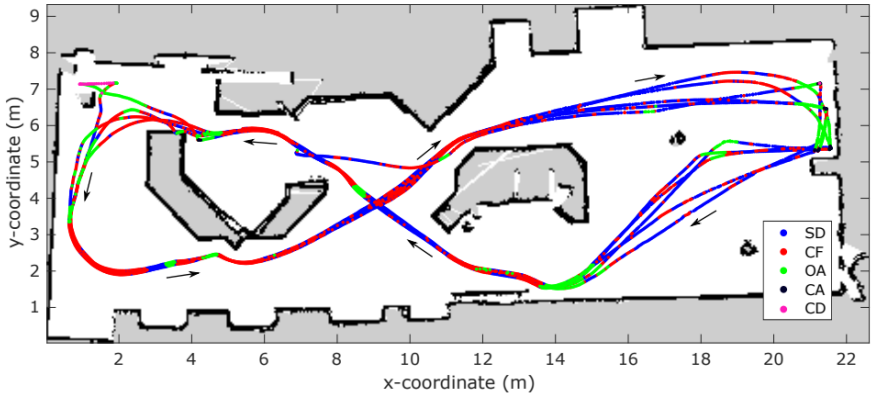
Application:

Pulse Echo Radar for navigation

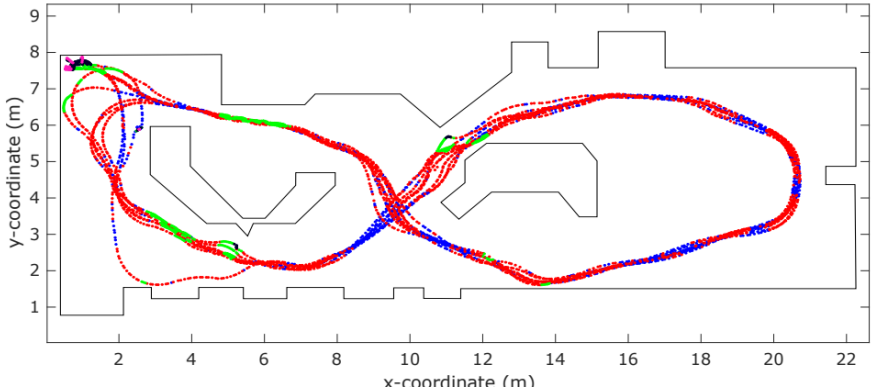
Pulse Echo Radar for navigation

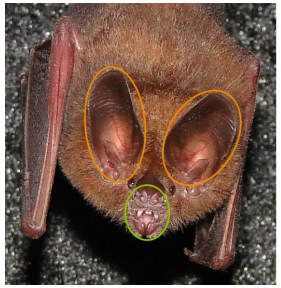
- Test echo sensor model
- Compare real recordings against simulation
- Autonomous navigation with behavior-based control
  - Collision detection
  - Obstacle avoidance
  - Corridor following
  - Straight drive

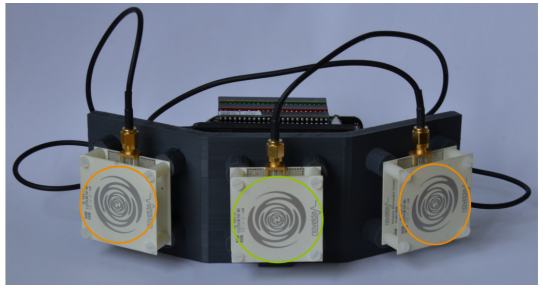




(a)







Pulse Echo Radar for navigation: Demo

CLICK HERE

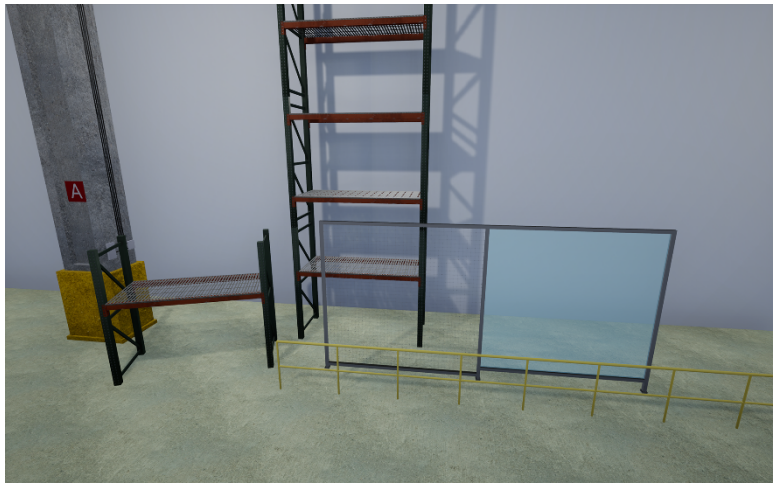
Application:

Camera segmentation

What do we want to detect ?

- Three categories of objects:
  - **Static objects: walls, ground, fences, racks, pillars…**
  - **Movable objects: pallets, boxes, barrels, couch, tables, doors…**
  - **Dynamic objects: persons, robots, forklift, pallet jack…**

Static objects



Movable objects

Dynamic objects

What is panoptic segmentation ?

- **Panoptic segmentation is a combination of instance and semantic segmentation**
  - Semantic segmentation: Each pixel is assigned a class label
  - Instance segmentation: Only cares about the segmentation of the instance into separate objects
  - Panoptic segmentation: both **amorphous regions** (ground, sky…) and **countable objects** (cars, persons…) are **assigned to a class** and **each countable object is uniquely segmented**

Semantic segmentation

Instance segmentation

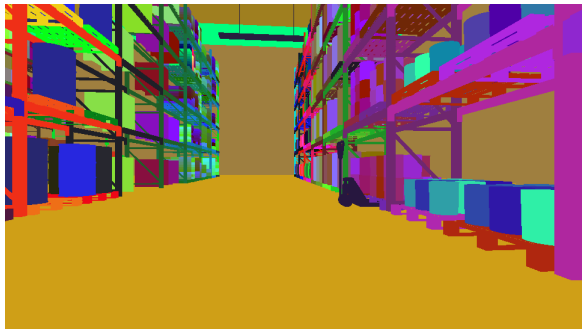
Panoptic segmentation

Data preparation and conversion

Unreal/Airsim generated Data

	A	B	C	D
1	**ObjectName**	**R**	**G**	**B**
2	doorframe_2	255	255	255
3	doorframe_3	255	255	127
4	rack_10	255	255	191
5	rack_15	255	127	191
6	gate_1	127	255	191
7	gate_2	127	127	191
8	wall_1	191	255	127
9	wall_10	191	127	255
10	wall_20	191	127	127

Excel file that map colors to label



Segmented image

COCO Panoptic segmentation format

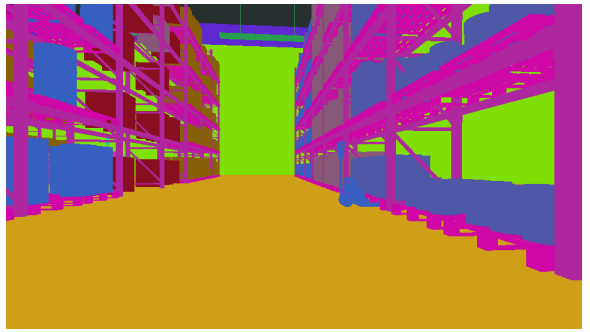
- Define Things and Stuff
- Merge the same instance from the Stuff categories

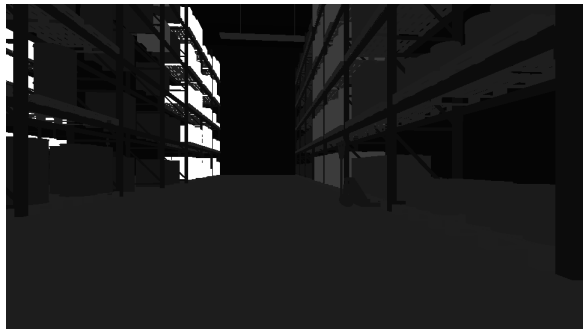
"categories": [{ "id": int,
"name": str,
"supercategory": str,
"isthing": int,
"color": list }]

"image": { "id": int,
"width": int,
"height": int,
"file_name: str}

"annotations": { "id": int,
"image_id: int",
"category_id": int ,
"segmentation": RLE or [polygon],
"area": float,
"bbox": [x,y,width,height],
"iscrowd": 0 or 1 }

Annotation Json file

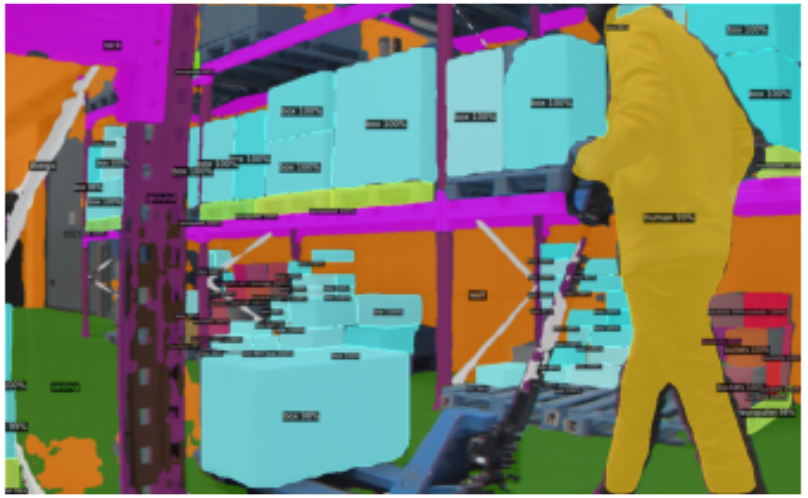




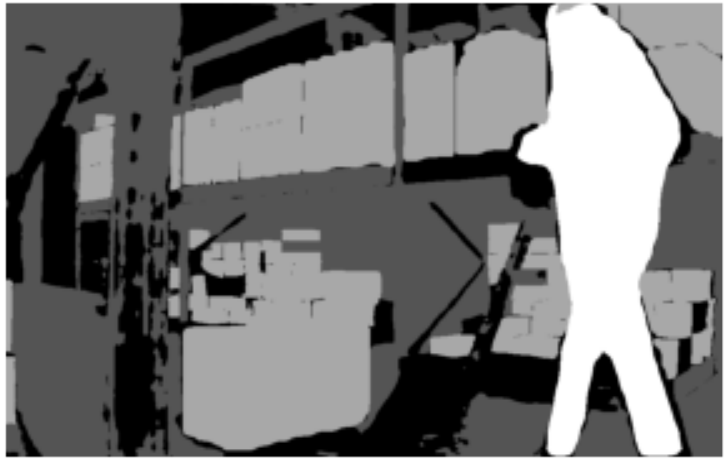
Segmentation mask

Algorithm steps

Input Image



Inference

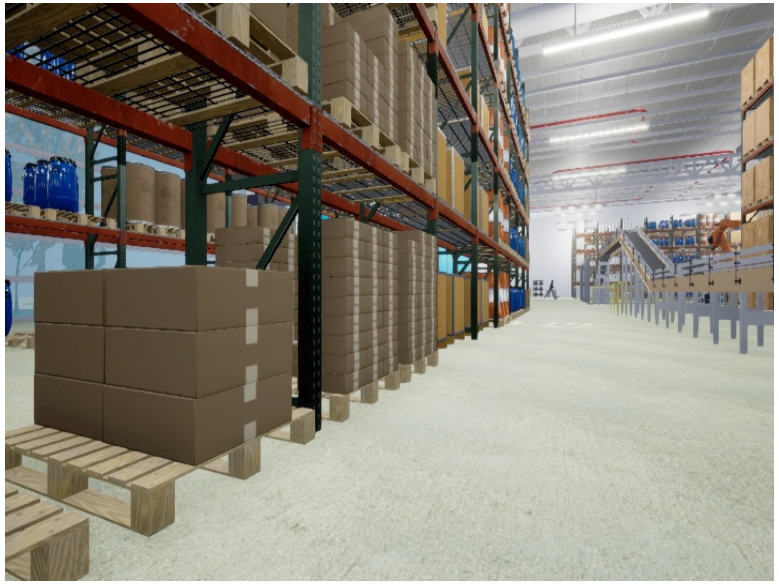


Class Masking

Bounding Boxes

Dataset Types

Large standard Warehouse



4 000 images

Synthetic reproduction of our warehouse

2 000 images

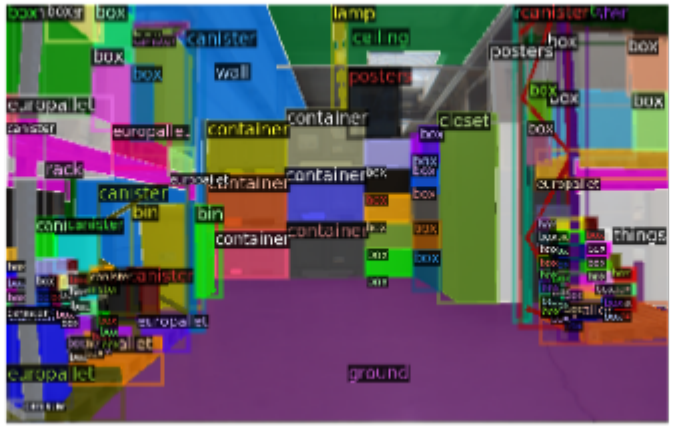
Real data from our real warehouse

500 images

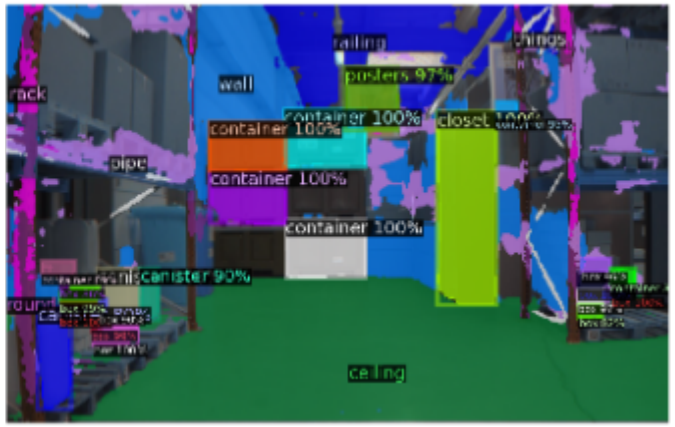
Results

RGB image

Ground truth image

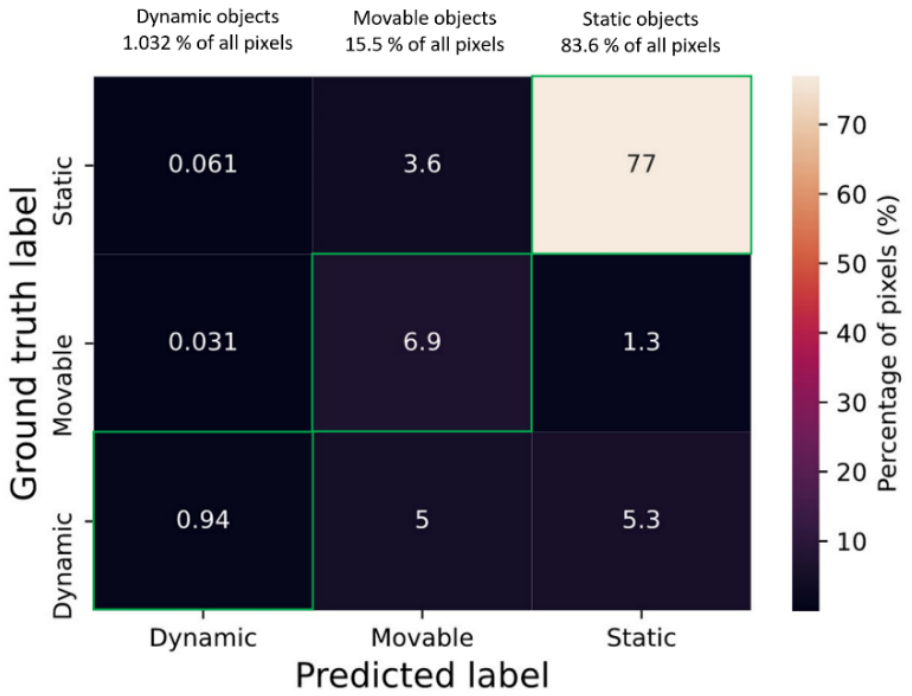


Panoptic image



Trained in simulation Infered on real data	Trained in simulation and on real data Infered on real data
PQ = 52	PQ = 148

Benchmark PQ of 60 for models trained on large dataset

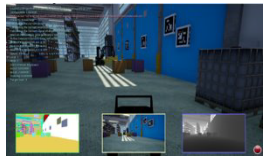


Application:

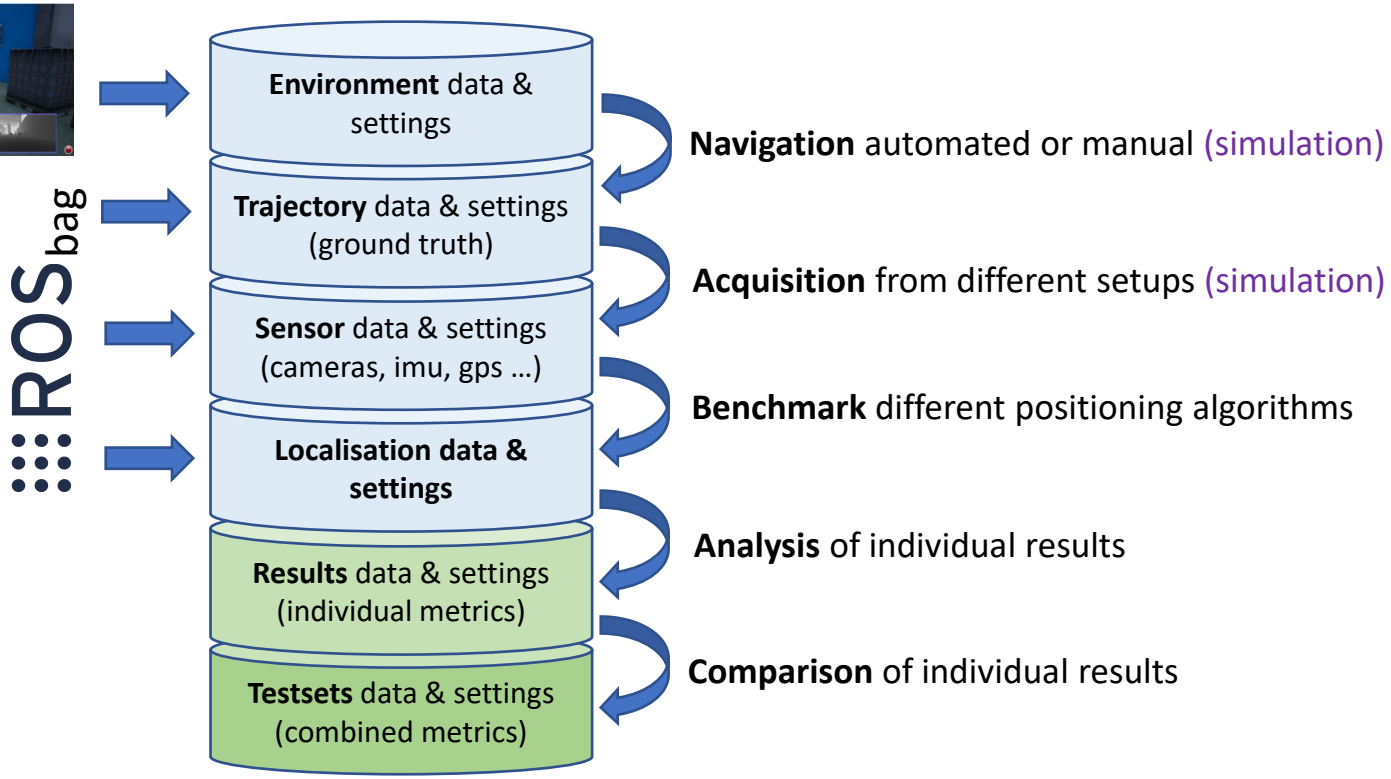
SLAM test toolbox

SLAMTest toolbox - uses

Airsim simulator



ROSbag



3 main flows

- Import localization data + generate metrics and reports
- Import sensor data + benchmark several SLAM + generate metrics and reports
- Simulate environment + benchmark several SLAM + generate metrics and reports

Robustify SLAM by understanding the environment

RGB image

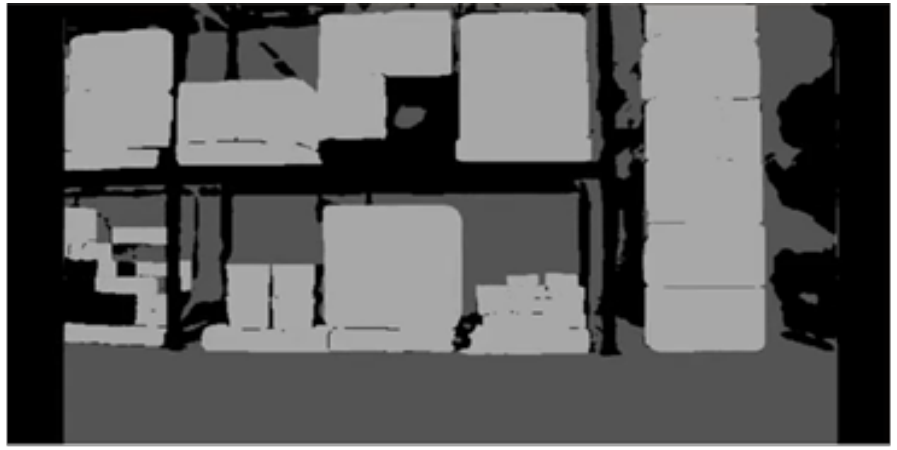
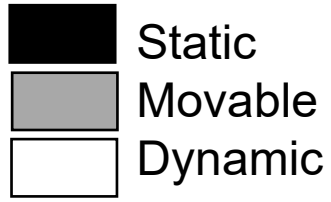
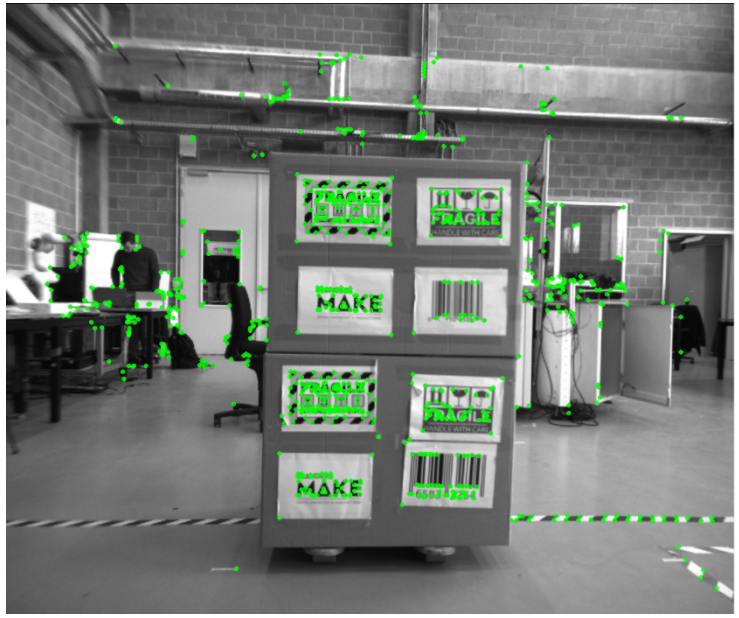


Image with mask classes

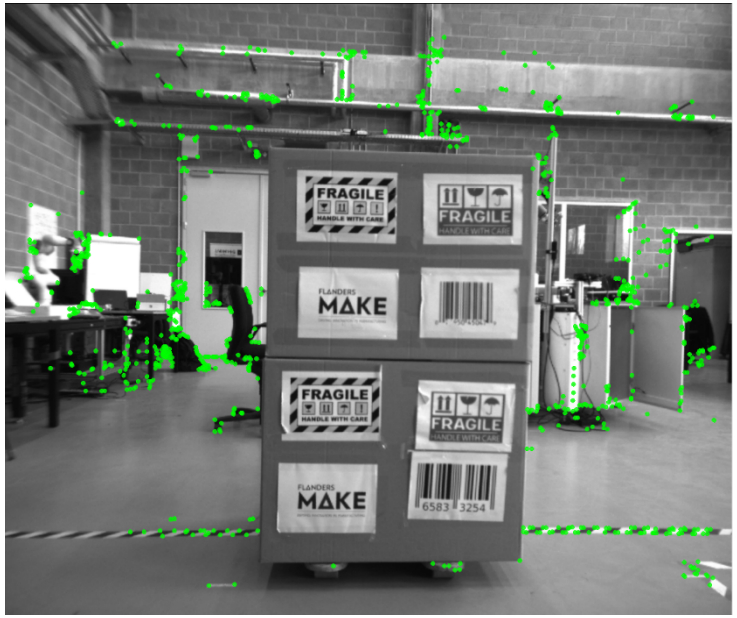


Influence of masking on SLAM



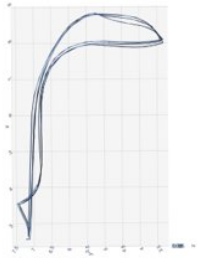
Standard SLAM Algorithm

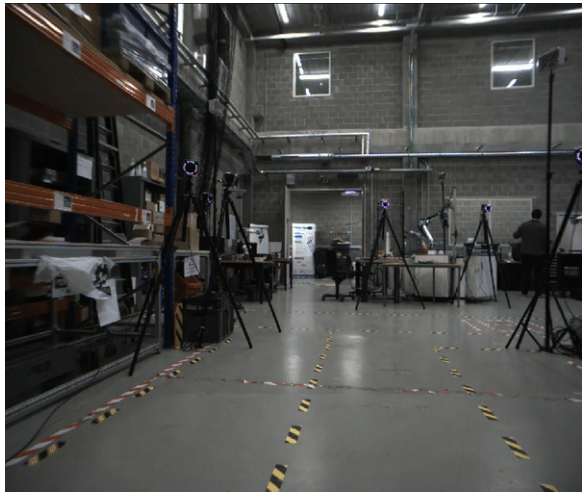
Masked Image



FM-SLAM Algorithm

Performance of SLAM algorithm in simulation compared to real life





Replay same trajectory in simulation

Real results

	Baseline	UWB prior
stereo odom	0.77	0.06
Lidar odom	0.03	0.03

Simulation results

	Baseline	UWB prior
stereo odom	0.26	0.16
Lidar odom	0.08	0.07

Future work

- Digital shadow/twin and extend HIL(T)
- Introduce fixed-step simulation mode
- Unreal 4.27/5 upgrade
- Outdoor environmental procedural generation
- Panoptic segmentation-based world generation
- More low-level models

Usage & Licensing

- Multiple open-source simulators have gone closed-source/commercial
  - Microsoft AirSim, LG LGSVL, ...
- Cosys-AirSim is open-source with MIT License
  - Matches original AirSim license
- Continue and build AirSim community
- More adoption from industry and academic research
- Interesting partnerships and co-development
- Contact us!
  - wouter.jansen@uantwerpen.be
  - jeanedouard.blanquart@flandersmake.be
  - connor.verhulst@flandersmake.be
  - cosyslab.uantwerpen.be
  - github.com/cosys-lab

Q & A