




D3.4 CPHS Extended Reality based tools for increasing situational awareness

<i>Authors</i>	UPAT, ISI, PASEU, ROBOTEC
<i>Work Package</i>	WP3 - Model based CP(H)S Layer Design and Development supporting Distributed Assisted, Augmented and Autonomous Intelligence
	<p style="text-align: center;">Abstract</p> <div style="border: 1px solid black; padding: 10px;"> <p>The overall aim of this deliverable is to demonstrate AR-based enhancement tools that will improve the human-in-the-loop situational awareness when being on-site or in remote locations. Situation awareness is a well-studied area in the field of ergonomics. A user (e.g., driver, operator) finds himself/ herself in a position where readiness and reflexes naturally decay deteriorating his response time to request to intervene messages. Boosting situation awareness facilitates the user to be in charge of the current situation, enables him/her to solve meaningful problems in a manner that is as natural as possible, improves decision-making and performance in a complex, dynamic environment, thus increasing productivity, reducing stress, improving user self-esteem while acting as safety countermeasure, using different augmented reality tools (e.g., AR Glasses, mobile devices and markerless tracking).</p> </div> <p style="text-align: center; font-size: small;">Funded by the Horizon 2020 Framework Programme of the European Union</p> <div style="text-align: center;">  </div>

D3.4 AR Interfaces Supporting Object and Scene Manipulation for Increasing Situational Awareness

Deliverable Information	
<i>Work Package</i>	WP3- Model based CP(H)S Layer Design and Development supporting Distributed Assisted, Augmented and Autonomous Intelligence
<i>Task</i>	T3.4. CPHS Extended Reality based tools for increasing situational awareness
<i>Deliverable title</i>	AR Interfaces Supporting Object and Scene Manipulation for Increasing Situational Awareness
<i>Dissemination Level</i>	PU
<i>Status</i>	Final
<i>Version Number</i>	0.3
<i>Due date</i>	M28
Project Information	
<i>Project start and duration</i>	01/01/2020 – 31/12/2022, 36 months
<i>Project Coordinator</i>	
	Industrial Systems Institute, ATHENA Research and Innovation Center 26504, Rio-Patras, Greece
<i>Partners</i>	<ol style="list-style-type: none"> 1. ATHINA-EREVNITIKO KENTRO KAINOTOMIAS STIS TECHNOLOGIES TIS PLIROFORIAS, TON EPIKOINONION KAI TIS GNOSIS (ISI) the Coordinator 2. FUNDACIO PRIVADA I2CAT, INTERNET I INNOVACIO DIGITAL A CATALUNYA (I2CAT) 3. IBM ISRAEL - SCIENCE AND TECHNOLOGY LTD (IBM ISRAEL) 4. ATOS SPAIN SA (ATOS) 5. PANASONIC AUTOMOTIVE SYSTEMS EUROPE GMBH (PASEU) 6. EIGHT BELLS LTD (8BELLS) 7. UNIVERSITA DELLA SVIZZERA ITALIANA (USI), 8. TAMPEREEN KORKEAKOULUSAATIO SR (TAU) 9. UNIVERSITY OF PELOPONNESE (UoP) 10. CATALINK LIMITED (CATALINK) 11. ROBOTEC.AI SPOLKA Z OGRANICZONA ODPOWIEDZIALNOSCIA (RTC) 12. CENTRO RICERCHE FIAT SCPA (CRF) 13. PANEPISTIMIO PATRON (UPAT)
<i>Website</i>	www.cpsosaware.eu

Control Sheet

VERSION	DATE	SUMMARY OF CHANGES	AUTHOR
0.1	15/03	Initial draft	UPAT
0.2	01/04	Pre-final draft	ISI, PASEU, ROBOTEC, UPAT
0.3	10/04	Finalization	UPAT

	NAME
Prepared by	UPAT
Reviewed by	RTC, 8BELLS
Authorized by	ISI

DATE	RECIPIENT
28/04	Coordinator
30/04	European Commission

Table of Contents

Executive Summary.....	6
1 Introduction.....	7
1.1 Connection with other Tasks.....	7
1.2 Structure of the document.....	7
1.3 Links with code and videos.....	8
2 State-of-the-art in situation awareness.....	9
3 AR-based visualization and interaction techniques.....	11
4 Situational awareness in automotive.....	13
4.1 Visualization of potholes and other obstacles.....	13
4.1.1 Saliency map estimation of the point cloud scene.....	14
4.1.2 Visualization results.....	16
4.1.3 Data Simulations.....	20
4.1.4 Interfaces and communication.....	20
4.1.5 AR Visualization.....	21
4.1.6 Vehicle Communication.....	21
4.1.7 Available code repositories.....	23
4.2 Visualization of occluded vehicles VR/AR UPAT simulator.....	25
4.2.1 Configuration Scene / Path Finding / Basic Assumptions.....	26
4.2.2 Procedure of Path Finding/Creating.....	29
4.3 Warnings related to driver’s drowsiness and other security aspects.....	38
5 Situational awareness in manufacturing.....	41
5.1 Estimation of the working envelope of a robotic arm and visualization of the octree based robot risk mapping.....	42
5.2 Warnings for the reconfiguration of the windshield based on pose recognition.....	49
5.3 Warnings based on operator’s proximity with the robot and visualization of safety zones.....	51
5.3.1 Robot modeling.....	52
5.3.2 Collision Risk Prediction.....	53
6 Conclusions.....	65
7 References.....	66

List of Figures

Figure 1: Situation Awareness Model [6], [7].	9
Figure 2: Different theoretical situational awareness models that address problems with different situational awareness levels [10].	10
Figure 3: Schematic diagram of the proposed methodology.	13
Figure 4: Image from the camera of the vehicle.	17
Figure 5: Example of saliency map extracted from the road scene shown in the previous Figure.	17
Figure 6: Example of segmentation.	17
Figure 8: Example of segmentation of the point cloud projected to the AR interface (ego 1).	18
Figure 7: Perspective projection of the points to the AR interface and image filling (ego 1).	18
Figure 9: Pothole recognition and visualization (ego 1).	18
Figure 10: Pothole recognition and visualization (ego 2).	19
Figure 11: Pothole presentation before be recognized by the ego 2.	19
Figure 12: Starting point of ego 2.	19
Figure 13: Point cloud map of both two vehicles (ego 1 and ego 2)	20
Figure 14: Flowchart of communicative vehicles for obstacle sharing.	22
Figure 15: (a) Proper Turn, (b) Roundabout (3 turns), (c) Small Turn	26
Figure 16: (a) Proper Turn, (b) Short Turn	27
Figure 17: Proper Turn	29
Figure 18: Short Turn	29
Figure 19: Path Finding Short Turn	30
Figure 20: Engine Methods	32
Figure 21: Question New Node	32
Figure 22: VR simulator and set up implementation.	33
Figure 23: Visualization using 3D wedge.	33
Figure 24: Visualization using 3D arrows.	34
Figure 25: Visualization using 3D minimaps.	35
Figure 26: Visualization using 3D Radar.	35
Figure 27: Visualization of the occluded silhouette via a 3D mesh overlay.	36
Figure 28: Visualization of the object via a representative 3D sphere.	36
Figure 29: Example of the AR implementation.	37
Figure 30: Integration of DMS with CARLA environment using wheel chair	39
Figure 31: Visualization of EAR, PERCLOS, capturing time, occupancy factor and driver's eye contour	40
Figure 32: Visualization of the eyes shape during the driving	40
Figure 33: Drowsiness detection alert	40
Figure 34: Collection of points while the robotic arm performs a pre-defined and planed work.	42
Figure 35: Example of a point cloud that defines the working envelope of a robotic arm.	43
Figure 36: The same in a view on top (z axis) defining the xy covering space.	43
Figure 37: 3D volumetric representation of the working envelope.	44
Figure 38: The same with above, in the field of view of an operator.	44
Figure 39: Pose estimation and class estimation for three different operators with different heights.	50
Figure 40: Schematic diagram of the AR-based system for safety-aware human-robot collaboration.	52
Figure 41: Dimensions of Kuka KR 150R 2700	52
Figure 42: Current robot pose	54
Figure 43:(a) 8cm voxel size, (b) 4cm voxel size, (c) 1cm voxel size.	54
Figure 44: State-aware collision risk factor point sets.	57
Figure 45: Dynamic safety zone construction with cube representation, (a) Perspective View, (b) On top View.	58
Figure 46: The virtual environment.	59

Figure 47: The helper generated cubes 59
 Figure 48: Static safety zones - Cube Representation 61
 Figure 49: Static safety zones - Fog representation 62
 Figure 50: Dynamic safety zones - Fog Representation. 62
 Figure 51: Dynamic safety zone construction with cube representation using Risk Assessment Occupancy Mapping. (a) Side View, (b) On top View. 63

List of Tables

Table 1: List of classes, struct and interfaces..... 47
 Table 2 List of documented files. 49

List of abbreviations/acronyms

Abbreviation	Definition
PERCLOS	PERcentage of eyelid CLOSure
CAD	Computer-Aided Design
JSON	JavaScript Object Notation
SNR	Signal-to-Noise Ratio
DoF	Degree of Freedom
EAR	Eye Aspect Ratio
FOV	Field of view
HMD	Head-Mounted Display
HRC	Human Robot Collaboration
CAV	Connected-Automated Vehicles
VR	Virtual Reality
AR	Augmented Reality
XR	Extended Reality
MR	Mixed Reality
cobots	Collaborative Robots
HMI	Human Machine Interfaces
GPS	Global Positioning System
LiDAR	Light Detection And Ranging
DMS	Driver Monitoring System
RGB	Red Green Blue
UI	User Interfaces

Executive Summary

This document is the final output of the activities performed in Task 3.4. The goal of this deliverable is the demonstration of visualization tools developed in the framework of T3.4 “CPHS Extended Reality based tools for increasing situational awareness” that aim to improve the situational awareness of the users. In our case studies, a user could be (i) a driver in a cooperative driving scenario for semiautonomous and connected vehicles or (ii) an operator working in collaboration with a robot in human-robot collaborative industrial environments.

Throughout the visualization tools, the users would be able to a) receive information streams regarding the task underway improving focus, b) receive notifications and visual aids regarding imminent dangers or accident-related factors.

Regarding the automotive pillar, the implementations for the situational awareness of the drivers are related to (i) the visualization of potholes and other obstacles in the range of the road, (ii) the visualization of occluded vehicles in the VR/AR UPAT simulator, (iii) the visualization of warnings related to driver’s drowsiness and other security aspects. On the other hand, regarding the manufacturing pillar, the implementations for the situational awareness of the operator are related to (i) the estimation of the working envelope of a robotic arm and the visualization of the octree based robot risk mapping, (ii) the estimation of warnings for the reconfiguration of the windshield based on the operator's pose recognition and (iii) warnings based on the operator’s proximity with the robot and visualization of safety zones.

1 Introduction

The interaction between the driver and the vehicle is the subject of many innovations regarding the drivers' activity, comfort and road safety. Autonomous vehicles provide the advantage of relieving the main user of the driving task in order to enhance other activities such as rest, work or leisure. The management of the vehicle during the journey, relieved of driving, will be very different and will have to take into account these new activities in which users will engage their attention. Public interest and fears about autonomous vehicles are slowly improving. But the societal rupture at stake requires us to study the levers to ensure the best possible acceptability. This task focuses on increasing the situational awareness of the driver, exploring new ways of interacting, staying informed and managing ambient comfort and dynamics.

Recent advancements in human-robot collaboration have enabled humans and robots to work together in a shared manufacturing environment [1]. Since their first use at industrial level, in 2008, Collaborative robots or Cobots have been used in a significant number of tasks combining the perception of a human with the efficiency of a robotic device. In many cases, repetitive and detailed tasks, which may also incorporate heavy payload, are being handled by robotic entities, such as in the automotive industry, where in a manufacturing plant light robotic arms assemble motors and pumps at high speed. In the provided working environment humans are able to be occupied with more abstract tasks which require complex thinking and a variety of interactions. Such working paradigms introduce the need for a context-aware safety system that prioritizes human protection. There are two main ways to make any collaborative task safer. The first is to program the robot to be able to adjust its movement according to the movement of the human user. Knowing the workspace of both humans and the robot is fundamental for successfully controlling the robot for that specific reason. Some works [2] make use of both workspaces to be able to find a set of ideal poses of a robot in order to help a human user with a specific task. Another way of creating a safer collaborative environment for both a human and a robot can be the accurate sensing of collisions between the human and the robot based on image feedback [3] or force/torque sensors [4]. However, in many industries, the change of the planning of a robot can be a difficult task due to the lack of time and often appropriate personnel. This leads to the second way of making a collaborative task safer, which is a proper training of the personnel and accurate information about the movement of the robot.

1.1 Connection with other Tasks

Works in Task 3.4 are aligned with work in WP3 "Model based CP(H)S Layer Design and Development supporting Distributed Assisted, Augmented and Autonomous Intelligence".

In particular, the methods for monitoring (facial and pose landmarks estimation) effectively important aspects of human activity and behaviour (T3.1), are seamlessly integrated into the CPSoSaware simulators in a variety of configurations and they are used for the visualization of useful or critical information. All these models are then collected and centrally managed by the Orchestration tools that support autonomic functionality, as delivered in D2.2 "CPSoS orchestration optimization tools" as a result of the work performed in T2.5. The simulators that have been developed are also used in T5.3 for the long life learning of users (manufacturing pillar) and for the familiarization of the drivers with the real-life implementation of the application (automotive pillar). Part of this analysis has been also presented in D6.2.

1.2 Structure of the document

- In Section 1, we make a short introduction about the main topic of this deliverable, setting the motivation and

the objectives of the deliverable, we present the connection of this deliverable with other tasks and deliverables, and the structure of the document.

- Section 2 presents the state-of-the-art related to situation awareness.
- Section 3 makes a short discussion about AR-based visualization and interaction techniques.
- In Section 4, we present the implementations that we made related to the automotive pillar for the situational awareness of the drivers. More specifically, we present and discuss three cases (i) the visualization of potholes and other obstacles in the range of the road, (ii) the visualization of occluded vehicles in the VR/AR UPAT simulator, (iii) the visualization of warnings related to driver's drowsiness and other security aspects.
- In Section 5, we present the corresponding implementations regarding the manufacturing pillar for the situational awareness of the operator. More specifically, we present three cases (i) the estimation of the working envelope of a robotic arm and the visualization of the octree based robot risk mapping, (ii) the estimation of warnings for the reconfiguration of the windshield based on the operator's pose recognition, and (iii) warnings based on the operator's proximity with the robot and visualization of safety zones.
- And finally, in Section 6, we draw the conclusions.

1.3 Links with code and videos

Below, we provide the code and video links for the discussed demos related to automotive pillar (Section 5):

1. Code link:

- https://github.com/Stagakis/ipc_shared_memory_demo
- <https://github.com/Stagakis/carla-data-generation>
- <https://github.com/Stagakis/roadpatch-with-pothole-generator>
- <https://github.com/Stagakis/saliency-from-pointcloud>
- <https://github.com/Stagakis/carlapclprocessing>
- <https://gitlab.com/vvr/derinosim/-/tree/master>

2. Video link:

- <https://drive.google.com/drive/folders/1mMIPYFkTFPGJ8nFELHL0Dqqv3XEHOjLd?usp=sharing>

Below, we provide the code and video links for the discussed demos related to manufacturing pillar (Section 6):

1. Code link:

- https://gitlab.com/vvr/octree_occupancy

2. Video link:

- <https://drive.google.com/drive/folders/18kOAGzThVC8BO7Z9CAwxWhfboUP05Xfa?usp=sharing>

2 State-of-the-art in situation awareness

One of the earliest and most widely used definitions of situational awareness describes it as the “perception of the elements in the environment within a volume of time and space, the comprehension of their meaning and the projection of their status in the near future” [5]. Based on this definition, situational awareness is comprised of three levels: **(1) perception, (2) comprehension, and (3) projection** (as shown in Figure 1). Perception (level 1 of situational awareness) involves the sensory detection of significant environmental cues. For example, operators need to be able to see relevant displays or hear an alarm sound. In the field, the use of other senses to gather information (e.g., the smell of burning wire) may also be pertinent.

Operators with a good feeling of comprehension (level 2 situational awareness) are able to understand the immediate impact of an outage on other parts of the system, or that a particular voltage value is “over the limit.” Projection, the highest level of situational awareness, consists of extrapolating information forward in time to determine how it will affect future states of the operating environment. This merges what the individual knows about the current situation with their mental models of the system to predict what is likely to happen next – for example, being able to project the impact on the system of removing an element from service. The higher levels of situational awareness allow operators to function in a timely and effective manner, even with very complex and challenging tasks.

The performance of the action is mainly determined by the choices made by the user. And the operation of these choices is mainly based on the 3 aforementioned levels of situational awareness:

- Perception of the elements of the environment (level 1)
- Understanding the meaning of the situation (level 2)
- Projection of future state and events (level 3)

Situational awareness is the permanent and necessary adaptation of the mental representation of the situation, and the understanding of the environment, as well as the anticipation of changing situations. This construction based on the continuous knowledge of the driver available in long-term memory. The main element of situational awareness, summarized in the Endsley model [6], [7] is presented in Figure 1. The model presents 2 sets of determinants that impact the situation awareness, the decision and the performance of action.

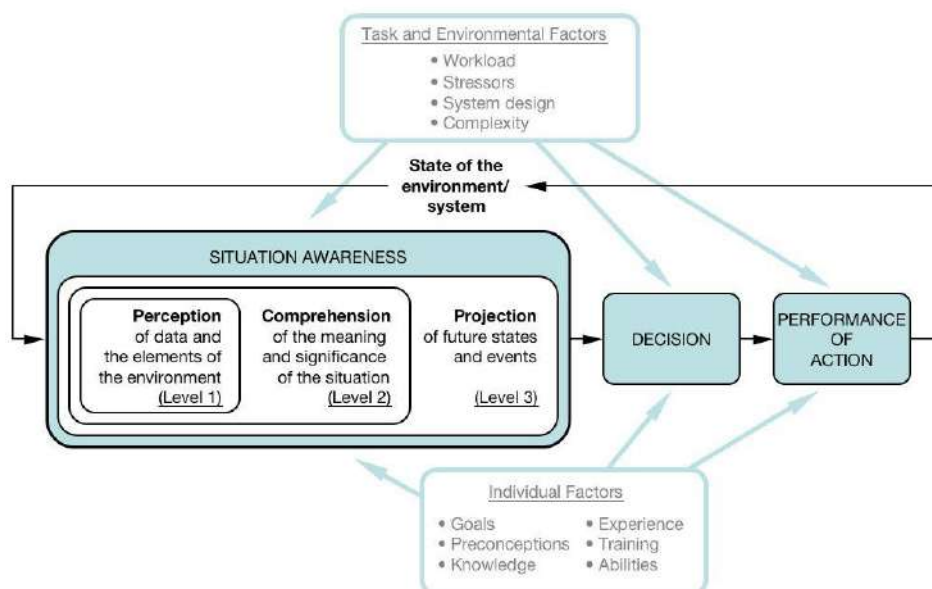


Figure 1: Situation Awareness Model [6], [7].

Each level can be impacted by several factors. Endsley describes 2 sets of factors that can impact each level, but also on decision and action performed:

- Task and environment factors (e.g., workload, stressor, system design, complexity)
- Individual factors (e.g., goals, personalizations, knowledge, experience, training, abilities)

The management of control between autonomous vehicles and drivers becomes important both from a cognitive and situational awareness point of view. The transitions between different levels of autonomous driving have to take into account all the variables linked to the dynamics of cognitive control. Expertise, routines, and also the environment play an important role in affecting the user action in reactive or anticipatory terms because it will impact the level of situational awareness and particularly the third level. For instance, when drivers begin to make the experience of the autonomous driving, they also start to raise up their cognitive control level, detaching themselves from the manual actions linked to the driving situation and leaving space (cognitive space) for more complex cognitive processes, reaching the highest levels of situation awareness. In an autonomous driving process, it is necessary to take into account the situational awareness needs in terms of expertise, time availability and specific Human-Machine Interfaces (HMI) oriented to all the three levels of situational awareness. In this sense, a predictive HMI, designed on specific information (e.g., situational cues, scenarios dynamics forecasts) could help to handle the control transitions.

Individual situational awareness. The individual situational awareness mainly focuses on how individual operators acquire situational awareness cognitively during the execution of a specific task. It is difficult to explain the formative aspects of situational awareness, and to prove the argument that knowledge in the head (or expectancy) can enable actors to create a rich awareness of their situation from very limited external stimuli.

Team and shared situational awareness. On the other hand, complex environments often involve multiple stakeholders; therefore, individual situational awareness may no longer be sufficient. Although the ability to evaluate situational awareness of individual team members plays a significant role in assessing team situational awareness, it is more important to measure the overall or shared team situational awareness during task execution. The focus, therefore, is shifted from individual situational awareness to team or system-level situational awareness.

Figure 2 illustrates different theoretical situational awareness models corresponding to different situational awareness levels [8], where no ubiquitously superior model could be claimed. Each model suits a particular problem depending on its fundamental nature [9]. A flexible situational awareness method is needed to deliver the required insights with moderate analytical effort.

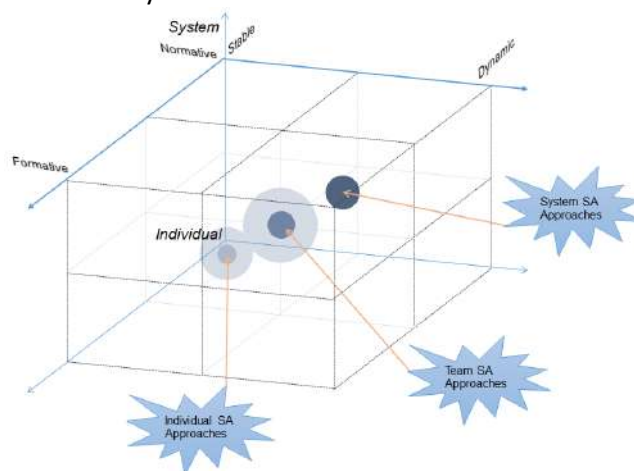


Figure 2: Different theoretical situational awareness models that address problems with different situational awareness levels [10].

3 AR-based visualization and interaction techniques

Information visualization is a research area that focuses on the design and development of new presentation approaches, visual layouts, visual interaction methods, data manipulation and transformation, and insight generation for information search, information exploration, and knowledge acquisition, for the purpose of performing various heterogeneous analysis tasks [11]. Visual Analytics is the study of knowledge generation based on interactive visual reasoning. It combines data analysis with interactive visualizations for an understanding and decision making on the basis of large and complex data [11]. Adaptive Visualization is an interactive, autonomously-evolving, learning, and constantly-improving visualization method and display of variables and conceptual structures, based on user-behavior, data characteristics, and other factors. The purpose of this approach is to amplify cognition and enable efficient information acquisition by the users [11]. Information visualization is strongly linked to human cognition-augmentation [12]. It has the potential to leverage human visual perception capabilities for influencing high-level cognitive processes such as retrieval from long-term memory, reasoning, learning, and understanding.

The purpose of an AR application for situational awareness is to:

- Facilitate the user to be in charge of a current situation
- Enable the user to solve meaningful problems in a manner that is as natural as possible
- Improve decision-making and performance in complex or dynamic environments
- Increase user's productivity
- Reduce stress
- Improve user's self-esteem
- Act as a safety countermeasure
- Reduce the possibility of accidents
- Provide better communication between users
- Extend the productive life of the CPSoS
- Make the training process easier

Using different augmented reality tools (e.g., AR Glasses, mobile devices and markerless tracking), the user can:

- a) Receive **information streams** regarding the task underway improving focus.
- b) Receive **personalized reminders** regarding other **parallel or scheduled tasks** significantly improving response time.
- c) Receive **notifications and visual aids** regarding **imminent dangers or accident-related factors**.

The type of information could be related to:

- Eye gaze
- Head direction
- Hand gestures
- Physiological values of user (temperature, heart-rate, inhalation, etc.)
- Environmental values
- Captured scene (e.g., point cloud) for an objects-of-interest analysis
- Localization (GPS)
- Personalized information related to each user separately
- Collaborative information (e.g., from other drivers or colleagues)

Types of situational awareness output

- Time series of real-time data (i.e., from sensors)

D3.4 AR Interfaces Supporting Object and Scene Manipulation for Increasing Situational Awareness

- Pop up messages (e.g., the output of decision-making algorithms)
- Real time visualization (i.e., transparent plans, trajectories)
- KPIs
- Augmented static or dynamic 3D objects and simulation
- Sounds
- On demand information (“How to...”)

Importance of awareness cues

- Alerts
- Warnings
- Notifications
- Information

Benefits of using AR for situational awareness

- Can increase collaboration and teamwork among fellow workers
- Helps the users to better understand abstract topics
- Involves gamification of learning which makes the process fun and interactive
- Allows the user to be in a distraction-free environment which helps them to better understand the concepts
- Users can experience things happening around them since these technologies come with intelligent learning content
- Provides understandable and real-time responses
- Provides visual cues without distracting the users from their physical environment

4 Situational awareness in automotive

It is estimated that in the near future the Connected-Automated Vehicles (CAV) will dominate on the roads. Indeed, major players in the automotive industry have already invested largely in designing vehicles with Level 4/5 automation and many started pilot-testing these vehicles in designated areas. Hence a lot of resources and manpower are being allocated to realizing the goal of fully-automated vehicles. Yet, the success of these efforts depends on whether the public would accept connected and automated vehicles and whether they would adopt these vehicles. Acceptance is an important barrier to the diffusion of any innovation in society including CAV. If CAV is not accepted by people, then people would not adapt it or use it, meaning that the technology might fail to be put to use.

4.1 Visualization of potholes and other obstacles

Information-centric technologies start to play a central role in the recent automotive industry boosting new research trends in semi or fully automated driving systems. Autonomous vehicles, ranging from level 3 to level 5 of autonomy [13], are expected to operate safely in real-life road conditions, but the reality is that obstacles like potholes, bumps, and other unexpected objects are not uncommon in an everyday driving context. For this reason, the detection and identification of such events is imperative for the reliable operation of such autonomous systems. To address this issue, we implement a real-time point cloud processing system that takes data generated from a LiDAR device and classifies the road environment into safe and hazardous areas identifying obstacles, pedestrians, other cars, etc. To complement the obstacle detection and tracking, we also develop a multi-agent system of vehicles sharing information between them, so that agents are notified of incoming obstacles even when there is no direct line of sight. Moreover, in the case of semi-autonomous vehicles, where the operator may be asked to take manual control of the car at any moment, a notification scheme – that directs the operator’s, possibly reduced, attention to the event that triggered the takeover request – is important. The state of the operator’s awareness is a core factor in handling the take-over requests. To that end, we propose an Augmented Reality system for presenting all the relevant information to the driver about incoming obstacles. Inherent challenges include the need for non-intrusive information display, avoiding the effects of tunnel vision which could lead to actually overlooking important information. Since the automotive industry is adopting AR interfaces in the form of windshields, AR-based technologies (beyond current AR headsets) are expected to be utilized in the near future for providing guidance to the driver, increasing his/her situational awareness, and facilitating cooperation with other vehicles and road users (e.g., pedestrians, bicycles).

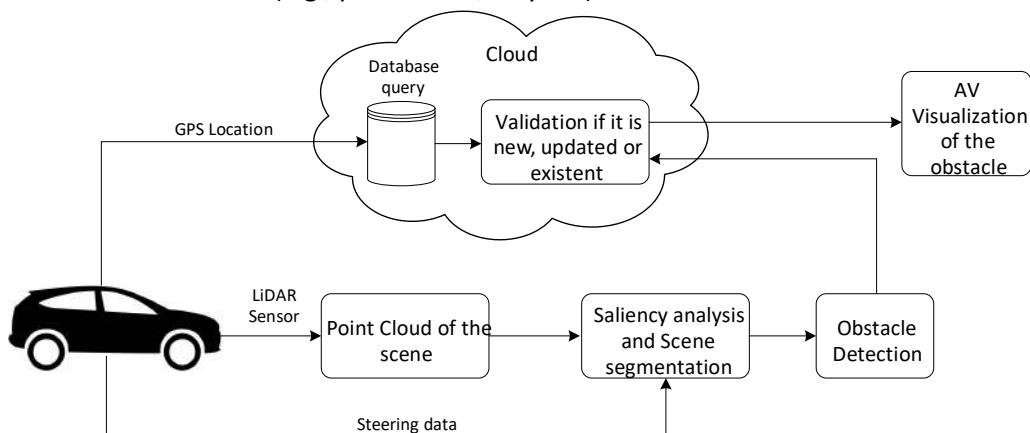


Figure 3: Schematic diagram of the proposed methodology.

We present a methodology on obstacle detection, whereas visualization and communication aspects will be also analyzed. The main components of the methodology are illustrated through a schematic diagram in Figure 3 and can be summarized in the next steps:

- **Extraction of saliency map:** A saliency value is estimated at any point in the point cloud scene based on its local geometry, as well as the geometry of neighboring points.
- **Scene segmentation:** The saliency map is used as feature to segment the point cloud into areas characterizing (i) the safe area of the road, (ii) be-aware or dangerous areas within the range of the road, and (iii) areas out of the range of the road.
- **Static object recognition:** Static objects (i.e., potholes and bumps) are identified and their point coordinates stored and used for AR-based visualization and communication to other nearby vehicles, as will be described later.
- **AR visualization:** The detected objects are projected in the AR coordinate system as 2D images.
- **Calculation of occupancy factor:** In order to adapt the AR interface to the situational awareness risk, we assume that the required level of attention is propositional to the occupancy of the road in front of (and around) the vehicle, and we, thus, formulate a metric (called occupancy factor) that characterizes the amount of obstacles (e.g., cars, potholes, pedestrians) located in the extent of the road, and their relative distance to the vehicle.

The input data constitute a sequence of point clouds \mathbf{P}_i , $i = 1, \dots, l$ that represents a set of l consecutive frames acquired by a LiDAR device. Each point cloud \mathbf{P}_i consists of m_i vertices \mathbf{v} , where the value of m_i may be different from frame to frame. The j -th vertex \mathbf{v}_j of the i point cloud \mathbf{P}_i is represented by the Cartesian coordinates, denoted $\mathbf{v}_j = [x_j, y_j, z_j]^T \forall j = 1, \dots, m_i$, where the index i of the point cloud is omitted for simplification. Thus, all the vertices can be represented as a matrix $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_{m_i}] \in R^3 \times m_i$. Let's also denote with Ψ_j^k the set of the indices of the k nearest neighbors of point j . For a face f defined by three vertices $(\mathbf{v}_{j_1}, \mathbf{v}_{j_2}, \mathbf{v}_{j_3})$, the outward unit face normal \mathbf{n}_f is calculated by the following equation:

$$\mathbf{n}_f = \frac{(\mathbf{v}_{j_2} - \mathbf{v}_{j_1}) \times (\mathbf{v}_{j_3} - \mathbf{v}_{j_1})}{\|(\mathbf{v}_{j_2} - \mathbf{v}_{j_1}) \times (\mathbf{v}_{j_3} - \mathbf{v}_{j_1})\|} \quad (1)$$

The point normal \mathbf{n}_j , representing the normal of each point separately, is calculated as:

$$\mathbf{n}_j = \frac{\sum_{\forall \mathbf{n}_f \in \Psi_j^k} \mathbf{n}_f}{|\Psi_j^k|} \quad (2)$$

4.1.1 Saliency map estimation of the point cloud scene

The purpose of this step is to calculate a metric of saliency at each vertex of a point cloud. For point clouds without context information, saliency characterizes the geometric properties in a local neighborhood, i.e., high saliency values represent more perceptually prominent vertices which usually correspond to sharp corners. On the opposite, the geometrically least important points are those that lie in flat areas. For the estimation of the saliency map, we implemented and modified the fusion technique presented in [14]. Changes in the implementation include the use of normals for the points, instead of guided normals of centroids that were utilized in the original version [14]. This was performed to accelerate computations, since the number of faces is usually approximately twice the number of vertices and thereby, the point normals are almost half in number to the centroid normals. For the sake of completeness, we present here our approach for the estimation of the saliency map of a point cloud scene, utilizing point normals. This fusion technique combines geometric with spectral saliency features that will be described in the following subsections. Each one of these saliency features has unique characteristics, making the whole pipeline to be more robust when it is applied to point clouds that have been acquired under real conditions, thus having been affected by noise and outliers. The method processes each frame independently

without examining past temporal information. Thus, as the methodology is applied for each point cloud in the sequence independently, for simplicity we omit the index i (indicating the frame number) from now on in the equations. For a point cloud \mathbf{P} with m vertices, a matrix $\mathbf{E} \in \mathbb{R}^{3m \times (k+1)}$ is constructed that includes in the first column the m point normals (\mathbf{n}_j) of each vertex $j, j = 1, \dots, m$, and in the subsequent k columns the point normals of the k nearest neighbors of vertex j (i.e. $\mathbf{n}^{j\kappa} \in \Psi_j^k$), as shown next:

$$\mathbf{E} = \begin{bmatrix} \mathbf{n}_1 & \mathbf{n}_{11} & \mathbf{n}_{12} & \dots & \mathbf{n}_{1k} \\ \mathbf{n}_2 & \mathbf{n}_{21} & \mathbf{n}_{22} & \dots & \mathbf{n}_{2k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{n}_m & \mathbf{n}_{m1} & \mathbf{n}_{m2} & \dots & \mathbf{n}_{mk} \end{bmatrix} \quad (3)$$

where $\mathbf{n}_j = [n_{jx}, n_{jy}, n_{jz}]^T$. These saliency features capture global information since for the matrix \mathbf{E} is constructed using the point normals of the whole scene. In order to exploit the geometrical coherence between neighboring normals, we apply Robust Principal Component Analysis (RPCA) to decompose the matrix \mathbf{E} into a low-rank matrix $\mathbf{L} \in \mathbb{R}^{3m \times (k+1)}$ and a sparse matrix $\mathbf{S} \in \mathbb{R}^{3m \times (k+1)}$. The matrix \mathbf{L} consists of the low-rank values $\bar{\mathbf{n}}$ of the point normals \mathbf{n} , represented by:

$$\mathbf{L} = \begin{bmatrix} \bar{\mathbf{n}}_1 & \bar{\mathbf{n}}_{11} & \bar{\mathbf{n}}_{12} & \dots & \bar{\mathbf{n}}_{1k} \\ \bar{\mathbf{n}}_2 & \bar{\mathbf{n}}_{21} & \bar{\mathbf{n}}_{22} & \dots & \bar{\mathbf{n}}_{2k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \bar{\mathbf{n}}_m & \bar{\mathbf{n}}_{m1} & \bar{\mathbf{n}}_{m2} & \dots & \bar{\mathbf{n}}_{mk} \end{bmatrix} \quad (4)$$

while the matrix \mathbf{S} consists of the corresponding sparse values, represented as $\dot{\mathbf{n}}$, which are zero if the row has normals with very similar values (i.e., the vertex lies in a flat area) and very large values if the row has normal with big dissimilarity (i.e., the vertex lies in a very sharp corner).

$$\mathbf{S} = \begin{bmatrix} \dot{\mathbf{n}}_1 & \dot{\mathbf{n}}_{11} & \dot{\mathbf{n}}_{12} & \dots & \dot{\mathbf{n}}_{1k} \\ \dot{\mathbf{n}}_2 & \dot{\mathbf{n}}_{21} & \dot{\mathbf{n}}_{22} & \dots & \dot{\mathbf{n}}_{2k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \dot{\mathbf{n}}_m & \dot{\mathbf{n}}_{m1} & \dot{\mathbf{n}}_{m2} & \dots & \dot{\mathbf{n}}_{mk} \end{bmatrix} \quad (5)$$

Sparsity of the matrix is assumed because flat areas are the most dominant geometrical pattern in most road scenes.

1) *Estimation of the geometrical saliency (global approach):* As the similarity of normals between neighboring points is a measure of geometrical coherence of the local neighborhood, we estimate the sparsity of the dissimilarity of normals and use it as feature for geometrical saliency, s_1 . Low values of the sparse matrix indicate that the normals of the point and its neighbors are similar (low-rank). This means that if all points in a neighborhood have similar geometrical characteristics, the respective patch represents a flat area. On the opposite, if there is high dissimilarity, this means that the surface has an irregular shape. For a point \mathbf{v}_j the geometric saliency feature, s_{1j} , is estimated by the values of the first column of the sparse matrix \mathbf{S} according to:

$$s_{1j} = \|\dot{\mathbf{n}}_j\|^2 = \sqrt{\dot{n}_{jx}^2 + \dot{n}_{jy}^2 + \dot{n}_{jz}^2} \quad \forall j = 1, \dots, m \quad (6)$$

where n_{jx} denotes the scalar value of the x coordinate, of the $[3 \cdot (j - 1) + 1]^{th}$ row, of the 1^{st} column of the \mathbf{S} matrix.

2) *Estimation of the spectral saliency (local approach)*: For the estimation of the spectral-based saliency, s_2 , for a vertex j of the point cloud, we use the submatrix $\mathbf{E}_j \in \mathbb{R}^{3 \times (k+1)}$, that includes the 3 corresponding rows of the matrix \mathbf{E} in Eq. (3):

$$\mathbf{E}_j = \begin{bmatrix} n_{jx} & n_{jx1} & n_{jx2} & \dots & n_{jxk} \\ n_{jy} & n_{jy1} & n_{jy2} & \dots & n_{jyk} \\ n_{jz} & n_{jz1} & n_{jz2} & \dots & n_{jzk} \end{bmatrix}, \quad \forall j = 1, \dots, m \quad (7)$$

In other words, each submatrix \mathbf{E}_j consists of the point normals of a local neighborhood of the vertex \mathbf{v}_j . Then for each one of these local matrices \mathbf{E}_j , the covariance matrix $\mathbf{R}_j \in \mathbb{R}^{3 \times 3}$ is calculated,

$$\mathbf{R}_j = \mathbf{E}_j \mathbf{E}_j^T \quad (8)$$

and decomposed into a matrix \mathbf{U} consisting of the eigenvectors and a diagonal matrix $\mathbf{\Lambda} = \text{diag}(\lambda_{j1}, \lambda_{j2}, \lambda_{j3})$ consisting of the corresponding eigenvalues. Finally, the spectral saliency s_{2j} of each vertex is calculated by the inverse l^2 -norm of the corresponding eigenvalues:

$$s_{2j} = \frac{1}{\sqrt{\lambda_{j1}^2 + \lambda_{j2}^2 + \lambda_{j3}^2}} \quad \forall j = 1, \dots, m \quad (9)$$

Eq. (9) indicates that large values of the term $\sqrt{\lambda_{i1}^2 + \lambda_{i2}^2 + \lambda_{i3}^2}$ correspond to small saliency features implying that the centroid lies in a flat area, while small values of the eigenvalues' norm correspond to large saliency, characterizing the specific centroid as a discriminative point.

3) *Normalization and fusion of local and global saliency*: Finally, we linearly scale the values of the geometric (s_1) and spectral (s_2) saliency in a range of [0-1] and combine them by weighted averaging, according to:

$$s_j = \frac{w_1 \bar{s}_{1j} + w_2 \bar{s}_{2j}}{w_1 + w_2} \quad \forall j = 1, \dots, m_i \quad (10)$$

where \bar{s}_1 and \bar{s}_2 denote the normalized geometric and spectral saliency features, and w_1 and w_2 the corresponding weights. We have used equal weights ($w_1 = w_2 = 1$) in all of our experiments, however the weights can be tuned to emphasize the local or global saliency descriptors.

4.1.2 Visualization results

Figure 4 presents an example of an image representing the front view of the vehicle as it is captured by the RGB sensor of the car. Additionally for demonstration purposes, in this example, a pothole is apparent in the road. Figure 5 presents the corresponding point cloud, captured by the LiDAR sensor of the vehicle, for the same frame as those presented in the previous figure. Moreover, for this point cloud, the saliency mapping has been estimated and colour heatmap visualizes the geometrical importance of each point. Then, in Figure 6 the scene segmentation for the identification of on-road obstacles is presented. The saliency map of each frame is used to categorize different regions of the scene, that are visualized in different colors:

- Blue: the safe area of the road

- Yellow: be-aware areas in the range of the road representing negative obstacles
- Cyan: hazardous areas in the range of the road representing positive obstacles
- Purple: obstacles beyond the boundaries of the road
- Red: Recognized or registered potholes

Figure 8 and Figure 12 illustrate the segmented point cloud projected to the AR interface of ego 1 and ego 2 correspondingly. Figure 7 shows the perspective projection of the points to the AR interface and image filling for the ego 1. In Figure 9 and Figure 10 the pothole recognition and visualization is depicted for the vehicles ego 1 and ego 2, while the visual presentation of a pothole before it is recognized by the ego 2 is presented in Figure 11. Finally, in Figure 13, the registered point cloud map of both two vehicles (ego 1 and ego 2) is presented.



Figure 4: Image from the camera of the vehicle.

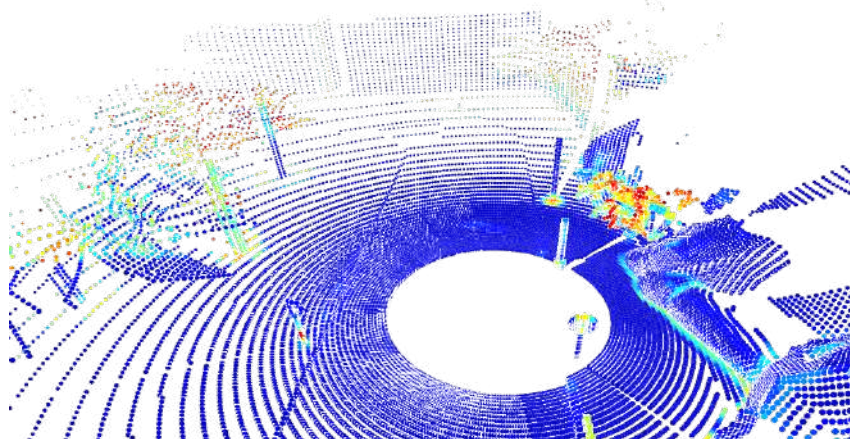


Figure 5: Example of saliency map extracted from the road scene shown in the previous Figure.

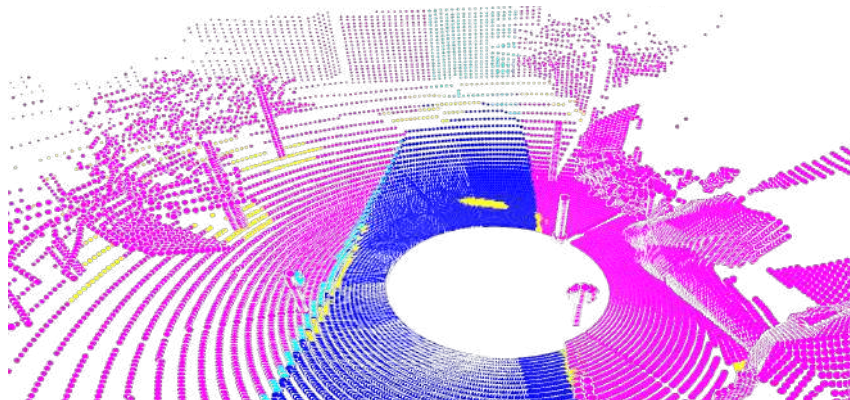


Figure 6: Example of segmentation.

D3.4 AR Interfaces Supporting Object and Scene Manipulation for Increasing Situational Awareness

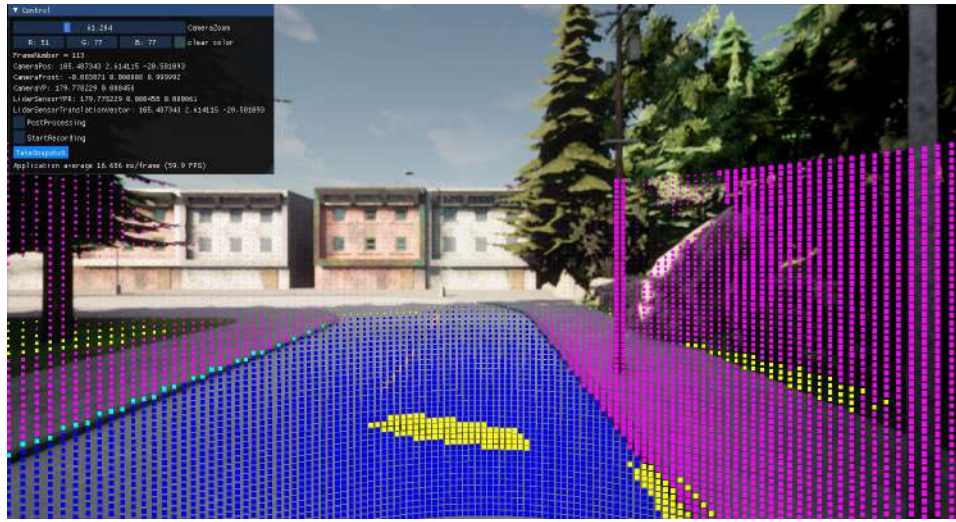


Figure 8: Example of segmentation of the point cloud projected to the AR interface (ego 1).

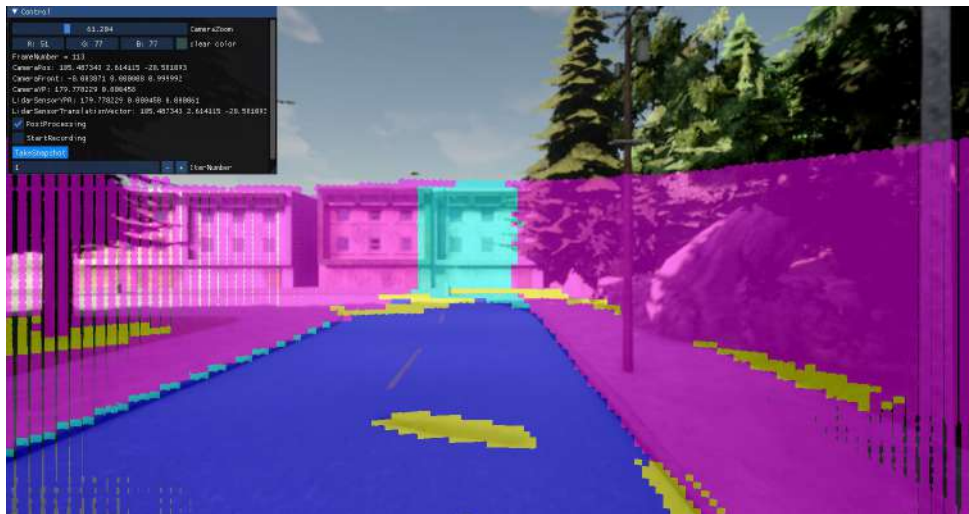


Figure 7: Perspective projection of the points to the AR interface and image filling (ego 1).

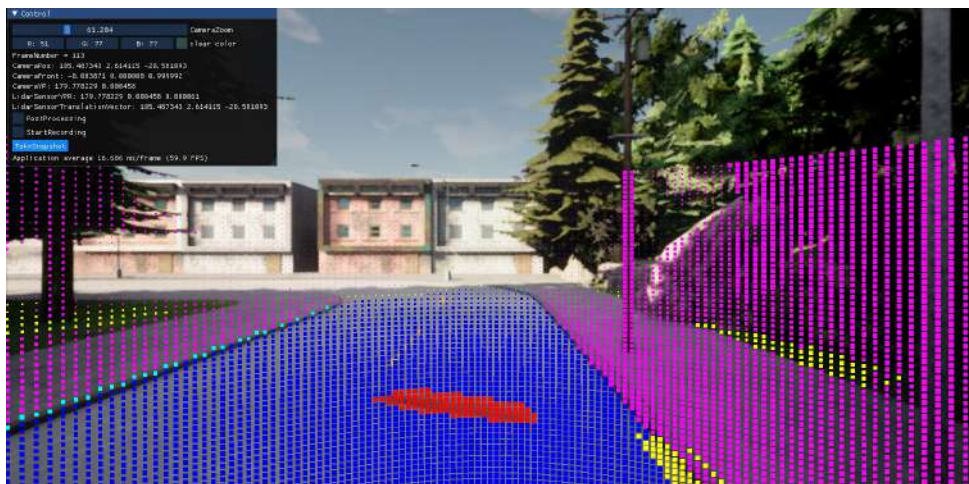


Figure 9: Pothole recognition and visualization (ego 1).

D3.4 AR Interfaces Supporting Object and Scene Manipulation for Increasing Situational Awareness

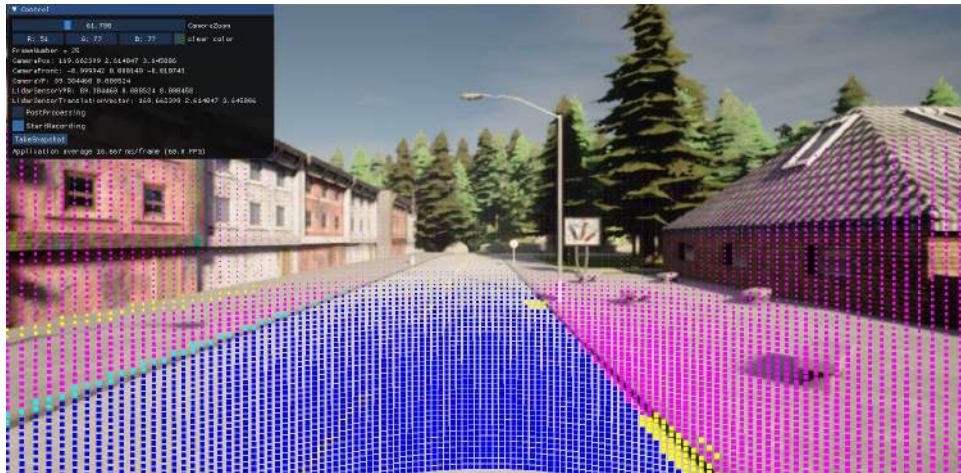


Figure 12: Starting point of ego 2.

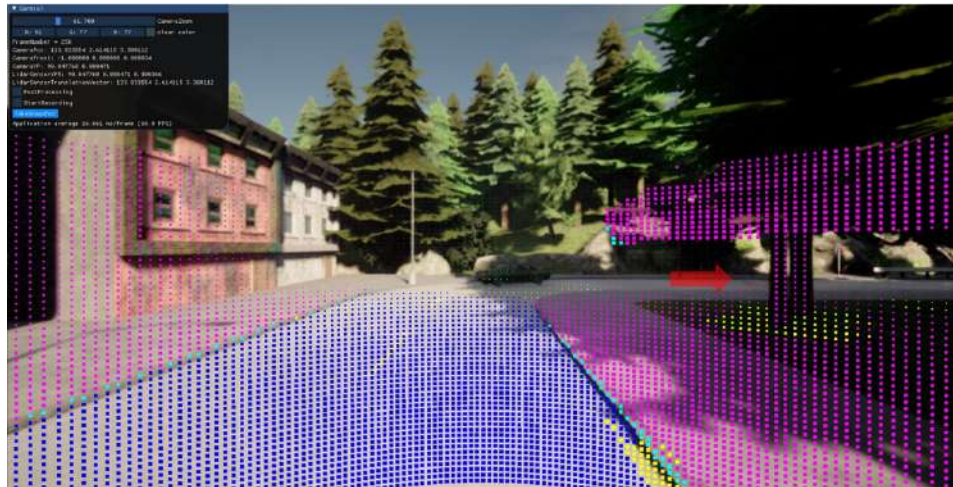


Figure 11: Pothole presentation before be recognized by the ego 2.

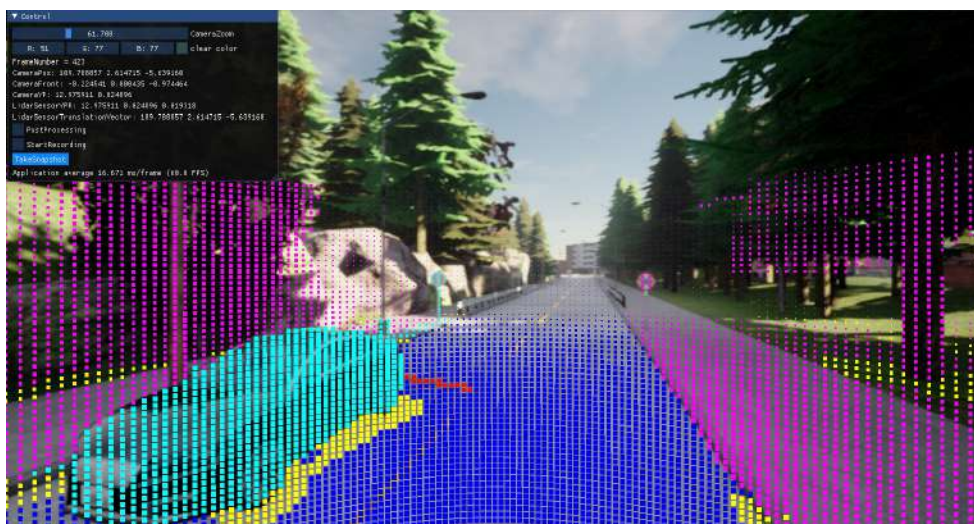


Figure 10: Pothole recognition and visualization (ego 2).

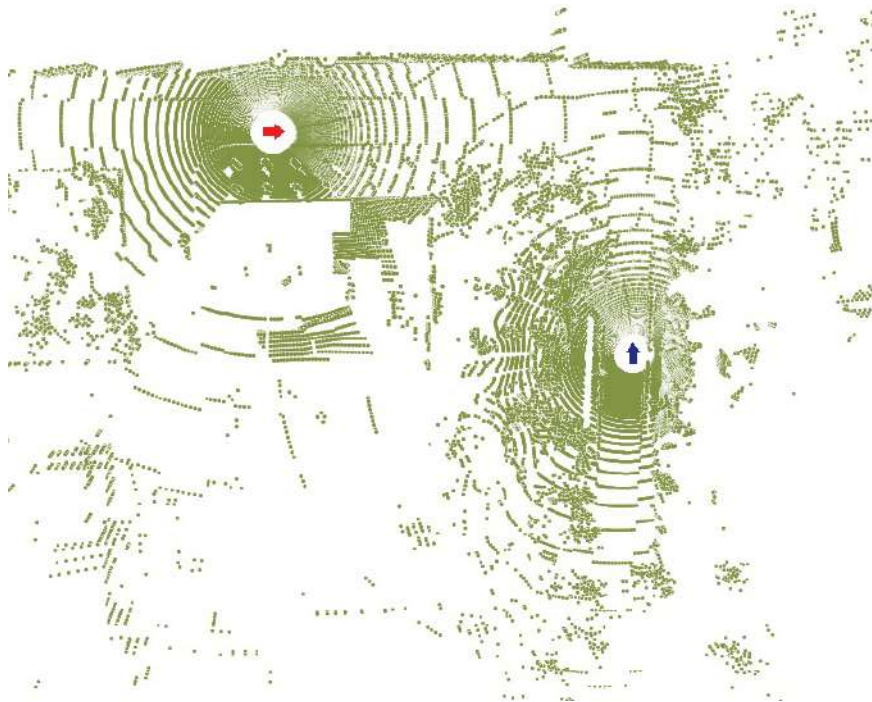


Figure 13: Point cloud map of both two vehicles (ego 1 and ego 2)

4.1.3 Data Simulations

For evaluation of our methodology, we created a rich dataset using CARLA, an open-source autonomous driving simulator [15]. CARLA is based on a server-client system, where the server is responsible for running the simulation that includes the calculation of physics, weather conditions, collision detection and sensor readings. It uses the OpenDRIVE specification [16] for defining junctions, traffic lights, etc., and is used by CARLA for simulating independent agents, such as other cars and pedestrians. This makes CARLA ideal for creating complex scenarios and realistic driving conditions for our tests.

The server running the simulations is powered by Unreal Engine¹. Clients can connect and request changes to a large variety of elements (being essential for the creation of scenarios), receive sensor data and manage input to the vehicle controlled by the user. CARLA supports a wide range of sensor suites with extensive configurability to its intrinsic parameters. In our work, we use a LiDAR sensor on top of the vehicle and a monocular RGB camera, placed in the front part of the car, for simulated data collection. By placing these sensors in an autonomous car and initiating its navigation in the virtual environment, we were able to create a very large dataset for evaluating our methods. In the future, we plan to assess the AR visualization effectiveness, in respect to reaction time and awareness increase, in a real environment with a driver manually controlling a vehicle.

Modifications in the CARLA simulator: Due to lack of datasets consisting of point clouds representing real road scenes with obstacles (potholes and bumps), we used the CARLA simulator to create obstacle-free environment data, in which we subsequently introduced simulated obstacles.

4.1.4 Interfaces and communication

In the case of partial or conditional driving automation, our framework could be used to prepare the driver to

¹ <https://www.unrealengine.com/en-US/>

quickly take the control of the vehicle if requested. Context awareness is a critical factor for successful take-over requests and a lot of effort has been devoted to determine the type of stimulus (e.g., visual, auditory, vibrotactile) [17] and the required time window [18]. In order to ensure that the driver is able to swiftly take-over the control of the vehicle in an efficient way, we developed a notification system that presents relevant information about the condition of the environment. Our notification system is based on non-intrusive visual cues to prevent tunnel visioning, alerting the driver of potential dangers and also directing his/her attention to the objects of interest that sparked the take-over request. In that way, in addition to assisting the human operator during manual driving, the system can, in times of automated driving, trigger the attention of the operator to possible external hazards, preparing him/her to resume control, while also highlighting the objects of interest that triggered the request.

The visualization technique presented in this section is designed as an AR windshield interface, although this is not restrictive, i.e., the method can be implemented in any AR interface.

4.1.5 AR Visualization

The visualization targets two different use cases. The first case is for day-to-day cruising which includes utilities and quality of life applications, while the second case concerns the handover scenario where the driver resumes manual control of the vehicle. The visualization of obstacles is performed by projection. Assuming the position is known for the AR interface and the LiDAR relative to the world, we construct a transformation matrix that transforms the points of the point cloud from the LiDAR relative coordinate system to the AR interface's coordinate system. The transformation between two different coordinates systems is typically done by applying serially a scale, a rotation and then a translation transformation. Since both coordinate systems are orthonormal, the scaling can be omitted. Also, by taking advantage of the rigid body nature of the vehicle where the LiDAR and AR interface are located, we also omit the rotation matrix given that, without loss of generality, we can assume that the two coordinate systems are aligned. According to these assumptions, the LiDAR coordinates are transformed to the AR interface's coordinates by a simple translation.

Afterwards, for projecting the points of the point cloud to the AR interface, we assume a simple pinhole camera model. If the AR interface is, for example, an AR windshield, then the windshield represents the image plane and the head of the driver the principal point with coordinates (x_0, y_0) . That way, the focal distance f represents the distance from the driver to the image plane. With the dimensions of the image plane (windshield), and specifically the aspect ratio, known, the frustum is fully defined and the projection can be made from a point in 3D windshield coordinates (x, y, z) to pixels (u, v) on the image plane using the following transformation.

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

An undesirable property is the sparsity of the projected pixels attributed to the sparsity of the point cloud. To overcome this limitation, we use an iterative nearest neighbor algorithm on the image space to fill the gaps between projected points.

4.1.6 Vehicle Communication

One of the advantages of autonomous vehicles is their ability to communicate with each other forming a cyberphysical system of systems. Many new opportunities arise from the ability of systems to share information, one of which is the transmission of objects or landmarks of interest that were previously observed by an agent, to other agents of the system who could benefit of such information. In particular, our work focuses on sharing

information about encountered obstacles, such as potholes and bumps, to vehicles through a centralized server. When a vehicle identifies an unexpected (i.e. unregistered) obstacle, the vehicle sends a request to the server and, after further inspection, the new potential obstacle is either discarded or added to the database. Vehicles may also send information regarding already known obstacles when they come across them, in case the obstacle needs updating in the database, e.g., it has increased in size or has been fixed. Through this communication system, a driver can be warned about potential hazards that may not yet be in his field of view and thus increase his performance and decision-making abilities. We should clarify that our work does not focus on communication protocols and defense mechanisms against potential network attacks, but rather defines a solid framework describing the roles of each node and the information flow.

By using the LiDAR-based obstacle detection method described previously, the vehicle's transmits the points belonging to the obstacle, segmented from the point cloud scene, to a central server. The information is coupled with a timestamp and the GPS location of the vehicle at that instance. The server then transmits to any vehicle in the vicinity of the obstacle, alerting (autonomous vehicles or human operators) about potential hazards from a large distance and thus helping alleviate the inability of the LiDAR sensor to identify obstacles from such range. In the case of a driver, we also use the AR interface of the vehicle to display, in a non-distracting manner, the location and nature of the potentially upcoming obstacle. As there is a need for periodical evaluation of the objects in the server database and update in the case of changes (potholes being repaired or worsened), we assign a geometry-based descriptor at each obstacle. Thus, every vehicle encountering the obstacle calculates the descriptor of the obstacle's surrounding area. The new descriptor is then transmitted to the server and is used to confirm whether the information is up-to-date or should be updated. An example flowchart of two vehicles showcasing our communication pipeline is shown in Figure 14.

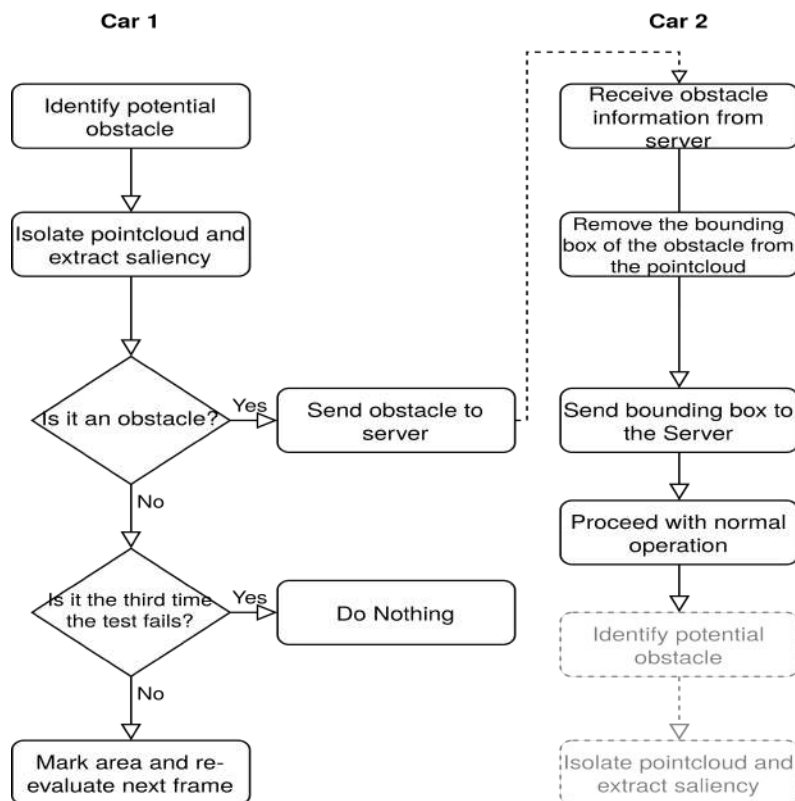


Figure 14: Flowchart of communicative vehicles for obstacle sharing.

4.1.7 Available code repositories

1. [Shared memory communication module \(https://github.com/Stagakis/ipc_shared_memory_demo\)](https://github.com/Stagakis/ipc_shared_memory_demo)

Scope

Implementation and demo for POSIX shared memory for communication between C, C++ and Python.

This repo provides for inter-process communication using tripple buffering to achieve a lock-free producer-consumer communication between processes. Support is provided for both C, C++ and Python in any producer-consumer configuration.

For Python to include the library, use:

```
from tbipc import SharedMemory
```

While for C and C++, add the following include:

```
#include <tbipc.h>
```

Examples

Inside the demo folder you can see examples of the use of tbipc for streaming images between two folders (For that demo, OpenCV is required)

Installation and running

```
mkdir build
cd build
cmake ..
make
sudo make install
```

1. [Generation of data for randomized multi-ego vehicle scenarios \(https://github.com/Stagakis/carla-data-generation\)](https://github.com/Stagakis/carla-data-generation)

A simple offline multi-Ego vehicle data (i.e., images, point clouds) generation script for the CARLA simulator.

To use the script you need to run `multi_data_generator.py`. The configuration variables are listed inside the Class `SimulationParams` in `configuration.py`. From there you can control various aspects of the simulation, like the number of pedestrians, vehicles, town, weather, type of sensors (cameras, LIDAR), etc.

For each Ego vehicle in the simulation that gathers data (minimum 1) it needs to have an associated spawn point inside the simulation and also an associated json file that describes sensors and their extrinsic and intrinsic parameters (for example, see `sensors.json` in the `Config` folder). Both the spawn point and the json file path must be defined at `ego_vehicle_spawn_point` and `sensor_json_filepath` respectively in Class `SimulationParams` in `configuration.py`.

The implementation uses queues and synchronous communication with the server such that the sensor output is always exactly synchronized with the world clock.

How to run

`python3 multi_data_generator.py` and a folder `_out` will be created with the respective Ego and sensor data.

2. Creation of potholes to be entered in Carla's towns (<https://github.com/Stagakis/roadpatch-with-pothole-generator>)

This is an automated script to ease the workload of creating potholes for the purpose of inserting them into a CARLA map. Included in the repo is also a meshlab script to automate the meshing process of the point cloud of the pothole for insertion in the Carla map.

Dependencies

The code uses the following packages:

- numpy
- open3d
- sklearn

How to use

The `pothole.py` script is responsible for reading the point cloud of the pothole and adding a flat plane around it to simulate a road patch. For this to happen, it needs to be supplied with the folder of the `.ply` files from this repo as its input:

https://github.com/ruirangerfan/rethinking_road_reconstruction_pothole_detection/tree/main/dataset/model1].

The script afterwards outputs the result in the `completed_potholes/` and also populates the `intermediate_files/` folder with the last processed pointcloud (use this for debug purposes).

The `eval.py` script in turn, reads the ground truth point cloud (the version with the road patch generated from `pothole.py`) and the point cloud generated from this repo (<https://github.com/Stagakis/saliency-from-pointcloud>) and outputs a confusion matrix for the point classification (pothole and road classes).

3. Saliency map extraction from point cloud (<https://github.com/Stagakis/saliency-from-pointcloud>)

This code is mainly used for the estimation and extraction of the saliency mapping given a point cloud. More specifically the outcome of this code is:

- Files with the saliency values per each point of the point cloud
- Heat map visualization of the saliency mapping, indicating the high and the low saliency values
- Segmentation of the point cloud scene into regions of interest as well as the pothole and obstacle estimation and visualization

4. Point cloud projection to RGB images <https://github.com/Stagakis/carlapclprocessing>

Scope

This repo is our point cloud visualization framework on RGB images. The point cloud is color coded for identifying saliency regions and the code for the classifications is in this repo <https://github.com/Stagakis/saliency-from-pointcloud>. This work is part of our research on driver awareness using AR displays. Both the point cloud and the image are generated from CARLA. At the moment, it is operated offline, but plans for online implementation are on the roadmap.

Resources

Download the resources (LIDAR point cloud, images, etc.) and extract them. The program by default checks in the top level directory of the project for resources0/ and resources1/ folder.

[Resources for 2 agents](#)

Dependencies

The project is striving to be portable, everything needed should be in deps folder.

Installation and running

Until some issued with the commit history are fixed, I recommend you to use git shallow clone. After the cloning is complete, navigate to the project folder and run:

```
mkdir build
cd build
cmake ..
make
./ARCarla
```

4.2 Visualization of occluded vehicles VR/AR UPAT simulator

DerinioSim is an autonomous driving simulation built in Unity3D using C# that has been developed for the purposes of CPSoSaware. The repository with all the code and the examples can be found in <https://gitlab.com/vvr/deriniosim/-/tree/master/>.

The master branch contains only the required assets to run/configure the core simulation, i.e. the overall car behavior (applying torques, steering, keeping distances from other cars etc.) and car paths. Before proceeding to analyze the main components, let's take a quick look on the available scenes and their correspondent purposes:

Scenes

1. **Configuration Scene:** In that scene, the user can configure/debug the desired paths for the automotive simulation
2. **Main Scene:** By using the saved configurations from the previous scene, a user can experience an auto/manual driving simulation.

Then, we proceed to describe the main assets of the project:

Assets

1. **Nature:** Nature Objects such as trees, bushes etc. Mostly related to the environment and background. We grouped them to avoid confusion with custom packages that one may create. In general, they should not be modified.
2. **Prefabs:** Prefabs that are actively used to the simulation such as NPC cars and points
3. **Windridge City:** Same principles as **Nature** folder.

Following, we will analyze the scripts, how car path configuration is established and how a car function cycle takes place.

Folders

1. **General:** Folder that provide general functionalities such as finding the nearest object with a specified layer, the minimum distance or demonstrate a pseudorandom behavior.
2. **Intro Scene:** Folder that contains everything related to the configuration of every possible car path in the scene.
3. **Main Scene:** Folder that contains every additional, in comparison with the intro scene, functionality related to the main scene.
4. **Car Behavior:** Folder that contains all the car behavior functionalities.
5. **Player Related Scripts:** Additional scripts related to handling specific functionalities mainly for the player, such as switching from auto to manual mode.

All the main function handling is being done through the Data script. This script is attached on the CarPlayer gameobject and is the main script of the simulation. Inside you can see that a series of other helpful classes are called and updated on every frame. The main output of the Data script can be seen on the Unity Editor, by clicking the PlayerCar when running the simulation.

4.2.1 Configuration Scene / Path Finding / Basic Assumptions

We assume that a car is either turning or going straight. Whenever we want the car to change its orientation, even the smallest modification, we consider it a turn.

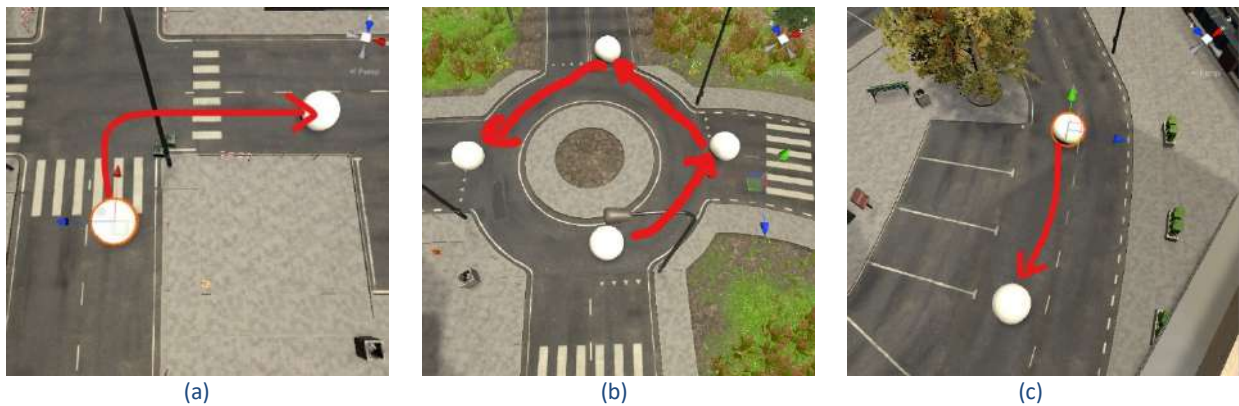


Figure 15: (a) Proper Turn, (b) Roundabout (3 turns), (c) Small Turn

I want to create my own path. How should my point's coordinate system be oriented?

- For **Start Points** the **RED** axis should be pointing to the front Midpoint.
- For **Mid Points** the **RED** axis should, generally, point straight ahead. In case I want a short turn, I should, point the **RED** axis as straight as I can to the target StartPoint.

Which are the type of points?

We have three types of points:

1. **Spawn Point:** Cars are being spawned here.
2. **Start Point:** You'll find them at the start of each *straight line* and at the end of each *turn*.
3. **Mid Point:** You'll find them at the start of each *turn* and at the end of each *straight line*.

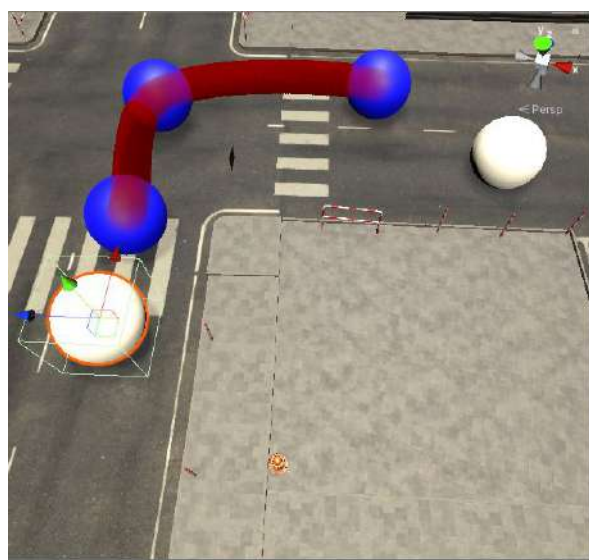
How straight lines are being generated?

No lines are generated, we just assign in the **Start Point** the front **Mid point** through a script. As a result, whenever a car is near a **Start Point**, it will be directed to the correspondent **Mid Point**.

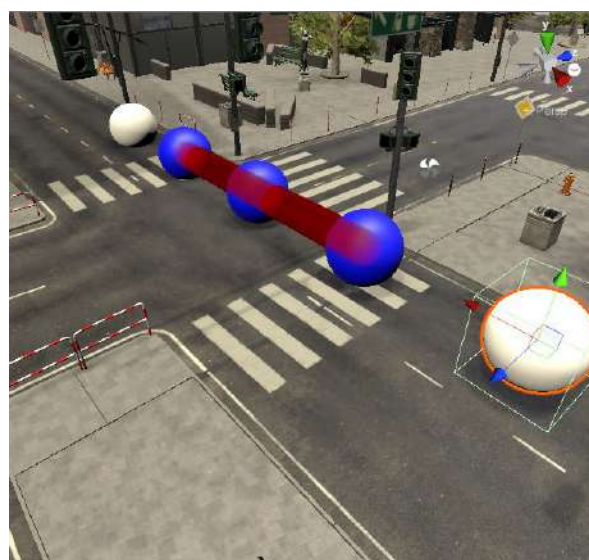
How turns are being generated?

Before analyzing the turns, we should consider the basic steps included in turn generation:

1. Find the target Start Point
2. Spawn three extra control points (*Point Closest To Mid Point*, *Intersection Point*, *Point Closest To Start Point*)
3. Interpolate using Catmull-Rom through the set of these 5 points:
 - a) Mid Point
 - b) Point Closest To Mid Point
 - c) Intersection Point
 - d) Point Closest To Start Point
 - e) Start Point
4. Spawn the spline



(a)



(b)

Figure 16: (a) Proper Turn, (b) Short Turn

How these extra control points are being spawned?

Taking into account that we have two kinds of turns (a really short and the "proper") we use a different method for each one.

Short Turn:

1. Spawn the IntersectionPoint in the middle of the distance between MidPoint and StartPoint
2. Same for ClosestToMidPoint for the distance between MidPoint and IntersectionPoint
3. Same for ClosestToStart for the distance between IntersectionPoint and StartPoint

```

...
// if angle between the two vectors is smaller than 3 degrees
if (Vector3.Angle(onNormal, dirFromMidToStartPoint.normalized) <= 3f) //Short Turn
{
    positionOfIntersectionPoint = midPoint.transform.position + dirFromMidToStartPoint.normalized *
dirFromMidToStartPoint.magnitude / 2f;
    positionClosestToMidPoint = midPoint.transform.position + dirFromMidToStartPoint.normalized *
dirFromMidToStartPoint.magnitude / 4f;
    positionClosestToStart = midPoint.transform.position + dirFromMidToStartPoint.normalized *
dirFromMidToStartPoint.magnitude * 3f / 4f;
}

```

Proper Turn:

1. Find the Vector from MidPoint to StartPoint
2. Create the projection of the previous vector on MidPoint.transform.right Vector
3. Spawn the IntersectionPoint in the *MidPoint.transform.right* direction by adding to MidPoint.transform.position the magnitude of the projection vector + an offset
4. Spawn the ClosestToMidPoint *MidPoint.transform.right* direction by adding to MidPoint.transform.position the half magnitude of of the projection vector
5. Spawn the ClosestToStart in the normalized *dirFromIntersectionPointToStart* direction by adding to positionOfIntersectionPoint the half magnitude of the dirFromIntersectionPointToStart vector. Last but not least, we add in the *MidPoint.transform.right* direction multiplied by an offset1.

```

else //Proper Turn
{
    positionOfIntersectionPoint = midPoint.transform.position + onNormal * (projectionOfVectorAtoB.magnitude + offset);
    positionClosestToMidPoint = midPoint.transform.position + onNormal * (projectionOfVectorAtoB.magnitude / 2f);
    Vector3 dirFromIntersectionPointToStart = startPoint.transform.position - positionOfIntersectionPoint;
    positionClosestToStart = positionOfIntersectionPoint + dirFromIntersectionPointToStart.normalized *
(dirFromIntersectionPointToStart.magnitude / 2f + offset)+midPoint.transform.right*offset1;
}
...

```

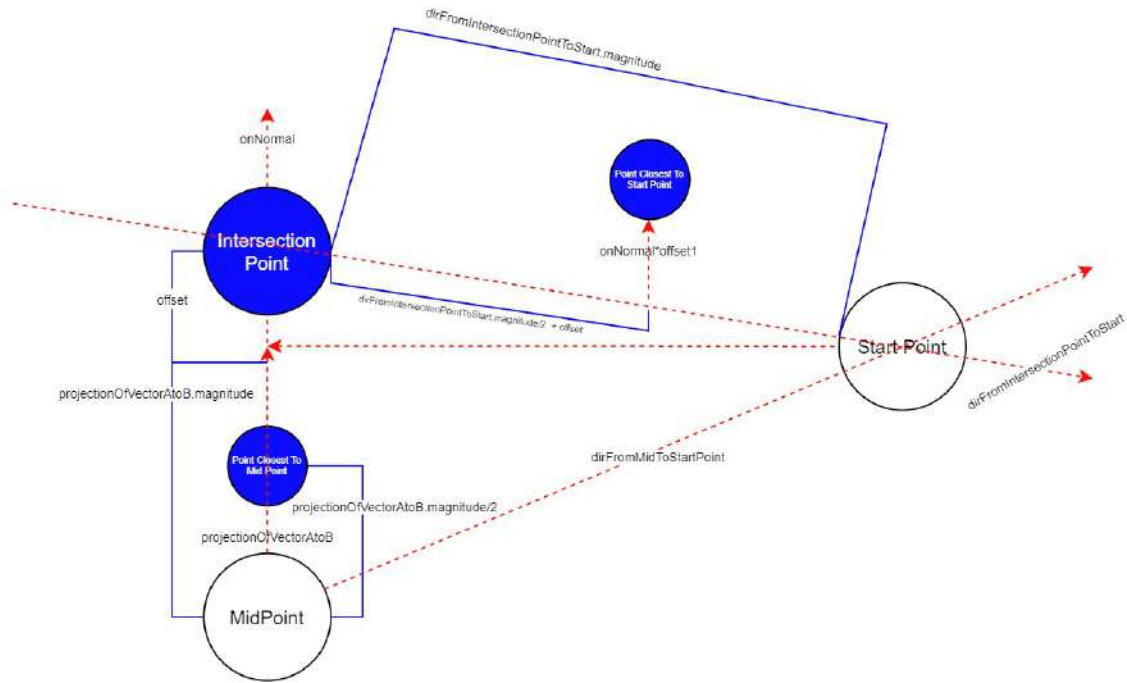


Figure 17: Proper Turn

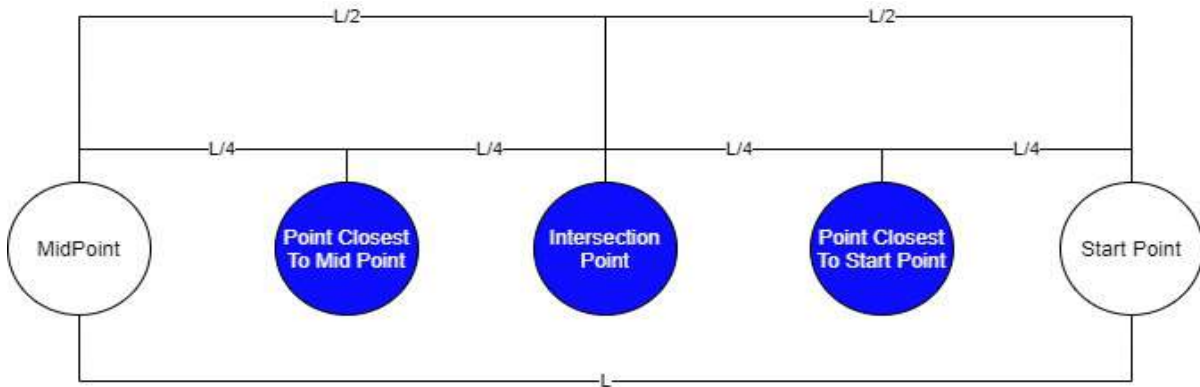


Figure 18: Short Turn

4.2.2 Procedure of Path Finding/Creating

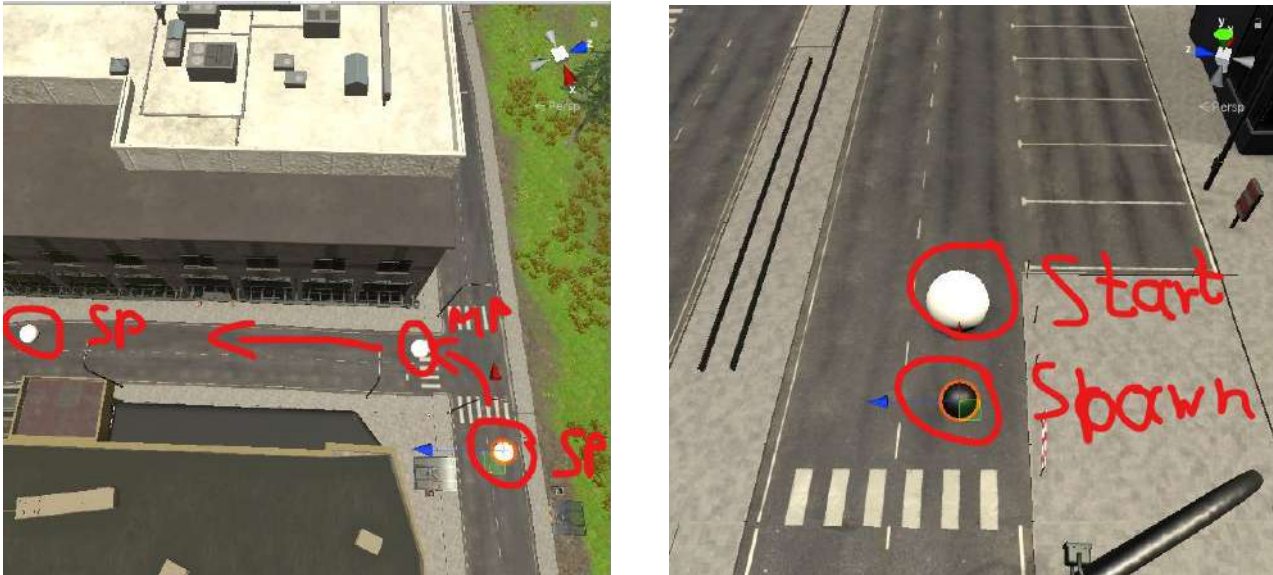


Figure 19: Path Finding Short Turn

Path finding procedures, can be located on **Configuration Scene**. In hierarchy, an individual may find, the *GeneralManager* gameobject which has an instance of *SimulationManagement.cs* and two child gameobjects, namely *Point Manager* and *Car Spawner*.

SimulationManagement.cs

```
_pointRegister.AssignPossiblePaths(); //Create all the possible paths that a car can take
_carSpawner.SpawnTheCars(); //Spawn a number of cars to test your paths
if (_saveAssets) // In case you want to auto spawn prefabs of these new assets (new Spawnpoints and Paths) to use them in
main scene.
{...}
```

AssignPossiblePaths (...)

- Assign the proper .cs to every point w.r.t their layer

```
public GameObject _paths; //Get the root element of all points
public void AssignPossiblePaths()
{
    LayerMask midPointlayer = LayerMask.NameToLayer("Path_MidPoint");
    LayerMask startPointlayer = LayerMask.NameToLayer("Path_Start");

    foreach (Transform child in _paths.transform)
    {
        if (child.gameObject.layer == startPointlayer) child.gameObject.AddComponent<StartPoints>();
        else if (child.gameObject.layer == midPointlayer) child.gameObject.AddComponent<MidPoints>();
    }
}
...
```

- Get the front *MidPoint* for each *StartPoint* and save it to the correspondent instance of *StartPoints.cs*

- Inform that *MidPoint* that it shouldn't (when searching for *StartPoints*) take into account that *StartPoint*.

```
...
foreach (Transform child in _paths.transform)
{
    if (child.gameObject.layer == startPointlayer)
    {
        child.gameObject.GetComponent<StartPoints>()._firstMidPoint =
StartPoints.AssignTargetMidPoint(child.gameObject);

child.gameObject.GetComponent<StartPoints>()._firstMidPoint.GetComponent<MidPoints>().StartPointDirectlyBehindMe
= child.gameObject;
    }
}
...
```

- Find every available *StartPoint* for each *MidPoint*
- Save the correspondent splines

```
...
foreach (Transform child in _paths.transform)
{
    if (child.gameObject.layer == midPointlayer)
    {
        child.gameObject.GetComponent<MidPoints>()._targetStartPoints =
MidPoints.StartPointsInRange(child.gameObject.transform,
child.gameObject.GetComponent<MidPoints>().StartPointDirectlyBehindMe, child.gameObject.tag);
    }
}
foreach (Transform child in _paths.transform)
{
    if (child.gameObject.layer == midPointlayer)
    {
        child.gameObject.GetComponent<MidPoints>().startTargets = MidPoints.GenerateTurns(child.gameObject,
child.gameObject.GetComponent<MidPoints>()._targetStartPoints);
    }
}
}
```

How a Car navigates during the Simulation?

D3.4 AR Interfaces Supporting Object and Scene Manipulation for Increasing Situational Awareness

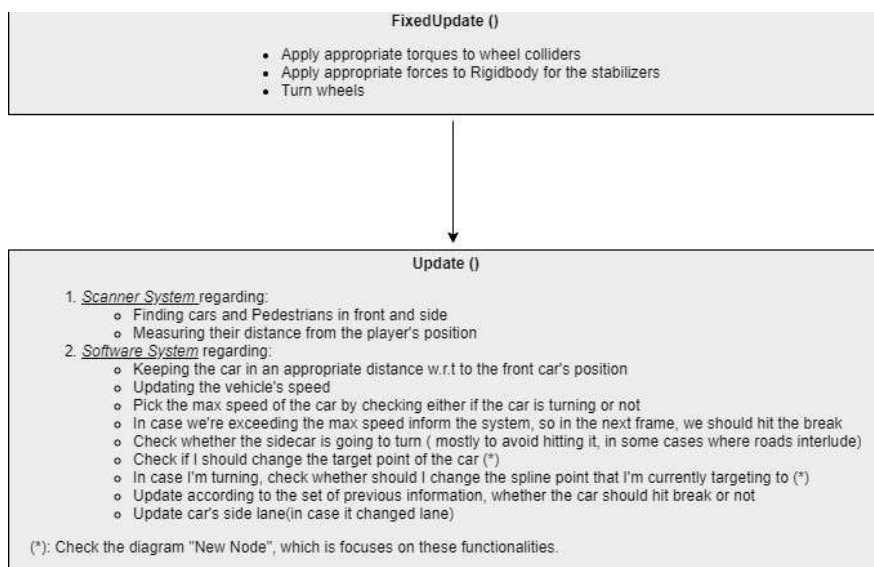


Figure 20: Engine Methods

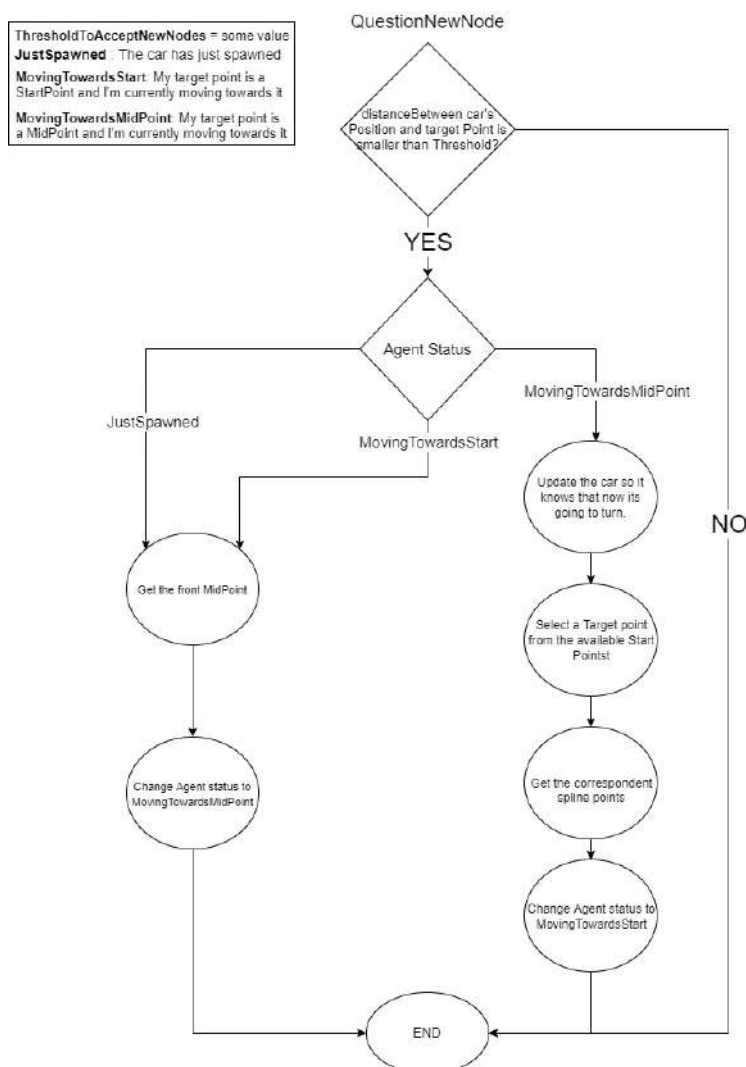


Figure 21: Question New Node

Sensor System

Every car entity is equipped with rays that update surrounding information regarding pedestrians, scooters, and nearby vehicles, during the driving simulation. These are being recast every frame, in Unity3D and are displayed below in figures, simulating LiDAR systems, that have been already presented in current autonomous vehicle technology.

Visualization Techniques

A visualization technique should be composed of multiple attributes to be considered valid (Figure 22). These are:

- Target Representation: The elements that correspond to our surrounding objects.
- User-target indicator: The object that acts as the user's reference visualization
- Direction indication: The forward vector of user's orientation.
- Distance indication: The way of providing the distance information to the user.

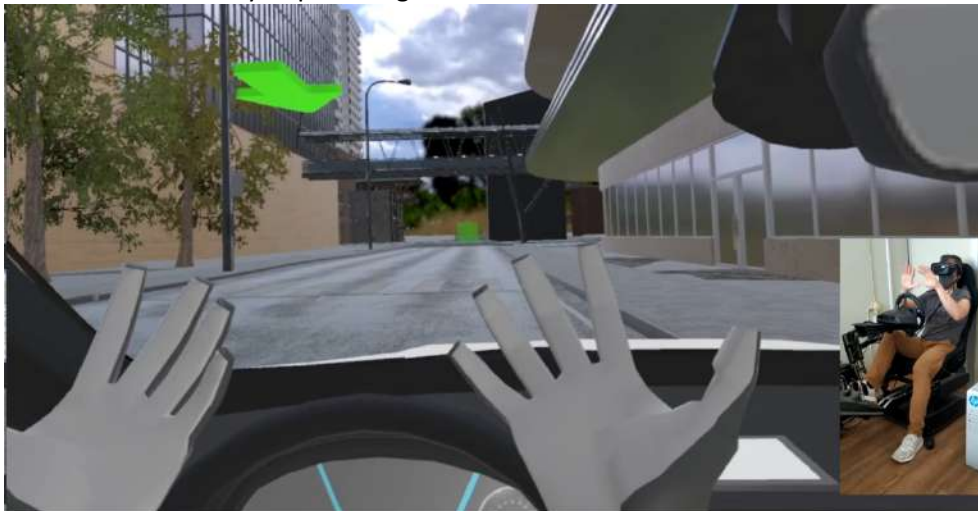


Figure 22: VR simulator and set up implementation.

The four, state of the art, techniques that were, primarily, developed were 3D Wedge, 3D Arrow, 3D Minimap and Radar. We also proceed to document the 3D Wedge's enhancement implementation (3D Wedge+) and, finally, introduce the technique of rendering plain occluded meshes in different forms. Their correspondent positions with regard to the driver's HMD, are modified to achieve aesthetically pleasing result in VR.



Figure 23: Visualization using 3D wedge.

3D Wedge

This visualization method (Figure 23) is based on rendering objects in a form of a pyramid-like scheme with the square base on the object's end. The height of the pyramid is linearly scaled with regard to the distance between the user's car reference and the object, where the side length of the base has a non-linear connection. The above methodology, in combination with transparent visualization and removal of lines that may obscure visual interpretation, can provide information about several nearby target's in the desktop/VR environment.

As a reference point, the user's vehicle position is being used. The forward vector is correlated with user's heading. All 3D Wedges are anchored to it and the overall cluster shows the correspondent distance and directions of the objects. To minimize visual obstruction, we proceed to also employ the introduced angle rotation of the above 3D Wedge system equal to this between the vertex reference point and the user's view.

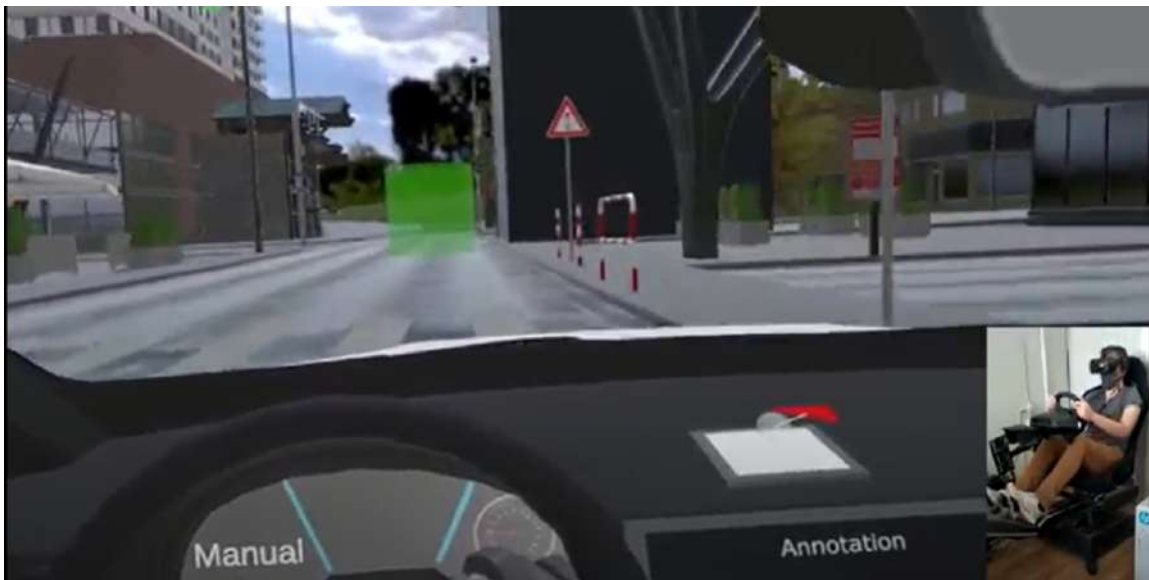


Figure 24: Visualization using 3D arrows.

Arrow

In this implementation (Figure 24), a cluster is being created, comprised of three-dimensional arrows. Their mesh involves a stick body and an arrow tip. Each 3D Arrow's direction follows the paired objects position and is anchored to a central vertex point, which represents, the user's vehicle position. The length of the stick is linearly scaled as a function of the correspondent distance, with regard to a predefined threshold by the research team.

Arrows are limited to display off screen and occluded information, thus, showing only cars, pedestrians and scooters that aren't in user's FOV, as shown below:

Whenever an offscreen/occluded object is to be displayed, an arrow is instantiated with a random colour. In case of absence of occluded information, the anchor point is disabled.



Figure 25: Visualization using 3D minimaps.

3D Minimap

3D Minimap is a visualization method (Figure 25) composed of three layers of concentric spheres. These three levels provide an estimation of the object's distance in relation to the vehicle's position, which is being represented with the correspondent model. That object is placed in the middle of the sphere, where the occluded/off-screen information is being rendered to the form of small spheres placed on the overall system with the appropriate position. In our case, we placed a car mesh in the middle of the minimap and represented user's Field of View (FoV) with a semi-transparent triangle mesh, as shown below.



Figure 26: Visualization using 3D Radar.

Radar

Radar comprises one of the 3D radar visualization techniques being studied (Figure 26). Objects of interest are represented as small squares, while text is added to display the height information. The visualization has a thick

highlighted line coming out from the middle to represent the user's orientation direction and rotated whenever he/she rotates his/her head horizontally.



Figure 27: Visualization of the occluded silhouette via a 3D mesh overlay.



Figure 28: Visualization of the object via a representative 3D sphere.

Since the navigated 3D environment was almost flat, we chose not to include height information in our scenarios.

Occluded Meshes Overlay

This technique is relatively simple, in comparison with the previous ones and mainly focuses on rendering occluded information at the appropriate distance (Figure 27). These meshes can vary from visualizing the as-is object to representing it as a plain sphere (Figure 28). To mimic HMDs, we proceeded to render objects only visible by the correspondent entities, in each case. For distance information, we chose to annotate it above the rendered mesh.

As one may notice, in our experiments, we performed slight modifications to the existing visualization techniques to incorporate the vehicle entity to the simulation. Namely, we are updating the position of the objects regarding the car orientation not the user's HMD while we still proceed to render the heading vector according to him. That is because, whilst being on the road, the user may, continuously, engage into several slight head movements thus changing HMD's horizontal/vertical orientation resulting in confusion. In addition to that, we proceed on implementing certain annotation regarding the occluded objects categories to reduce increasing attention span.

Design

First, participants were provided with a short amount of time inside the simulation's introductory scene to familiarize themselves with the Gamified UI, car behavior, controls, and navigation system for both auto and manual mode. We also deployed a motion simulator chair to emulate the forces being applied to the driver's whilst accelerating.

Due to the existence of two different, in nature evaluation metrics, we proceed to adjust the vehicle's autonomy level (auto/manual) and the set of related objects to the correspondent case-study. The nature of the developing hazards is also fine-tuned to mainly include visualization obstruction generated dangers. Due to the fact, that currently we are focusing on capturing, in a qualitative manner, the general results, we proceed on providing 1 test for each visualization technique to our small number of participants (5 individuals).

AR Implementation

The AR implementation mimics the VR functionality (Figure 29) by attempting to provide to the end user an augmented view with rendering occluded objects in the FOV. The rendering techniques presented in the VR application are also implemented here.

In respect to the AR application infrastructure, in order to be able to track and thus render the occlude objects in the Hololens 2 device, we have attached a mobile device to every moving target and propagated their GPS coordinates to the Hololens Device through an android app. Then by receiving the positions of each target in combination with the coordinates of the Hololens device (i.e the user) we are able to compute the distance between them and render these objects in an appropriate distance in Unity3d.

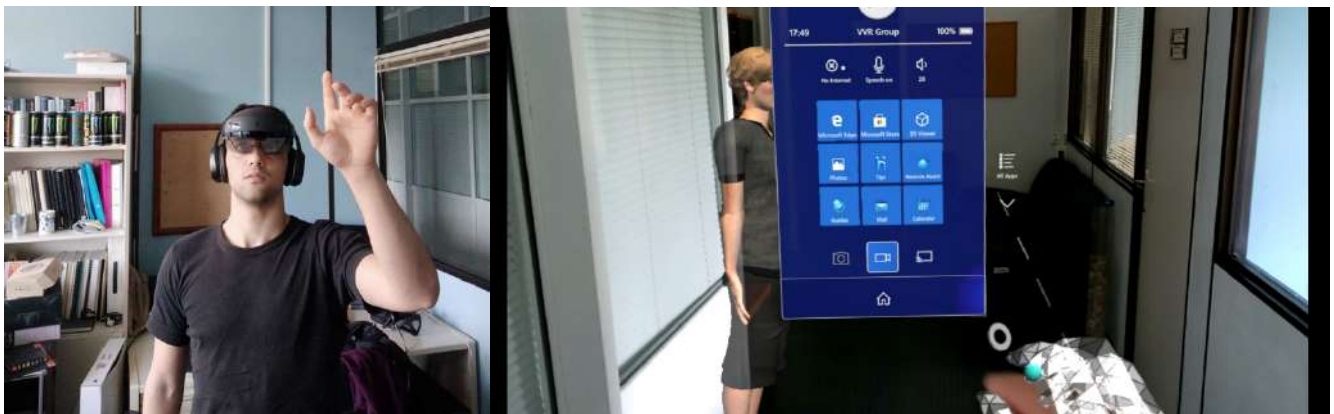


Figure 29: Example of the AR implementation.

4.3 Warnings related to driver's drowsiness and other security aspects

In this section, we will present the visualized results of the drowsiness detection implementation that provides real-time monitoring of users/drivers and estimates their drowsiness based on metrics (e.g., EAR, PERCLOS) proposed by the literature (More details about the implementation have been discussed and presented in D3.1) https://gitlab.com/isi_athena_rc/cpsosaware/drowsiness-detection.

The dependencies of the repository

```
# import the necessary packages
from scipy.spatial import distance as dist
from imutils.video import VideoStream
from imutils import face_utils
from threading import Thread
import numpy as np
import playsound
import argparse
import imutils
import time
import dlib
import cv2
import csv
from datetime import datetime
from datetime import timedelta
```

Estimation of the EAR factor

```
# compute the euclidean distances between the two sets of
# vertical eye landmarks (x, y)-coordinates
A = dist.euclidean(eye[1], eye[5])
B = dist.euclidean(eye[2], eye[4])

# compute the euclidean distance between the horizontal
# eye landmark (x, y)-coordinates
C = dist.euclidean(eye[0], eye[3])

# compute the eye aspect ratio
ear = (A + B) / (2.0 * C)
```

Visualization of the eye contour

```
# compute the convex hull for the left and right eye, then
# visualize each of the eyes
leftEyeHull = cv2.convexHull(leftEye)
rightEyeHull = cv2.convexHull(rightEye)
cv2.drawContours(frame, [leftEyeHull], -1, (0, 255, 0), 1)
cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1)
```

Visualization of an alarm message when the drowsiness is detected

```
# draw an alarm on the frame
cv2.putText(frame, "DROWSINESS IS DETECTED!", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
```

Visualization of information related to EAR and PERCLOS factor and the capturing time in seconds

```
# draw the computed eye aspect ratio on the frame to help
# with debugging and setting the correct eye aspect ratio
# thresholds and frame counters
cv2.putText(frame, "EAR: {:.2f}".format(ear), (300, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
cv2.putText(frame, "PERCLOS: {:.2f}".format(perclos)+" NF: {:.2f}".format(Number_of_frames)+" CNF:
{:.2f}".format(Close_number_of_frames), (30, 40), cv2.FONT_HERSHEY_SIMPLEX, 0.4, (0, 255, 255), 2)
cv2.putText(frame, "seconds: {:.2f}".format(time.time()-start_time), (300, 60), cv2.FONT_HERSHEY_SIMPLEX, 0.3, (255, 0,
255), 2)
```

Occupancy factor

Additional security aspects are calculated (i.e., occupancy factor), the value of which represents the free available space of the road around the moving vehicle.

- After the acquisition of the point cloud, by the LIDAR sensor, a geometry processing step occurs so that the saliency map of the point cloud to be estimated.
- Saliency mapping corresponds to each vertex of the point cloud with a value based on its geometrical importance.
- Vertices that lie in totally flat areas take the lowest value, while vertices lying in very sharp corners take the highest value.
- For the estimation of the occupancy factor, we take into consideration only these vertices that (i) belong to the range of the road and also (ii) correspond to the lowest saliency value.
- Finally, the occupancy factor is estimated as the sum of the inverse norm2 distance between each vertex, of the previously aforementioned set of vertices, and the point (0, 0, 0) of the vehicle.
- The bigger this value, the more free road space for driving.

Integration with wheel chair (Figure 30)

- Integration of DMS with CARLA environment to evaluate the dangerousness of the driving under different states of driver's drowsiness and different conditions of the road (e.g., traffic, Figure 31-Figure 33). More details will be presented in D4.5.
- We use an empirical metric (occupancy factor), extracted by the LIDAR representation of the scene, that shows how "clear and open" is the road beyond of the driver's field of view.
- Based on this value the road's condition can be separated into three categories:
 - Safe road without objects
 - Road with small objects (e.g., potholes) or cars in a far distance from the vehicle
 - Road with a lot of traffic, parked cars in a range of the road etc.



Figure 30: Integration of DMS with CARLA environment using wheel chair

D3.4 AR Interfaces Supporting Object and Scene Manipulation for Increasing Situational Awareness

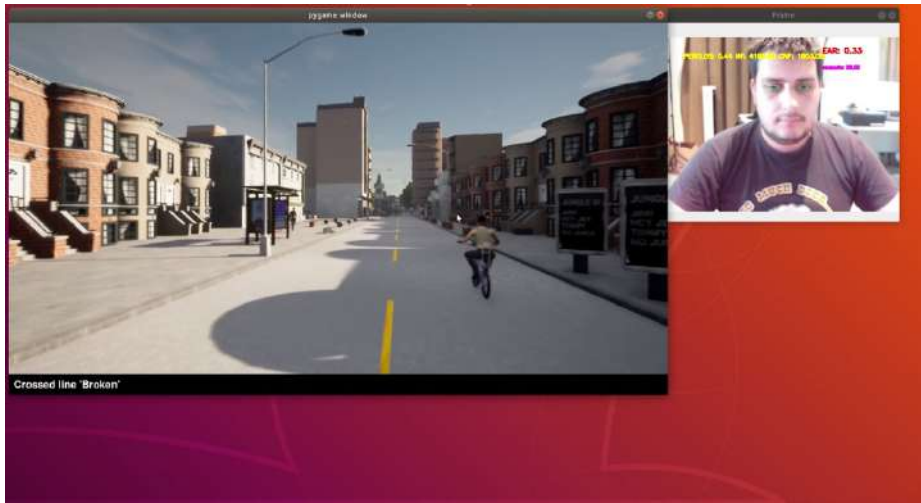


Figure 31: Visualization of EAR, PERCLOS, capturing time, occupancy factor and driver's eye contour

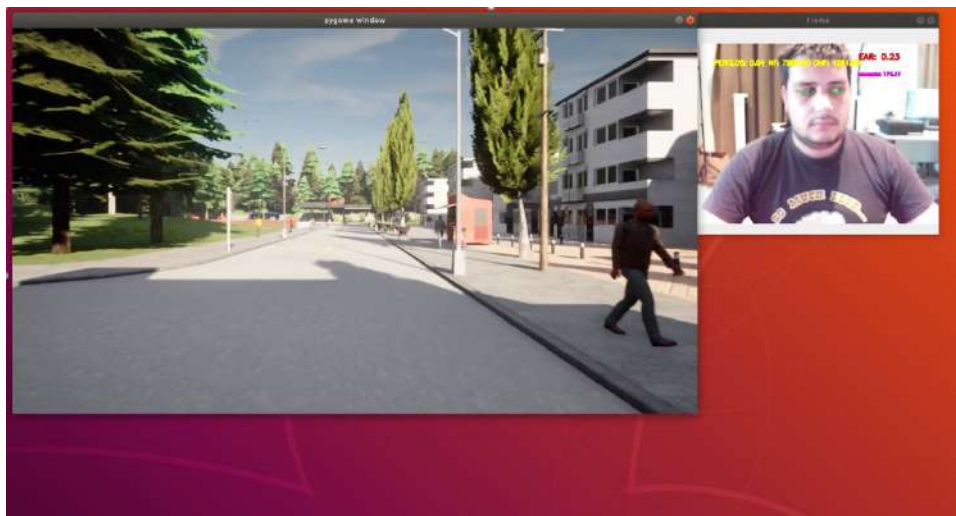


Figure 32: Visualization of the eyes shape during the driving

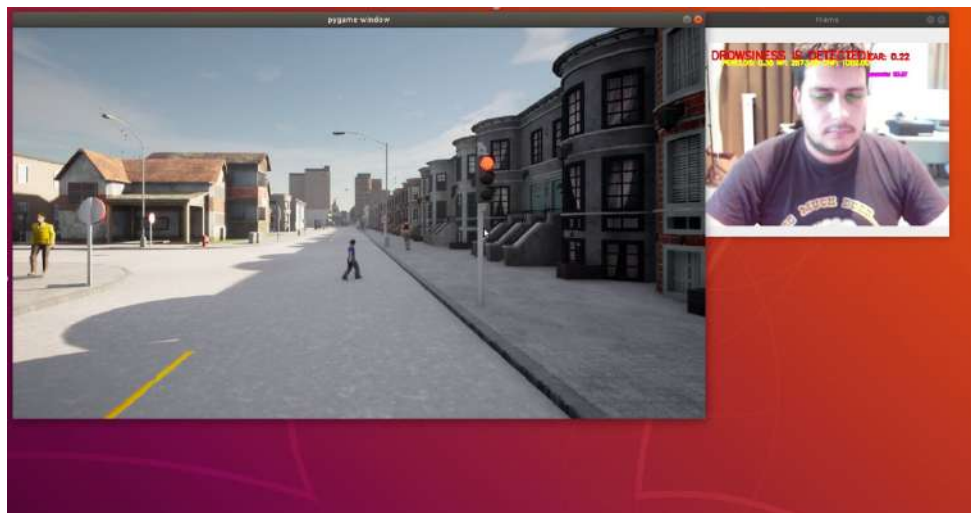


Figure 33: Drowsiness detection alert

5 Situational awareness in manufacturing

To ensure maximum results, the majority of industrial companies have already incorporated safety courses regarding the use and even the presence of workers around heavy operating machinery in their reoccurring training schedules. Risk assessment, process control and machine safety educative services are now even outsourced to specialized external partners to achieve optimal performance. This can be justified by the fact that a high paced environment which incorporates, in our case, the use of rigid, heavyweight robotic arms moving in high speed along with the fragile human life, can be proved dangerous to the latter, in various real-time scenarios, either due to software/hardware failure or having the human factor as the root cause. While onsite, hands on, safety training is important in decreasing significantly the risk of injury, virtual training through the use of state-of-the-art technologies, such as virtual, augmented or mixed reality (AR/VR/MR) [19], offers a vast amount of possibilities to get safely trained (onsite or remotely) on handling a variety of tasks (troubleshooting, maintenance etc.) in shared or collaborative working environments. The importance of these benefits can be illustrated by the fact that an increasing amount of industries have already decided to incorporate these technologies in their novice workers training. On the one hand, the nature of VR technology provides a complete virtual environment/test-bed, avoiding all the possible existing hazards that may occur during a live hands on demonstration in an industrial workplace, in real time. Moreover, it provides both trainers and trainees with a variety of tools (3D objects, effects) to enhance visualization thus understanding of complex, high level information, which may be difficult to achieve in hands-on educative courses. On the other hand, AR technology is able to combine the best of both worlds by enhancing the real world training in a shared manufacturing environment with visuals, hints, sounds and other additional information. As a result, the participants are able to access virtually enhanced educative material along with the important live experience during an educative course. Another important aspect is the data structure used for representing the virtual objects and information rendering for situational awareness. The variability of workspace configurations in industry creates the requirement of a flexible framework and data structure that is able to be adapted to specific workflows with minimal effort. Point clouds are recognized as a very promising means for representing 3D content in AR/VR; yet to obtain a high level of realism and faithfulness, a very high amount of data is needed, demanding efficient coding solutions. Octrees have been widely used for point clouds geometry coding and compression [20], since they are characterized by adaptive partitioning capabilities.

AR-based applications

This augmentation technique is already an integral part of multiple studies on human-robot collaboration. In one of such studies [21] a novel integrated mixed reality system for safety-aware HRC using deep learning and digital twin generation was proposed. Their approach can accurately measure the minimum safe distance in real-time and provide MR-based task assistance to the human operator. This technique integrates MR with safety-related monitoring by tracking the shared workplace and providing user-centric visualization through smart MR glasses for safe and effective HRC. A virtual robot, the digital twin of the respective live entity, is registered accurately to the real robot utilizing deep learning-based instance segmentation techniques. The main reason for using the digital twins is to calculate safety-related data more effectively and accurately and provide to the user some relevant safety and task assistance information. Another study [22] augments the real environment by rendering semi-transparent green and red cubes signifying the safe area and the constrained robot's work-space, respectively. Through visual alerts and warning messages about potential hazards at the shop floor, the system increases the robot operator's awareness. These alerts contain messages originating from the execution control system and may refer to robot movements and/or devices that operate in the cell and potentially can harm the human. A recent paper about safety in HRC [23] proposed the use of a depth-sensor based model for work-space monitoring and an interactive augmented reality User Interface (UI). The AR UI was implemented on two different hardware systems: a projector-mirror setup and a wearable AR gear equipment (Hololens). It is important to highlight the fact that Hololens-based AR is not yet suitable for industrial manufacturing while the projector-mirror

setup shows clear improvements in safety and work ergonomics, accordingly to the user experience assessment that took place. Last but not least, another interesting example of safety zones in the field of HRC is presented in [24]. The researchers created virtual boundaries around the area considered to be safe utilizing AR technology. When the worker approaches border, the border elements located most closely to the user starts growing and changing color such that elements grow and turn from green to yellow to red with decreasing distance.

VR-based applications

In comparison with the augmented one, it eliminates possible arising hazards that may occur in a real world scenario during training routines. To elaborate, a novice worker can be trained without worrying about safety issues related to his possible collaboration/interactions with the robot. However, the importance of safety zone awareness, during the training, is an integral part in every HRC. Regarding this, a recent VR application [25] was designed to simulate safety targeting strategies in human-robot collaborative manufacturing. The work reported two alternative safe collaboration strategies, to 'reduce speed' and 'move back', in the task of handling fabric when building aerospace composite material parts. Both methods are triggered when the distance of the human to the robot falls below a certain threshold. Virtual reality is also being employed in studies concerning cost-effective solutions, rather than just safety oriented ones [26]. In more detail, safety zone boundaries were created around the robot, which, when violated by the trainee, lead the program to automatically prompt a warning message explaining the operator's unsafe behavior. Another important safety training framework in HRC is introduced by [27], to empower safety concern and encourage constructing clear, distinguished instructions regarding machine operation. However, the visualization of safety zones is not restricted to only HRC applications. The researchers in [28] focused on the detection of potential collisions of a 3D moving vehicle model in an industrial environment, and visualized not only the safety zones around the desired entity but the vehicle path as well. Also, visual feedback was developed to illustrate the part of the moving object which collides with the environment.

5.1 Estimation of the working envelope of a robotic arm and visualization of the octree based robot risk mapping

For the estimation of the working envelope of a robotic arm, the knowledge of the exact possible trajectories and movement of the robot is required. As a working envelope, we assume the volume of the space in which a part of the robot could appear when it is moving while it performs a specific job. Additionally, the degree of freedom and the range of angles between the linked parts have to be known. Figure 34 shows different frames of a robotic arm while it performs a specific work, each point represents the exact position of the robot.

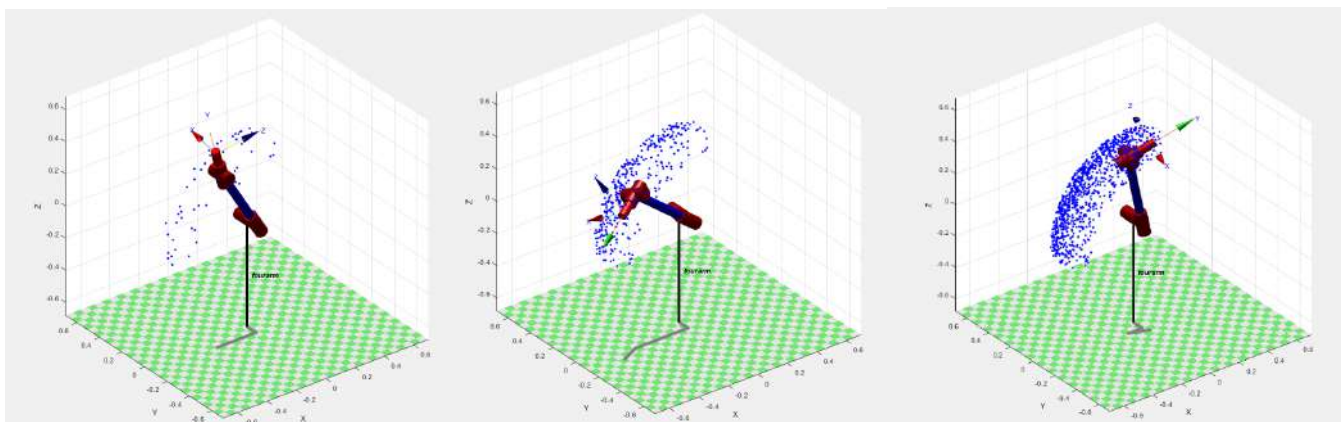


Figure 34: Collection of points while the robotic arm performs a pre-defined and planned work.

Figure 35 presents an example of a 3D working envelope consisting of 3D points.

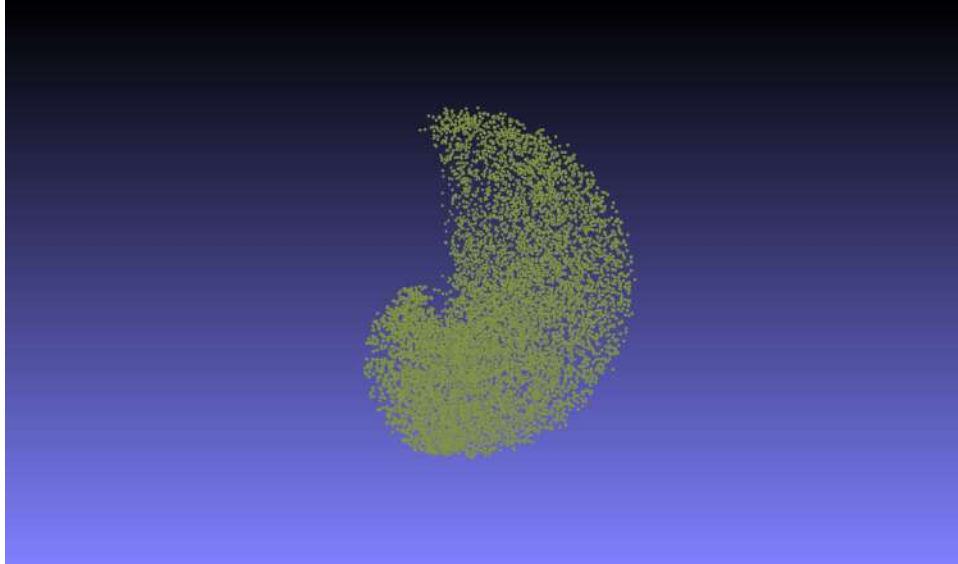


Figure 35: Example of a point cloud that defines the working envelope of a robotic arm.

The same point cloud, from a different view, is presented in Figure 36. The knowledge about the dimensions and the shape of the working envelope can be used to extract the safety zones. For example, in this figure we can define the xy area in which the robot can appear during its work.

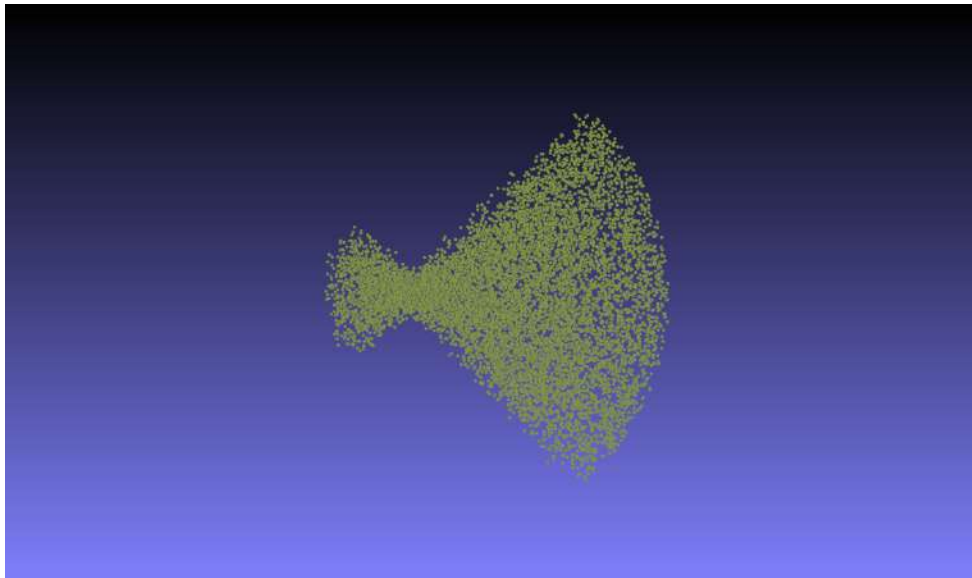


Figure 36: The same in a view on top (z axis) defining the xy covering space.

The point cloud representation can be used in order to reconstruct a 3D structure known as 3D mesh. Figure 37 and Figure 38 present the volumetric representation of the working envelope from two different field of views.

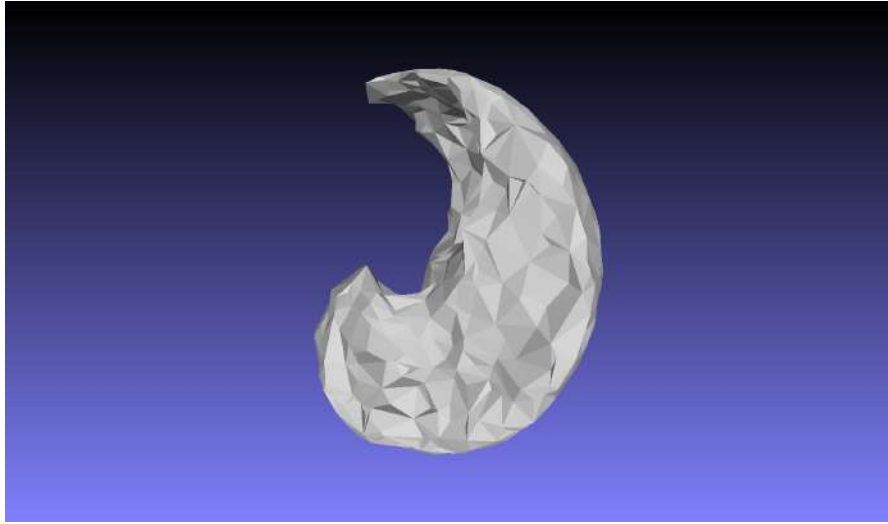


Figure 37: 3D volumetric representation of the working envelope.

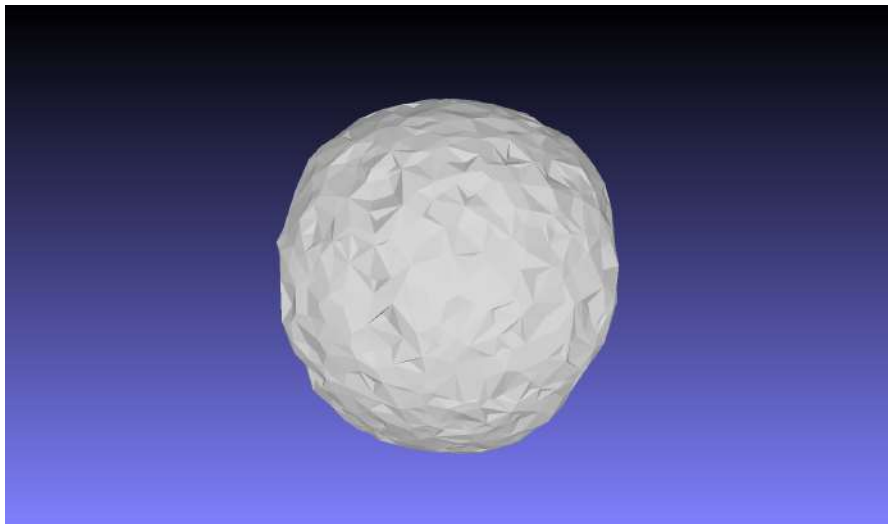


Figure 38: The same with above, in the field of view of an operator.

Octree based robot occupancy risk assessment library Documentation

Risk assessment occupancy mapping implementation for human-robot collaboration VR and AR applications

Dependencies (Required)

PCL 1.8
OctoMap
Eigen
JsonCpp
Optional
OpenCV - For scanning and projecting 2D occupancy map

Supported platforms

Currently only linux is supported. The generated data can be used on any platform that can parse JSON or

OctoMap files.

Building

Before proceeding with building make sure that all dependencies are installed.

```
mkdir build && cd build
cmake ..
make -j7
```

Usage / Prerequisites

- The programs expect specific robot describing configuration files and optionally trajectory files. Both are encoded in JSON files.
- The robot parts' bodies are PCL point cloud files. These can be generated from CAD models using PCL's `pcl_mesh2pcd` or `pcl_obj2pcd` programs. To be compliant with the sample configuration file create the folder `./data/kuka_kr_150r/` in the root directory of the repo and extract the content in it.

Generate occupancy map for a known robot trajectory

```
./generate \
-c ../sample/config_kuka_kr_150r.json \
-a ../sample/trajectory_kuka_kr_150r.json \
-o <output_file_prefix> \
[-r <min_voxel_size_in_meters>] \
[-d <tree_depth>] \
[-j <num_threads>] \
[-v]
```

Generate exhaustive occupancy map for a given joint rotation step

```
./generate \
-c ../sample/config_kuka_kr_150r.json \
-F \
-s <step_in_degrees> \
-o <output_file_prefix> \
[-r <min_voxel_size_in_meters>] \
[-d <tree_depth>] \
[-j <num_threads>] \
[-v]
```

Generate trajectory JSON file for exhaustive search

This is a useful pre-processing step if we want to generate an exhaustive occupancy map faster with multithreading. The output file can then be used as a regular trajectory file.

```
./exhaustive \
-c ../sample/config_kuka_kr_150r.json \
-o <output_file> \
[-s <num_steps_per_joint>] \
[-v]
```

Convert data octree to color octrees for each trajectory step

```
./colorize \
```

```
-i <input_octree.data> \
-a ../sample/trajectory_kuka_kr_150r.json \
-o <output_folder> \
[-m <risk_multiplier>]
[-r <min_voxel_size_in_meters>] \
[-d <tree_depth>] \
[-j <num_threads>] \
[-v]
```

Convert color octrees to JSON describing colored cubes in space

```
./color2json \
-i <input_folder> \
-o <output_folder> \
-a ../sample/trajectory_kuka_kr_150r.json \
[-m <risk_multiplier>] \
[-d <desired_depth>] \
[-j <num_threads>] \
[-v]
```

The desired depth that is passed determines the resulting cube/voxel size, that being $s=r*2^{(td-d)}$, where r is the octree resolution(min voxel size), td is the tree depth and d is the desired depth. For example, if we have a tree 16 levels deep with a resolution of 1cm and we set the desired depth to 13, then the resulting voxel size will be 8cm.

Load tree and access risk values

After a ConfigSpaceTree is generated (using the [generate](#) program) and stored in a binary file (*.ot) we can use it to get the occlusion risk for an AR/VR application.

Register the tree type in OctoMap and load the tree from a file:

```
ConfigSpaceTree dummy = ConfigSpaceTree(0.01);
std::string input("path\\to\\octree.ot");
AbstractOcTree* tree = AbstractOcTree::read(input);
if (!tree) {
    std::cout << "could not parse OcTree from " << input << std::endl;
    return 0;
}
if (!dynamic_cast<ConfigSpaceTree*>(tree)) {
    std::cout << input << " does not contain a ConfigSpaceTree" << std::endl;
    return 0;
}
ConfigSpaceTree* octree = dynamic_cast<ConfigSpaceTree*>(tree);
```

Get the risk of a specific node/location:

```
ConfigSpaceTreeNode* node = tree->search(x, y, z, depth); // depth determines resolution
double risk = 0;
```

```
// typically time_range = 5000(ms), multiplier = 3125000, ceil = 4
// t is current time since the start of the robot's movement, in milliseconds
if (node != NULL) risk = node->getCollisionTimeRiskFactorRekurs(t, time_range, multiplier, ceil)
```

Get a risk tree contacting the quantized risk level values in all space for a known animation:

```
// For better memory management create this once and reuse it
RiskLevelTree* riskTree = new RiskLevelTree(octree.getTreeDepth());
tree->getTimeRiskTree(timestamp, animationId, lookForwardMillis, lookBackMillis, multiplier, ceil, riskTree);
```

Get a risk tree contacting the quantized risk level values in all space calculating the most probable animation:

```
// For better memory management create this once and reuse it
RiskLevelTree* riskTree = new RiskLevelTree(octree.getTreeDepth());
tree->getTimeRiskTree(frame, frame, lookForwardMillis, lookBackMillis, multiplier, ceil, riskTree);
```

Access all nodes of RiskLevelTree:

```
tree->getTimeRiskTree(t, riskTree, time_range, multiplier, ceil);
for (auto it = riskTree->begin_leafs(); it != riskTree->end_leafs(); it++){
    // Node center coordinates and size. Members of iterator object, accessed with "." operator
    double x = it.getX();
    double y = it.getY();
    double z = it.getZ();
    double s = it.getSize() // the side of the volume occupied by the current node
    // Member of pointer(iterator is pointer to node), accessed with "->" operator
    int nodeRiskLevel = it->riskLevel;
    /* Do something here, e.g. render a cube */
}
```

Class List

Table 1 presents the classes, structs, unions and interfaces with brief descriptions.

Table 1: List of classes, struct and interfaces.

Animation	A class containing a series of frames describing a robot's movement
Component	A class describing the component of a robot, that usually being Link or a Joint
Frame	A class containing information about a robot's configuration at a given time
Joint	A Component functioning as a Joint in a Robot chain
JsonAnimation	An Animation that is parsed from JSON
JsonFrame	A Frame that is parsed from JSON
JsonJoint	A Joint that is parsed from JSON
JsonLink	A Link that is parsed from JSON
JsonRobot	A Robot that is parsed from JSON
Link	A Component functioning as a Link in a Robot chain
Robot	A Component functioning as a Robot chain
Thread	An object-oriented thread implementation to use for parallel execution
ThreadPool	An object-oriented thread pool implementation to use for parallel execution

ConfigSpaceTree	Octree class containing data about robot configurations
StaticMemberInitializer	
ConfigSpaceTreeNode	Octree node containing robot configuration data generated by simulation
ConfigurationSnapshot	A class containing a snapshot of a the configuration of a Robot
ConvexPoly	A class describing convex polygon that is used to generate an octree occlusion slices and projection map
indexed_point2_t	A helper struct used to sort the 2D points in a clockwise fashion
LineSegment	A class describing a finite line segment in 3D space
Plane	A class describing a 2D plane in 3D space that is used to generate an octree occlusion slices and projection map
Preference	A helper class to define a preference that can be provided using terminal input arguments
PreferenceParser	A helper class to parse input arguments and covert them to program parameters
RiskLevelTree	Octree class containing occlusion risk level data
StaticMemberInitializer	
RiskLevelTreeNode	Octree node containing occlusion risk level data
SafeQueue	A thread safe queue implementation
Slice	A class describing a 2D slice of a ConfigSpaceTree
Slicer	A class that slices a ConfigSpaceTree into 2D slices that are parallel to a slicing Plane
VoxelCube	A class describing a cube, used to represent a voxel in the octree structure
Animation	A class containing a series of frames describing a robot's movement
Component	A class describing the component of a robot, that usually being Link or a Joint
Frame	A class containing information about a robot's configuration at a given time
Joint	A Component functioning as a Joint in a Robot chain
JsonAnimation	An Animation that is parsed from JSON
JsonFrame	A Frame that is parsed from JSON
JsonJoint	A Joint that is parsed from JSON
JsonLink	A Link that is parsed from JSON
JsonRobot	A Robot that is parsed from JSON
Link	A Component functioning as a Link in a Robot chain
Robot	A Component functioning as a Robot chain
Thread	An object-oriented thread implementation to use for parallel execution
ThreadPool	An object-oriented thread pool implementation to use for parallel execution
ConfigSpaceTree	Octree class containing data about robot configurations
StaticMemberInitializer	
ConfigSpaceTreeNode	Octree node containing robot configuration data generated by simulation
ConfigurationSnapshot	A class containing a snapshot of a the configuration of a Robot
ConvexPoly	A class describing convex polygon that is used to generate an octree occlusion slices and projection map
indexed_point2_t	A helper struct used to sort the 2D points in a clockwise fashion
LineSegment	A class describing a finite line segment in 3D space

File List

Table 2 is a list of all documented files with brief descriptions.

Table 2 List of documented files.

3d	
3d.h	
ConvexPoly.hpp	
LineSegment.hpp	
Plane.hpp	
Superquadric.hpp	
VoxelCube.hpp	
build	
ConfigSpaceTree_Export.h	
config_space_tree	
ConfigSpaceTree.hpp	
DLLDefines.h	
RiskLevelTree.hpp	
utils.h	Useful utilities for extracting information from ConfigSpaceTree
parallel	
SafeQueue.hpp	
Thread.hpp	
ThreadPool.hpp	
robot	
Animation.hpp	
Component.hpp	
ComponentType.hpp	
ConfigurationSnapshot.hpp	
Joint.hpp	
JointType.hpp	
JsonAnimation.hpp	
JsonJoint.hpp	
JsonLink.hpp	
JsonRobot.hpp	
Link.hpp	
robot.h	
Robot.hpp	
slicer	
Slice.hpp	
Slicer.hpp	
color2json.cpp	A program to encode the content of a ColorOcTree into a JSON text file. This file can then be parsed without requiring a dependency to OctoMap.
colorize.cpp	A program to convert a ConfigSpaceTree to a ColorOcTree using an time-based occlusion risk factor. The output ColorOcTree can then converted to JSON or parsed and visualized using OctoMap tools like Octovis
conversions.hpp	
exhaustive.cpp	A program to generate an animation JSON file containing all the configurations of a robot for an exhaustive simulation
generate.cpp	A program to generate a ConfigSpaceTree for a robot
InputSimulator.hpp	
Preference.hpp	
PreferenceParser.hpp	
utilities.hpp	Simple utilities

5.2 Warnings for the reconfiguration of the windshield based on pose recognition

The main objective of the developed manufacturing simulator is the implementation of an immersive VR/AR system that can be used in many CPSoSaware use cases, such as for increasing the operator's situational awareness during human-robot cooperation in real-time manufacturing conditions, for the safe training of new operators in an industrial environment, for ergonomics analysis of the operator under different actions during his/her collaboration with a robot etc. More details about the simulator, and how it is used, will be discussed and reported in deliverables (more details have been discussed in D2.5)

Regarding the industrial environment, an industrial KUKA robot performs assembly operations in a shared workspace with a human operator. More specifically, the developed simulator emulates a robotic movement and the corresponding operator's actions in a digital workspace in Unity3D. When an operator is physically interacting with a robot, trying to accomplish a task, his/her posture is inevitably influenced by the robot's movement. A system able to be adapted according to human anthropometrics is dependent on several components. Firstly, the system has to receive some input from the operator in order to be able to adapt. Based on the ergonomic information a decision-making module should find the right set of actions in order to minimize strain and

ergonomic risk factors based on a robot response model. Finally, a robot control module should send control information to the robot such that the wanted set of actions is carried out [29]. More details have been discussed in D3.1 for the pose recognition system and in small scale trial in D6.2.

This task focuses on exploring computer vision and corresponding warnings and visual information for ergonomic assessment with the purpose of monitoring the operator and adapting the position of an assistive collaborative robot in order to improve the working environment for the human, by improving the operator's posture in the work environment. The reference scenario is related to the safety aspects and two steps are followed (i) an initial estimation of the operator's height and operator's anthropometrics classification. (ii) Adaptation of the robot's position and relative trajectories according to predefined scenarios based on the operator's classification.

An RGB camera is used to monitor the human's actions and then a pose estimation algorithm is running to extract the posture landmarks and also to calculate the current anthropometric state, in real-time. A total of 3 different classifications are used based on the operators' height, leading to 3 adaptable robot responses (adaptation scenarios) to allow the human to work in an optimal ergonomics state.

The algorithmic framework identifies the operators' class, based on their height, and the corresponding selected scenario configures the cooperating robot's movement parameters to adapt to the environment in a way that is ergonomically most comfortable for the interacting user. To mention here that it is required not to perform an active real-time adaptation of the robot's trajectories during the use-case phases; the robot's trajectories should be pre-planned and validated in VR, but not reactively adapted during the Human-Robot interaction. Once the system recognizes the operator's anthropometrics, it identifies the proper final position of the gripper in the safety zones and picks up the pre-planned trajectory to reach the final position. The aim is to avoid programming the robot with collision avoidance techniques that might be necessary in case of adaptive trajectories.

We assume the existence of 3 different classes based on the operator's height. More specifically, class 1 consists of operators with height < 175 cm, in class 2 the operators' height is between 175 and 185 cm, and finally in class 3 the operators are taller than 185 cm, as presented in Figure 39.

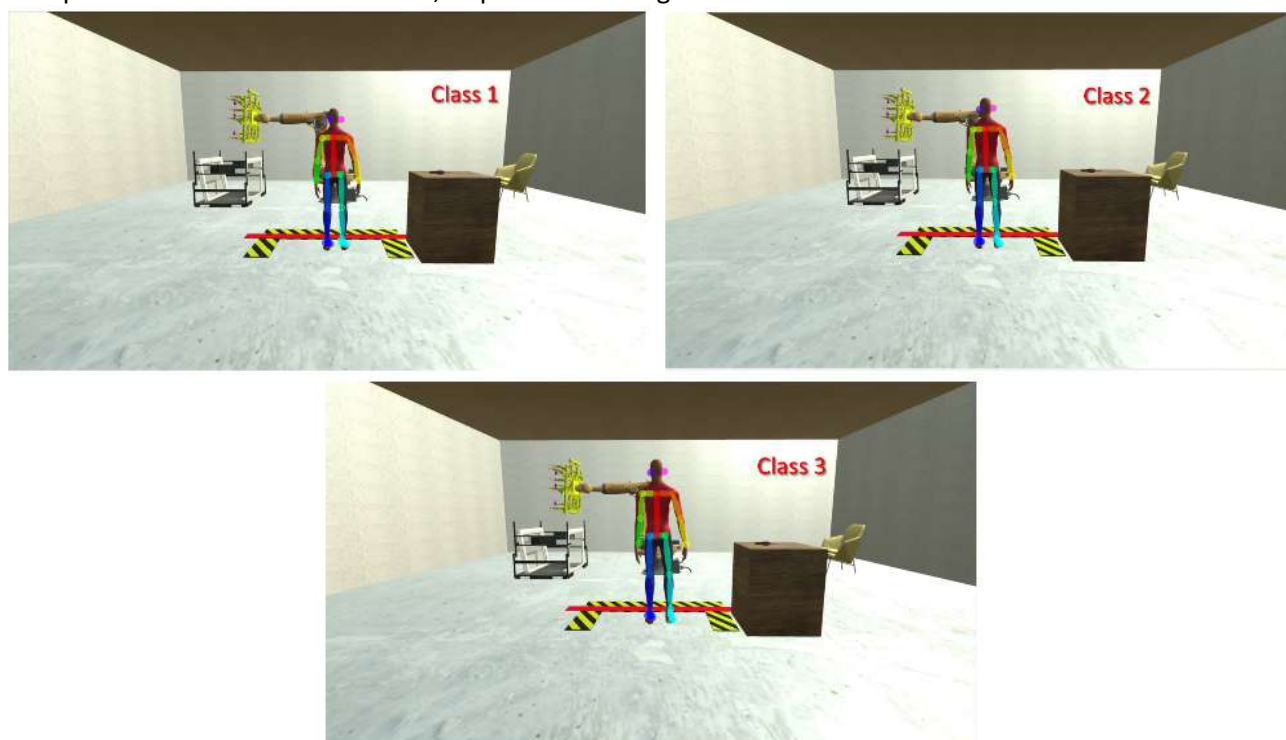


Figure 39: Pose estimation and class estimation for three different operators with different heights.

5.3 Warnings based on operator's proximity with the robot and visualization of safety zones

Recent hybrid manufacturing paradigms where humans and robots operate in a shared environment impose new challenges for safeguarding the operator's safety and robot's integrity. Standard solutions relying on isolating the workspace of robots from human access during their operation are inefficient and impractical. This section presents an extended reality-based method to increase human cognitive awareness of the robot's dynamic occupancy space for enabling safe human-robot collaborative manufacturing operations. Specifically, a dynamic and state-aware occupancy probability map, indicating the forthcoming risk of human-robot accidental collision in the 3D workspace of the robot, is calculated using octrees and rendered in an augmented or virtual environment using Unity 3D. Our framework allows to generate both static zones (taking into consideration the entire configuration space of the robot) and dynamic zones (generated in real time by fetching the occupancy data corresponding to the robot's current configuration), that can be potentially utilized for improved short term collision risk prediction.

An immersive VR or AR system for safety-aware human-robot collaboration (HRC) is presented. The work is based on the definition of a simple robot configuration description format and the use of an octree structure that provides the flexibility to simulate a workspace at various resolutions, depending on the available computational power, while at the same time maintaining the ability to store rich information for high dimensional workspaces. We propose a solution to efficiently render safety zones in Hololens2 AR equipment by combining the Unity3D game engine and occupancy mapping simulations. Our contribution is threefold:

- We are emulating a robotic movement in a digital workspace in Unity3D to be used as a prior knowledge during runtime computation of (spatiotemporal) human-robot collision risk.
- We are introducing the use of octree-based representation in relation to machinery movement to calculate dangerous areas around the robot, by utilizing information about the configuration state of the robot.
- We proceeded on experimenting with different rendering methods in Unity in VR/AR for static and dynamic safety zones.

We developed an immersive AR system for increasing situational awareness during human-robot collaboration in manufacturing. The method is illustrated in the case of an industrial robot (such as the Kuka KR 150R 2700) performing assembly operations in a shared - with a human operator - environment. The system evaluates and visualizes the dynamic risk of collision while the robot is in motion. This is achieved in real-time with the use of occupancy probability maps, pre-calculated through robotic motion simulations. In order to safeguard the method's generalization and applicability, robotic motion simulations are not restricted to predefined robotic trajectories but rather span the whole configuration space, calculated based on the robots' kinematic model. Realistic 3D rendering is achieved thanks to Unity's real-time 3D technology. The occupancy mapping is computed through the use of octrees for efficient data representation and storage, with high and adaptive sampling density capabilities. It is illustrated in the virtual environment (using Unity3D) through the use of helper cubes for a simplified implementation. The methodological components are illustrated in the schematic diagram of Figure 40 and described with more details in the following sections.

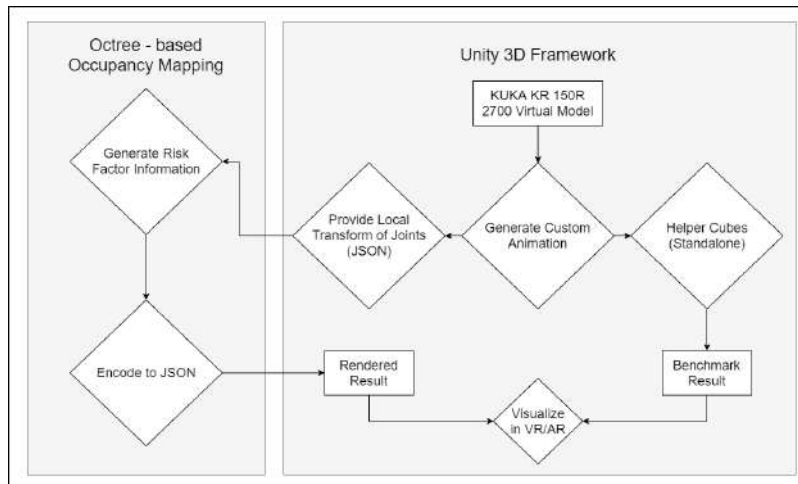


Figure 40: Schematic diagram of the AR-based system for safety-aware human-robot collaboration.

5.3.1 Robot modeling

The robot is represented as a list of linked components (Figure 41). Each component is either a link or a joint. All components have a base transformation matrix representing the transformation between the point of connection to the previous and the next component at their initial position. To test our method we used a 6-DoF robotic arm (a robotic arm with 6 degrees of freedom). The particular arm that we use, is the KUKA KR 150R 2700 model. This arm has a typical design of a 6-DoF robotic arm and consists of 6 revolute joints. The 3 joints closer to the base of the robot control the position of its end effector. The workspace of the robot is determined by the dimensions of the links of the robot and the rotation limits of the first 3 joints. As most of robotic arms in this category, the last 3 axes closer to the end effector of the robot intersect so that the end effector can successfully be rotated in any axis in the 3D space. The joint components have a modifiable value introducing one degree of freedom each. The modifiable value is used to calculate the component's final transformation matrix. The current implementation only allows for rotational joints. The body of each component is modelled as a point cloud. The point cloud is generated from CAD models of the robot using ray tracing operations implemented with the point cloud library PCL [30]. By applying the component transformations to the points we can simulate 3D space occupancy for any given configuration. The configuration of the robot is defined as a JSON array. Each JSON object represents a component and defines the type, size, rotational axis and value range of each component. The robotic movement is assumed to follow a number of predefined trajectories (selected by the user). By increasing this number, all possible trajectories and configurations can be simulated. A trajectory is defined as a sequence of poses in the configuration space.

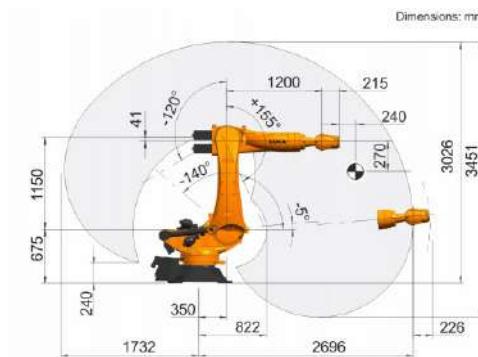


Figure 41: Dimensions of Kuka KR 150R 2700

5.3.2 Collision Risk Prediction

In order to calculate and render the dangerous areas around the robot we need to store information about the possible locations that the robot can occupy and accordingly define a collision risk metric. In a similar work [31], this is performed by randomly sampling points within the robot's operating space and generating a collision risk factor for a given trajectory. The drawback of this approach is that the risk factor is only calculated once and does not include temporal information. In this work, we present three ways to calculate collision risk that are based on increasing level of available information. Given the robot's initial configuration, the simplest approach for occupancy risk prediction is to consider all possible positions the robot can reach at any time point. In this case, temporal information is not considered and the calculated 3D probabilistic map of the unsafe area is static, thus can be pre-calculated in advance (offline) given the robot's kinematic constraints. The exhaustive scanning of all possible configurations is performed by recursively applying a step modification to each joint's value and calculating the position of all components at each iteration. The step is a significant contributing factor to the total computational time of the full scan and can be modified depending on the accuracy requirements. In the more elaborated approach we consider the configuration of the robot at a given time point and calculate the occupancy risk only for the next few time steps. In this case the 3D occupancy probability map is dynamic, as it follows the movement of the robot. Parallelization in the calculation of the occupancy map for every possible initial configuration allows to optimize the computational time. While the previous static or dynamic probabilistic occupancy maps provide a measure of collision risk, they weigh equally each one of the potential directions of movement of the robotic arm. In order to improve short-term prediction, in the third approach we take into account (in addition to the robot's pose) also the angular velocity of each robotic joint at each time point. This provides a state-aware collision risk with directionality, obtained by penalizing changes in the rotation of the robotic joints, and thereby improving estimation in the immediate future.

5.3.2.1 Static occupancy map

A safety zone is extracted using the occupancy maps by projecting them onto a 2D plane parallel to the ground. Starting with the plane $y = h_{max}$, where h_{max} is the maximum y value of any occupied node's bounding box in the map the octree is progressively sliced with a step dy . The scanning finishes after the plane $y = h_{min}$ is reached, where h_{min} is the minimum y value of any occupied node in the map. On each interval we calculate the intersections between the current plane and the occupied node bounding boxes. The enclosing area of the intersection polygons are considered occupied areas. The finite 2D plane containing all the polygons is divided into discrete equal rectangular areas and the final occupied area is calculated by projecting each polygon onto the discrete plane. Then all polygon vertices that are at the border between the occupied and non-occupied area, are used to construct a final polygon P that encloses the entire occupied area. While P is representative of the safety zone shape, it cannot be directly used as one, because it offers a very small error tolerance and entering the zone would translate to almost certain collisions. To tackle this we must introduce a buffer distance between the occupied area and the border of the safety zone. This can be achieved in multiple ways, by dilating the occupied area of the discrete plane moving the border vertices outwards for the center by a constant factor, or simply generating the area using an octree of lower resolution, although the last two methods inhibit the risk of collision if the robot's body is close to the edge of the outer octree nodes. This expanded polygon P_e can then be used to render a semi-transparent virtual wall to indicate risk of entering a zone close to the robot. The safety zone that is calculated using the method above is similar to the safety fence installed in the shared work space, as it aims on preventing physical access in the area surrounding the robot. The advantage of this virtual fence is that an experience human worker maintains the ability to get close the robot when considered safe. On the other hand, the virtual safety zone maintains the inflexibility of a fence installation, as it does not adapt to changes in robot and human positions and movements.

5.3.2.2 Dynamic occupancy map

The area described in the previous section can be calculated for the entire configuration space, a predefined trajectory, or for individual configurations. In the case of the full configuration space map, the resulting zone is a static area regardless of the current and expected future positions of the robot. This unnecessarily restricts movement into a large area of the workspace when the robot moves predictably. To tackle this, we propose the use of dynamic safety zones. The dynamic safety zone illustrates the potential position of the robot depending on its current pose and movement, providing the operator with critical information about the workspace areas that can be safely accessed in the next time points.

We quantize the configuration space with a constant step or a constant number of steps per configuration space dimension. Then we calculate the occupancy map and safety zones for each value in the discrete configuration space, or trajectory step. Using a mapping function, we assign a key to each map and save it in storage. Depending on the type of forward simulations, that being either exhaustive or following a specific trajectory, we map the configurations and define their distance using the coordinates of the quantized configuration space, or the forward time difference, correspondingly.



Figure 42: Current robot pose

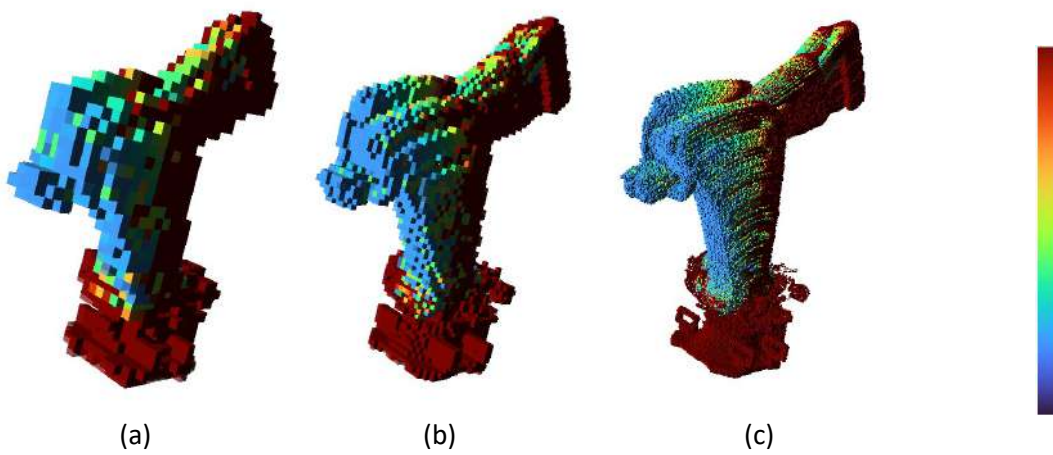


Figure 43:(a) 8cm voxel size, (b) 4cm voxel size, (c) 1cm voxel size.

Dynamic collision risk map with look-ahead time T visualized (for predefined trajectory of the robot) at various resolutions (voxel sizes) thanks to the use of octrees (

(a) (b) (c)

Figure 43). The risk factor is scaled by a constant factor R_s to compensate for the squared time difference in the denominator of Eq. (14) and illustrated in a blue (0) to red (1) colorscale.

In the first case we use the l_1 norm:

$$d(p, q) = \sum_{i=1}^n |p_i - q_i| \quad (11)$$

where p and q are two points in the n -dimensional configuration space. Given a step s , the values p_i , $i \in [1, n]$ of the point p represent the interval steps from the i^{th} degree of freedom lowest possible value required to reach this position, thus $p_i \in N^+$. Since the values of the configuration point are positive integers, given a segmentation of each degree-of-freedom's range to 255 values, we can represent all potential values of a joint using 8 bits of data. Going further, assuming a robot of maximum 8 DoF we represent each point in the configuration space as a 64-bit number, allowing for efficient storage of said configuration points in the occupancy map nodes. A collision risk factor can then be defined as:

$$R_k(x, y, z, c) = R_s \sum_{i=1, d \leq k}^l \frac{1}{d^2(p_{xyz}^i, c)} \quad (12)$$

where $d(p_{xyz}^i, c)$ is the distance between the configuration point c and the i^{th} configuration point p_{xyz}^i that lead to occupancy of the node corresponding to the coordinates (x, y, z) of the world, as defined in Eq. (11); k is the configuration distance lookup depth; l is the number of configurations that occupy the node corresponding to (x, y, z) ; R_s is a constant scaling factor.

In the case of simulating a known (expected) movement of the robot, we use the time t since the start of the movement (in milliseconds) as the representative key of the configuration. Using the key t we simply define the distance from t to t_0 as:

$$d(t, t_0) = \begin{cases} t_0 - t, & \text{if } t_0 \geq t \\ \infty, & \text{otherwise} \end{cases} \quad (13)$$

Using the time distance, the risk factor is then defined as:

$$R_T(x, y, z, t) = R_s \sum_{i=1, d \leq T}^l \frac{1}{d^2(t, \tau_i)} \quad (14)$$

where $d(t, \tau_i)$ is the timed distance between the current time t and the i^{th} time $\tau_i > t$ when the node corresponding to the coordinates (x, y, z) of the world is occupied, as defined in Eq. (13); T is the future lookup depth; l is the number of times the the node corresponding to (x, y, z) is occupied after t ; R_s is a constant scaling factor. At runtime, we apply the same quantization and mapping function to the current configuration of the robot and get the key to the corresponding occupancy map and safety zone. Additionally, we can fetch the neighboring configurations, within a predefined configuration distance and merge the safety zones, forming a map that corresponds to the robot's current and potential future positions. Using this risk factor we can generate a 3D map of the collision risk for a given configuration.

State-aware short term prediction of collision risk

To be able to estimate the collision risk between a robot and a human we need to have knowledge about the

robot's workspace. Given its workspace, a human user can effectively avoid the robot while working in the same area. Based on the time frame that we use, the workspace can be divided into two separate workspaces from which the one is subset of the other.

The superset is the kinematic-prior workspace of the Kuka robotic arm that can be acquired by the Denavit-Hartenberg parameters of the robot. The kinematic-prior workspace W_k contains all the possible 3D positions that the end-effector of the robotic arm is able to reach. It is calculated once and is only based on the mechanical characteristics of the robot.

The other workspace is the workspace created by an estimation of the robotic's arm future position and we will be referring to as W_e . This estimation takes into account sequential poses of the robot taken with a predetermined time step. Thus, in each time step we get the pose of the robot from which we can determine the individual velocities of each part of the robot.

A 6-DoF robotic arm consists of multiple coordinate systems. Each coordinate system corresponds to a different joint of the robot. For each sequential pair of coordinates systems the equation connecting the velocity of one to the other can be retrieved by the following equation. Considering the coordinate system B moves with respect to coordinate system A with a velocity of ${}^A U_{B0}$, the velocity ${}^A U_p$ of each point p along the link that connects only with the joint corresponding to the coordinate system B can be calculated as follows:

$${}^A U_p = {}^A U_{B0} + {}^A R_B {}^B U_p \quad (15)$$

or

$${}^A U_p = {}^A R_B {}^A U_{B0} + {}^A \omega_B \times {}^A R_B {}^B p \quad (16)$$

where ${}^A R_B$ is the rotation matrix between the two coordinate systems and ω is the angular velocity of each joint. From the aforementioned equations one can derive the equations describing the linear and angular velocity of each link of the robot. Given the joint $i+1$ moves in respect to joint i

$${}^0 \omega_{i+1} = {}^0 \omega_i + {}^0 k_{i+1} \dot{\theta}_{i+1} \quad (17)$$

where ${}^0 k_{i+1}$ is the unit vector of the coordinate system of joint $i+1$ with respect to the z-axis referring to the base system 0.

$${}^0 U_{i+1} = {}^0 \omega_i + {}^0 \omega_i \times {}^0 \pi_{i+1} \quad (18)$$

The knowledge of the velocities of each part of the robotic arm along with the general pose of the robotic arm at an exact moment are enough to construct the W_e of the robot. W_e along with the W_k and suitable buffer zones around them can divide the space into different zones based on collision risk.

W_e and hence the risk factor zones are updated for each new pose. Given the new pose q_{t+1} and the previous pose q_t one can derive the angular velocity of each joint. Our method creates rotation ranges D_i for each of the i joints, based on their angular velocity ω_i in that exact time. We make the assumption that if a joint rotates in one direction, then it is most likely to keep rotating in the same direction. If the joint keeps rotating towards the same direction, then we assume that it cannot reach a velocity ω_{i+1} 50% higher than its previous one ω_{it} . In the event of changing direction, the joint will be expected to be rotated in a smaller angle range, equal to the half of the previous one. Given that ϕ_i is the angle difference of joint i between two sequential states then the rotation range for this joint can be described by the following equation:

$$D_{i+1} = \begin{cases} [l_i \leq \theta_{i+1} - \frac{\phi_{i+1}}{2}, \theta_{i+1} + \phi_{i+1} \leq u_i], & \phi_{i+1} \leq 0 \\ [l_i \leq \theta_{i+1} - \phi_{i+1}, \theta_{i+1} + \frac{\phi_{i+1}}{2} \leq u_i], & \phi_{i+1} > 0 \end{cases} \quad (19)$$

where l_i and u_i are the lower and upper angle limits of joint i respectively.

This way, six different rotation ranges are created. One for each joint of the robot. Based on these rotation ranges we can estimate the next position of not only the end-effector of the robot, but of any point across any link of the robot. Since by nature the coordinate systems of the 3 last joints of the robot intersect with each other this means that their rotations will not affect significantly the space the robotic arm takes up. Thus, only the rotation ranges of the first three joints will be used for the risk factor analysis. As shown in Figure 44 the risk factor analysis at this stages consists of three 3D point sets, each corresponding to the estimation of the position of the end of each of the first three links of the robot.

These point sets form the superset V which can then be used to form a state-aware collision risk factor. We consider a 3D space C of $b \times b \times b$ dimensions in which the risk factor will be computed. Given that d_{sa} is the distance between a point $g \in C$ and a point $j \in V$.

$$d_{sa}(g, j) = |g - j| \quad (20)$$

To properly describe the risk factor, the j closer to the g has to be found in order to be able to compute the state-aware risk factor of g_{xyz} . The metric of the risk factor will be the distance between the two. Thus the risk factor will be inversely proportional to the minimum distance $d_{min}(x, y, z)$ between $g(x, y, z)$ and any point $j \in V$.

$$d_{min}(x, y, z) = \min_{j \in V} (d_{sa}(g_{xyz}, j)) \quad (21)$$

Then the normalized state-aware risk factor R_{sa} of a 3D point g_{xyz} can be described by the following equation:

$$R_{sa}(x, y, z) = \begin{cases} \frac{b}{d_{min}(x, y, z)}, & d_{min}(x, y, z) \neq 0 \\ 1, & d_{min}(x, y, z) = 0 \end{cases} \quad (22)$$

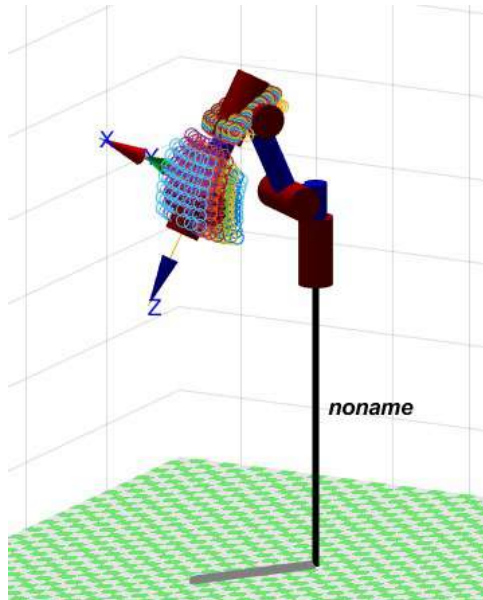


Figure 44: State-aware collision risk factor point sets.

5.3.2.3 Occupancy mapping by octrees

In order to encode the probabilistic 3D occupancy map in a more compact and flexible way, we used an octree structure based on OctoMap [32]. The octree structure allows to easily adapt the 3D space resolution depending on the application requirements. Starting with one root cube enclosing the entire area of interest, each node of the octree structure branches to 8 sub-cubes (voxels) of the same size, until the desired space resolution is reached. At the same time occupancy information for the entire configuration space can be efficiently stored in a small size file. Each node encodes the occupancy probability, number of configurations that lead to the node being occupied and a list of the configurations that lead to node occupancy.

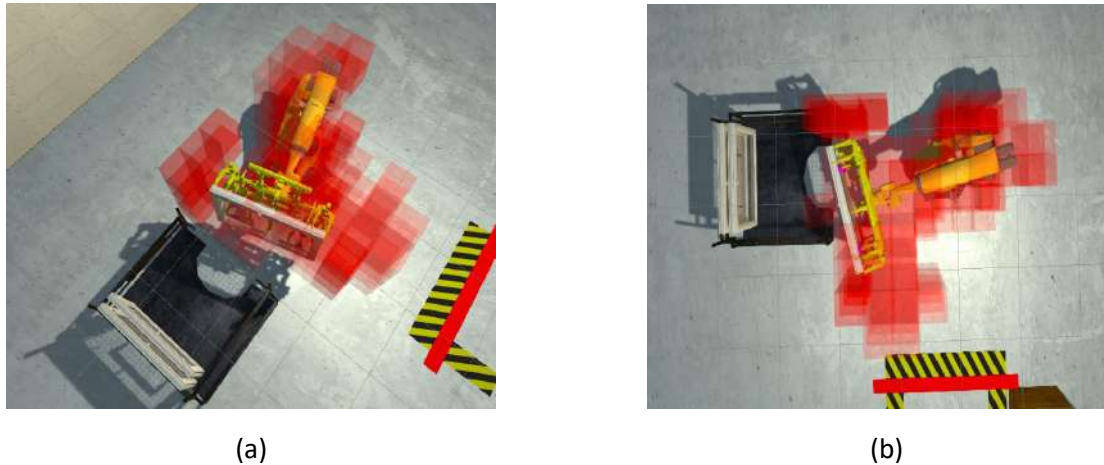


Figure 45: Dynamic safety zone construction with cube representation, (a) Perspective View, (b) On top View.

The occupancy map is generated either using the trajectory or full configuration space simulations. The base of the robot is positioned at the world origin. For each configuration of interest, we calculate the robot's surface point cloud and cast rays from the origin to the points. Since a single node can enclose multiple points, we take care to only count one hit per configuration for each node. This way the occupancy mapping is independent of density anomalies in the provided point clouds. The final result is a 3D occupancy probability map for the desired trajectory, or the entire configuration space. Additionally, we generate an independent occupancy map for each individual configuration of the set. Figure 45 shows an example of a dynamic volumetric collision risk factor computed for a given pose of the robot (shown in Figure 42) and illustrating the risk for the upcoming time period T . With the use of octrees in data representation, different resolution levels can be obtained according to the required level of visualization accuracy and memory limits.

5.3.2.4 AR/VR-based Visualization

Visualization of the simulated environment and rendering of the safety zones is performed with the Unity 3D game engine. The respective virtual industrial environment (Figure 46) is created using a variety of available 3D objects. The required functionality was achieved by C# scripts. For the visualization in the virtual world, Oculus Rift head-mounted display is employed, while the navigation of the user around the environment and the interaction with objects is rendered possible by the use of the Oculus controllers. In AR, HoloLens 2 equipment is utilized. The virtual environment of the current application is illustrated in Figure 46.

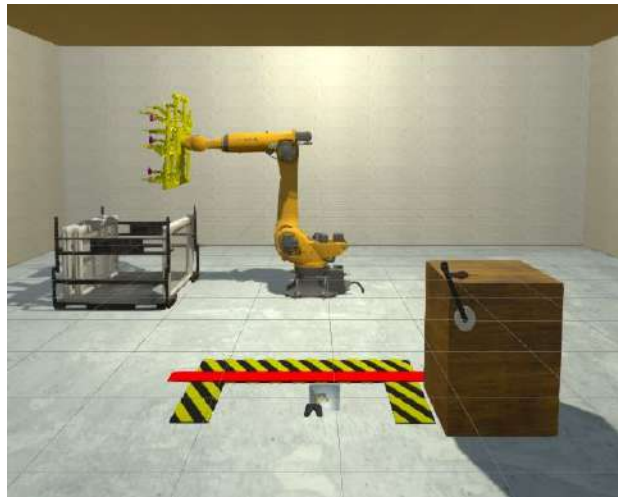


Figure 46: The virtual environment.

Occupancy mapping in Unity3D (Standalone Method)

As described earlier, the occupancy mapping is computed either using an octree-based representation (as explained above), or with a standalone method involving the helper cubes in Unity, being responsible for providing a benchmark outcome. This second approach is described below.

The initial step for the standalone method is to fill the space around the 3D robotic model with a large number ($n=715$) of small cubes (Figure 47), in which everyone has a unique id in order to be identifiable. Their position in the virtual world will be used later for the construction of the safety zones. Next, a custom animation for the 3D robot was designed which mimics a basic assembly operation. The role of the cubes will be demonstrated in the next step where these primitive shapes will detect the collisions with each part of the robot during its movement and register that information to enable the rendering procedure. This detection takes place at predefined time steps (a default value of 0.1 second between each record was selected). By computing which cubes are colliding with the robot's parts in every designed time step, a set of points is being generated, which, eventually sets the safety boundaries in every moment. That information is saved into JSON format, where each step has been registered through registering the overall set of the unique IDs and a respective boolean value for each one which specifies whether there was a collision in each step. Through the above pre-processing procedure, a set of JSON files is created, which will contain the needed information that will be used in the visualization of the safety zones.



Figure 47: The helper generated cubes

Using the same algorithm through which the helper cubes were generated, we instantiate empty Unity gameobjects (helper points) on the same positions as the cubes. The first registration in the JSON files will be referring to the first generated helper point and so on. In this way, we accomplish the matching between the helper points and the cubes. Through this matching, we can obtain the information of which helper points should be used in order for the safety zones to be rendered. This can be done by obtaining the second field of each registration within the JSON file. As mentioned above, this field shows which cubes collided with the robot's parts in each step of the predefined trajectory. The information that comes from the JSON files is read, obtained and stored in lists during the run time. For performance purposes, these lists are sorted, so that the registrations which match to the colliding cubes will be first. Here, the first part of the computations of the safety zones comes to an end. To incorporate both the static and dynamic zone functionality of the introduced octree-based representation, we deployed respective scenarios in both VR and AR environment, while also experimenting with different shapes/entities, i.e using surface model cubes and fog. So, for every visualization, there are specific categorizations:

- **Static Safety Zone Construction:** Here, we proceed on rendering the accumulated information of the overall animation regarding all the possible positions where the robotic arm may cross.
- **Dynamic Safety Zone Construction:** Here, we proceed on rendering a specific number of upcoming steps instead of computing every possible movement of the robot from start to end.

Both procedures displayed specific benefits which will be analyzed below. Another important factor lies on the shapes/models that were chosen for the rendering method. In our approach, we chose to use *cube* models and *fog* (by employing Unity's particle system) as different methods, for both VR and AR applications.

Algorithm 1 Helper Cubes Occupancy Mapping

```
1: Fill space around robot with desired number (n) of cubes
2: for Every time step do
3:   for Every cube do
4:     Calculate if cube collides with robot.
5:     Write to JSON
6:   end for
7:   Sort True/False IDs
8: end for
9: Save JSON file
10: Read JSON file
11: for Every time step do
12:   while Cube collides with robot do
13:     Render Cube
14:     Proceed to next Cube
15:   end while
16: end for
```

5.3.2.5 Rendering in VR

In Virtual Reality, initially, we proceeded on visualizing the static safety zones with the different rendering methods. Cube representation was first. It is clear that there is a distinction regarding the color of each cube,

which is correlated to the amount of times the robotic arm collides with the provided cube in total, considering all the available animations. Red color was related to the most dangerous position in relation with the robotic movement, while brighter ones were signaling safer areas. From the Figure 48, it is clear that poor Signal To Noise Ratio (SNR) was present, as the amount of cubes being rendered contribute to an overwhelming result.

The same distinction takes place in the fog representation (Figure 49) too, where the red colors signals the most dangerous zones, in contrast to the blue one, providing information about the minimum likelihood of contact. However, neither this method displayed significant improvement with regard to the previous implementation, although the SNR is higher.

Another important factor is that due to VR's nature, for each frame the workload is doubled as the image needs to be rendered twice (one for each eye) causing a significant drop to the overall performance (frames per second - fps), thus causing critical problems related to the overall experience and immersive capability of the simulation.

Due to this combination, the need of reduced information with higher SNR was imminent in order to be able to provide any useful VR/AR experience, as every AR, resource-restricted with regard to a desktop, equipment will experience the same kind of performance problems.

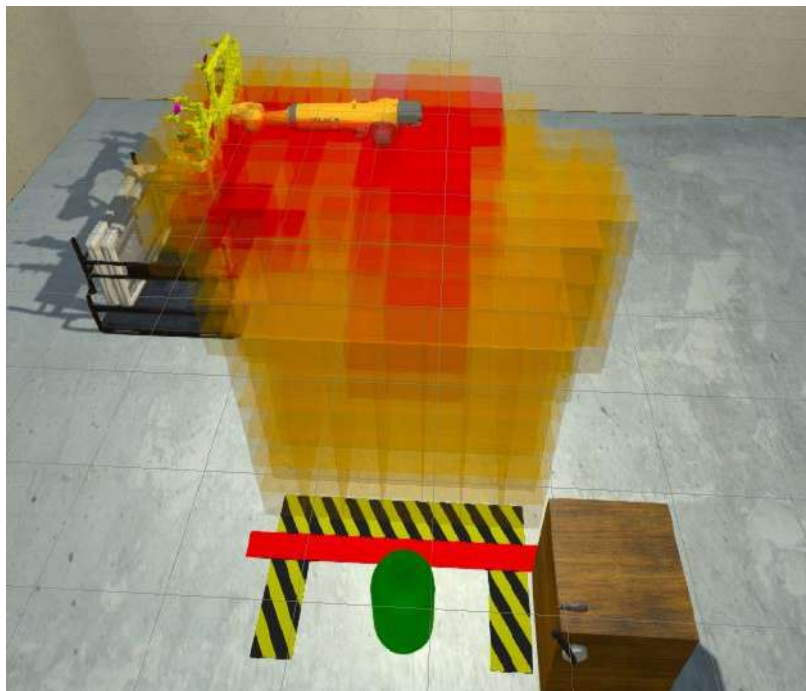


Figure 48: Static safety zones - Cube Representation

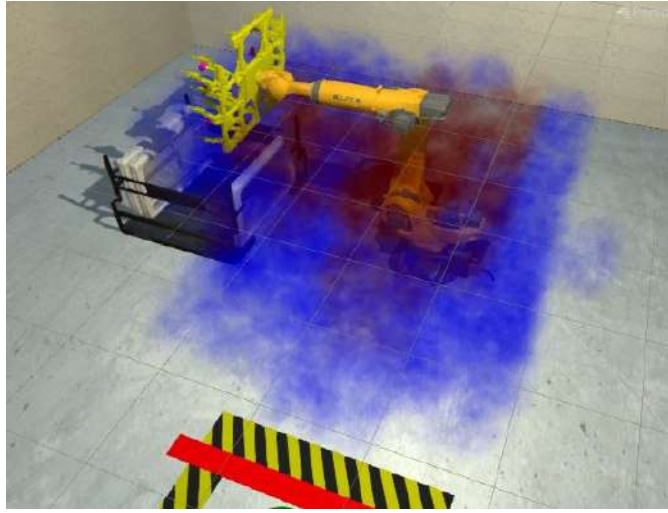


Figure 49: Static safety zones - Fog representation

By taking all this into consideration, we proceeded on visualising a fixed number of future steps, in contrast to the whole animation, i.e., dynamic rendering of safety zones. With this, we still inform the user about the imminent danger areas around him, but retain a small amount of shapes into the scene, resulting in a better performance-wise simulation.

This means that, for example in Figure 50, the safety zone which is rendered, concerns the position of the collided cubes from this step till the next 4. In the dynamic area rendering, when considering the fog particle system, we also provided a top view of the safety zones, which is rendered in the board, visible in Figure 50.

Last but not least, we incorporated the provided information from octree-based risk representation and rendered it with the cube method as shown in Figure 51.



Figure 50: Dynamic safety zones - Fog Representation.

5.3.2.6 Rendering in AR

The AR implementation is mostly focused on providing a tangible, mixed reality result to enable the use of the digital twin and the rendered safety zones in an actual working environment, through the use of ArUco markers, for example. To enable rendering in our simulation, we are mainly exploiting the data provided by the octree-based representation, as this depicts our introduced framework.

In order to deploy our VR Project into AR, camera settings need to change, to implement specific configuration profiles to map the Hololens 2 functionality inside the simulation. For this purpose, we proceeded on replacing the VR components inside our simulation with the respective AR ones, and we deployed our project to Hololens 2 visualizing the respective results.

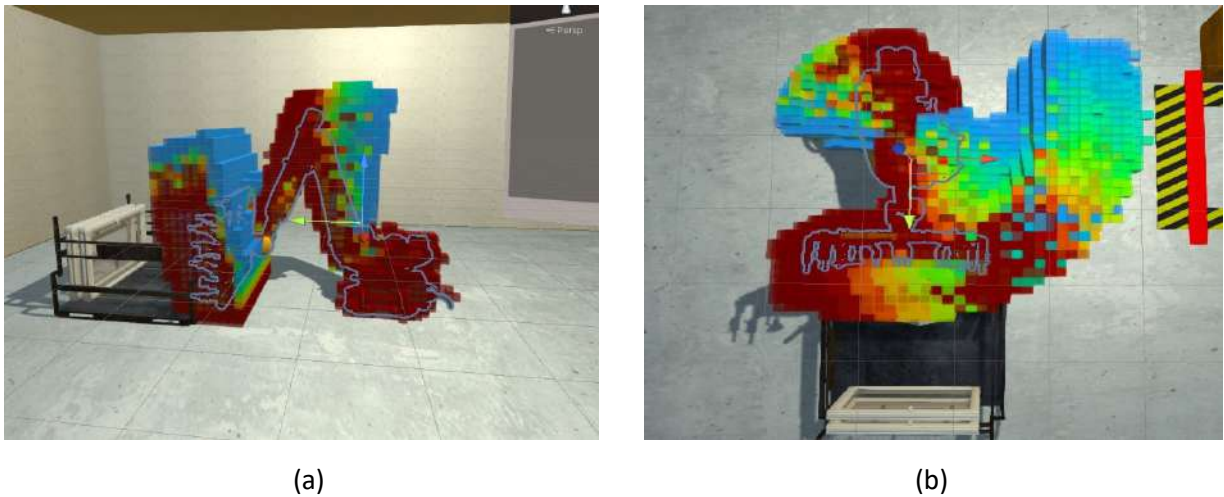


Figure 51: Dynamic safety zone construction with cube representation using Risk Assessment Occupancy Mapping. (a) Side View, (b) On top View.

In this section, we presented a framework for rendering static/dynamic safety zones in VR/AR. To emulate a real robotic movement, we developed a custom simulation of the robot's 3D digital model and placed it in to a virtual working environment. Probabilistic occupancy maps were used to estimate the risk of upcoming human-robot collision based on all possible movements of the robot's joints. In the case of specific assembly operations, the estimated spatio-temporal risk factor could be adapted to take into account only the programmed robotic motion trajectories reducing the visualized risk for other configurations.

Furthermore, we introduced a state-aware collision risk prediction term that takes into account not only the robot's pose, but also the rotation of the robotic joints at each time step, thereby allowing to predict the most probable direction of movement. However, it should be noted that while the assumption of smooth changes in joints' rotation holds for short time periods, it can't be used for more long term predictions. Moreover, this assumption is less valid when reaching the rotational limits (boundary points), thereby reducing the confidence in the estimation in those cases. In the future we intend to combine the dynamic and the state-aware collision risk terms through appropriate temporally changing weights, which will put more emphasis on the first or second term according to need for short or long-term prediction, and the confidence of the estimation at each time step.

Moreover, in this work the configuration of the robot at each time point is considered to be known, and the pose of the human operator is not taken into account. We are currently investigating the incorporation of 3D pose estimation algorithms based on visual information (through cameras placed in the workcell), that will allow to

D3.4 AR Interfaces Supporting Object and Scene Manipulation for Increasing Situational Awareness

identify an imminent danger, i.e., when the operator enters a high risk zone, and accordingly adjust the rendered information (e.g. through more opaque or vivid colors, or triggering of sound). While we are not focusing on control systems that may interrupt the robotic motion used in the industrial domain, we are interested in investigating the effect of advanced interfaces in augmented reality that increase the operators' awareness and safety in the best possible way. Thereby, future work includes the performance of a user evaluation study that will allow to better understand the limitations of our proposed approach, providing a potential of improvement, also in relation to human factors.

6 Conclusions

To summarize, in this deliverable novel visualization tools are demonstrated in order to improve the situational awareness of the users in challenging use cases of automotive and manufacturing pillar. The studies focus on drivers of semiautonomous vehicles and operators working in human-robot collaborative industrial environments. The developed visualization tools provide the users with information regarding their tasks in order to enhance their situational awareness and also receive notifications and visual aids regarding potential danger situations that may cause accidents.

The provided solutions aim to improve the situational awareness of the drivers in a cooperative driving scenario for semiautonomous and connected vehicles and the operators that work in collaboration with robots in human-robot collaborative industrial environments. Regarding the automotive pillar, three different use cases are presented (i) the visualization of potholes and other obstacles in the range of the road, (ii) the visualization of occluded vehicles in the VR/AR UPAT simulator, (iii) the visualization of warnings related to driver's drowsiness and other security aspects. Regarding the manufacturing pillar, also three different use cases are presented (i) the estimation of the working envelope of a robotic arm and the visualization of the octree based robot risk mapping, (ii) the estimation of warnings for the reconfiguration of the windshield based on the operator's pose recognition and (iii) warnings based on the operator's proximity with the robot and visualization of safety zones.

7 References

- [1] H. Liu, Y. Wang, W. Ji, and L. Wang, "A context-aware safety system for human-robot collaboration," *Procedia Manufacturing*, vol. 17, pp. 238–245, 2018..
- [2] N. Vahrenkamp, H. Arnst, M. Wächter, D. Schiebener, P. Sotiropoulos, M. Kowalik, and T. Asfour, "Workspace analysis for planning human-robot interaction tasks," in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*. IEEE, 2016, pp.
- [3] A. Mohammed, B. Schmidt, and L. Wang, "Active collision avoidance for human–robot collaboration driven by vision sensors," *International Journal of Computer Integrated Manufacturing*, vol. 30, no. 9, pp. 970–980, 2017..
- [4] A.-N. Sharkawy and N. Aspragathos, "Human-robot collision detection based on neural networks," *Int. J. Mech. Eng. Robot. Res*, vol. 7, no. 2, pp. 150–157, 2018..
- [5] Endsley, M. R. (1995). *Toward a theory of situation awareness in dynamic*.
- [6] Endsley, M. R. (1988). *Design and Evaluation for Situation Awareness Enhancement*. Proceedings of the Human Factors Society Annual Meeting, 32(2), 97-101. <https://doi.org/10.1177/154193128803200221>.
- [7] Endsley, M. R., & Jones, D. G. (2012). *Designing for situation awareness: An approach to human-centered design* (2nd ed.). London, UK: Taylor & Francis. Gallese, V., & Lakoff, G. (2005). *The brain's concepts: The role of the sensory-motor system in concept*.
- [8] T. Nguyen, C. P. Lim, N. D. Nguyen, L. Gordon-Brown and S. Nahavandi, "A Review of Situation Awareness Assessment Approaches in Aviation Environments," in *IEEE Systems Journal*, vol. 13, no. 3, pp. 3590-3603, Sept. 2019, doi: 10.1109/JSYST.2019.2918283..
- [9] Stanton, N. A., Salmon, P. M., Walker, G. H., & Jenkins, D. P. (2010). Is situation awareness all in the mind?. *Theoretical Issues in Ergonomics Science*, 11(1-2), 29-40..
- [10] Stanton, N. A., Salmon, P. M., Walker, G. H., Salas, E., & Hancock, P. A. (2017). State-of-science: situation awareness in individuals, teams and systems. *Ergonomics*, 60(4), 449-466..
- [11] K. Nazemi, D. Burkhardt, D. Hoppe, M. Nazemi, and J. Kohlhammer, "Web-based Evaluation of Information Visualization," *Procedia Manuf.*, vol. 3, no. Ahfe, pp. 5527–5534, 2015..
- [12] R. E. Patterson et al., "A human cognition framework for information visualization," *Comput. Graph.*, vol. 42, pp. 42–58, 2014..
- [13] 2. "Taxonomy and definitions for terms related to driving automation" SAE Standard J3016201806.
- [14] G. Arvanitis, A. S. Lalos, and K. Moustakas, "Robust and fast 3-d saliency mapping for industrial modeling applications," *IEEE Transactions*.
- [15] A. Dosovitskiy, G. Ros, F. Codevilla, A. M. López, and V. Koltun, "CARLA: an open urban driving simulator," *CoRR*, vol. abs/1711.03938, 2017. [Online]. Available: <http://arxiv.org/abs/1711.03938>.
- [16] Asam opendrive. [Online]. Available: <https://www.asam.net/standards/detail/opendrive/>.
- [17] P. Bazilinskyy, A. Eriksson, S. Petermeijer, and J. de Winter, "Usefulness and satisfaction of take-over requests for highly automated driving," 10 2017..
- [18] H. White, D. Large, D. Salanitri, G. Burnett, A. Lawson, and E. Box, "Rebuilding drivers' situation awareness during take-over requests in level 3 automated cars," 04 2019..
- [19] M. Pavlou, D. Laskos, E. I. Zacharaki, K. Risvas, and K. Moustakas "Xrsise: an xr training system for interactive simulation and ergonomics assessment," *Frontiers in Virtual Reality*, vol. 2, p. 17, 2021..
- [20] R. Schnabel and R. Klein, "Octree-based point-cloud compression." in *PBG@ SIGGRAPH*, 2006, pp. 111–120..
- [21] S. H. Choi, et al. , "An integrated mixed reality system for safety-aware human-robot collaboration using

deep learning and digital twin generation,” *Robotics and Computer-Integrated Manufacturing*, vol. 73, p. 102258, 2022..

- [22] G. Michalos, P. Karagiannis, S. Makris, O. Tokcalar, and G. Chryssolouris, “Augmented reality (ar) applications for supporting human-robot interactive cooperation,” *Procedia CIRP*, vol. 41, pp. 370–375, 2016.
- [23] A. Hietanen, R. Pieters, M. Lanz, J. Latokartano, and J.-K. Kämäräinen, “Ar-based interaction for human-robot collaborative manufacturing,” *Robotics and Computer-Integrated Manufacturing*, vol. 63, p. 101891, 2020..
- [24] M. Krüger, M. Weigel, and M. Gienger, “Visuo-tactile ar for enhanced safety awareness in human-robot interaction,” in *HRI 2020 Workshop on Virtual, Augmented and Mixed Reality for Human-Robot Interaction (VAM-HRI)*, 2020..
- [25] G.-C. Vosniakos, L. Ouillon, and E. Matsas, “Exploration of two safety strategies in human-robot collaborative manufacturing using virtual reality,” *Procedia Manufacturing*, vol. 38, pp. 524–531, 2019.
- [26] P. Adami et. al, “Effectiveness of vr-based training on improving construction workers’ knowledge, skills, and safety behavior in robotic teleoperation,” *Advanced Engineering informatics*, vol. 50, p. 101431, 2021..
- [27] M. Dianatfar, J. Latokartano, and M. Lanz, “Concept for virtual safety training system for human-robot collaboration,” *Procedia Manufacturing*, vol. 51, pp. 54–60, 2020..
- [28] M. Giorgini and J. Aleotti, “Visualization of agv in virtual reality and collision detection with large scale point clouds,” in *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*. IEEE, 2018, pp. 905–910..
- [29] Marike K. van den Broek and Thomas B. Moeslund. 2020. Ergonomic Adaptation of Robotic Movements in Human-Robot Collaboration. *ACM/IEEE International Conference on Human-Robot Interaction (HRI '20)*. Association for Computing Machinery, New York, NY, USA.
- [30] R. B. Rusu and S. Cousins, “3D is here: Point Cloud Library (PCL),” in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011..
- [31] Z. Makhataeva, A. Zhakatayev, and H. A. Varol, “Safety aura visualization for variable impedance actuated robots,” in *2019 IEEE/SICE International Symposium on System Integration (SII)*, 2019, pp. 805–810..
- [32] Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W. (2013). OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*.
- [33] Tanida, K., & Poppel, E. (2006). A hierarchical model of operational anticipation windows in driving an automobile (32–40). New-York: Marta Olivetti Belardinelli and Springer-Verlag, 187..
- [34] Vicente, K. J., & Rasmussen, J. (1988). On Applying the Skills, Rules, Knowledge Framework to Interface Design. *Proceedings of the Human Factors Society Annual Meeting*, 32(5), 254–258. <https://doi.org/10.1177/154193128803200501>.
- [35] Dynamic safety zone construction with cube representation using Risk Assessment Occupancy Mapping.
- [36] Lorenzo Vianello, Jean-Baptiste Mouret, Eloïse Dalin, Alexis Aubry, Serena Ivaldi. Human posture prediction during physical human-robot interaction. *IEEE Robotics and Automation Letters*, IEEE 2021, 6 (3), pp.6046-6053. 10.1109/LRA.2021.3086666. hal-031152.
- [37] Marike K. van den Broek and Thomas B. Moeslund. 2020. Ergonomic Adaptation of Robotic Movements in Human-Robot Collaboration. In *Companion of the 2020 ACM/IEEE International Conference on Human-Robot Interaction (HRI '20)*. Association for Computing Machin.