D6.8 - Safety concept for robotic systems (creation)

corosect.eu

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101016953

Author(s)/Organisation(s)	Jose-Ramon Martinez-Salio (Atos)
Contributor(s)	
Work Package	WP6
Delivery Date (DoA)	31/12/2022
Actual Delivery Date	06/02/2023
Abstract:	Documentation of planned CoRoSect safety concept for robotic systems in insect farms.

Document Revision History			
Date	**Version**	**Author/Contributor/ Reviewer**	**Summary of main changes**
29/09/2022	0.1	Jose-Ramon Martinez	First working version
01/11/2022	0.2	Jose-Ramon Martinez	Second version
08/11/2022	0.3	Miquel Mila Prat	Third version
22/11/2022	0.4	Jose-Ramon Martinez	Fourth version
05/12/2022	0.5	Jose-Ramon Martinez	Final version for peer review
19/12/2022	0.6	Jose-Ramon Martinez	Final version
25/1/2023	0.7	Jose-Ramon Martinez	Final additions after peer review

Dissemination Level		
PU	Public	**x**
PP	Restricted to other programme participants (including the EC Services)	
RE	Restricted to a group specified by the consortium (including the EC Services)	
CO	Confidential, only for members of the consortium (including the EC)	

Funding Scheme: Innovation Action (IA) ● Topic: H2020-ICT-46-2020

Start date of project: 01 January, 2021 ● Duration: 36 months

CoRoSect Consortium			
Participant Number	**Participant organisation name**	**Short name**	**Country**
1	UNIVERSITEIT MAASTRICHT https://www.maastrichtuniversity.nl/	UM	NL
2	ETHNIKO KENTRO EREVNAS KAI TECHNOLOGIKIS ANAPTYXIS https://www.certh.gr/	CERTH	GR
3	HOCHSCHULE EMDEN/LEER https://www.hs-emden-leer.de/en/	HSEL	GER
4	LUONNONVARAKESKUS https://www.luke.fi/	LUKE	FIN
5	OULUN AMMATTIKORKEAKOULU OY - OULU UNIVERSITY OF APPLIED SCIENCES https://www.oamk.fi/fi/	OAMK	FIN
6	FUNDACION PARA LAS TECNOLOGIAS AUXILIARES DE LA AGRICULTURA http://www.fundaciontecnova.com/	TECNOVA	ES
7	KATHOLIEKE UNIVERSITEIT LEUVEN https://www.kuleuven.be/kuleuven/	KU LEUVEN	BEL
8	ATOS IT SOLUTIONS AND SERVICES IBERIA SL https://atos.net/en/	ATOS	ES
9	ROBOTNIK AUTOMATION SLL http://www.robotnik.es/	ROB	ES
10	AGVR BV www.agvegroup.com	AGVR	NL
11	NASEKOMO AD https://nasekomo.life/	NASEKOMO	BG
12	ENTOMOTECH SL http://entomotech.es/	ENTOMOTECH	ES
13	ENTOCYCLE LTD https://www.entocycle.com/	ENTOCYCLE	GB
14	SOCIETA AGRICOLA ITALIAN CRICKET FARM SRL https://www.italiancricketfarm.com/	ICF	IT
15	INVERTAPRO AS https://www.invertapro.com/	INVERTAPRO	NOR
16	FIELD LAB ROBOTICS BV https://www.fieldlabrobotics.com/	FLR	NL
17	FoodScale Hub https://foodscalehub.com/	FSH	RS
18	AgriFood Lithuania DIH https://www.agrifood.lt/	AFL	LT
19	CENTRO INTERNAZIONALE DI ALTISTUDI AGRONOMICI MEDITERRANEI http://www.iamb.it/	CIHEAM	IT

Table of Contents

List of tables

List of figures

List of Abbreviations and Acronyms	
AGV	Automatic Guided Vehicles
CV	Computer Vision
DDS	Data Distribution Service
GIS	Geographic Information System
IMU	Inertial Measurement Unit
IP	Internet Protocol
JSON	JavaScript Object Notation
MQTT	Message Queuing Telemetry Transport
ROS	Robotic Operating System
SFM	Shop Floor Manager
SLAM	Simultaneous localization and mapping
SOTA	State-of-the-Art
YOLO	You only look once

Executive Summary

CoRoSect includes in its core concept the coexistence of people and machines in the factory space of insect farms. In such environments it is important to ensure that people are not exposed to danger (safety first directive) while not affecting the overall performance of the factory. Current deliverable alongside *D6.7 Safety concept for robotic systems (planning)* released in M12 and *D6.9 Safety concept for robotic systems (final)* due to M36 describes all the measures taken in the project to ensure such coexistence from the point of view of the experience gathered when building the software already planned. Thus, D6.7 deliverable focused on the theoretical planning of the safety concept with a discussion of all possible alternatives, the current deliverable, D6.8 documents the decision taken and the actual elements finally built for the project and, finally, D6.9, due to M36, will include the conclusions obtained after the real application of the software in the project´s pilots.
Since this is the second part of a three series documents, there will be some inevitable repetitions in some general descriptions and overall concepts. We have kept these repetitions (with some minor amendments) since they are needed to provide context for the document.

Introduction

[taken from D6.7] ”Safety comes first” is the first directive in the industry. CoRoSect project has the objective of the automation of insect farms passing from current ”almost manual” operations stage to a fully automated operational stage. Such automatization, that implies robots with minimal human supervision, creates new risks in safety. In a fully automated farm, we will have autonomous robots moving “elements” and operating in a dynamic environment at “high” speed and making their own decisions (e.g. about the path to take). We should ensure that this does not create any hazard to the human beings, no matter their behaviour (i.e. even if they enter into ”no trespass” areas or behave in a dangerous way). On the other hand, we cannot override any existing security measures already in place for the machines (that usually makes them stop when any problem or hazard occurs). Finally, we want to improve the efficiency of the operation of these machines by preventing them to collide and at the same time we want to keep them moving as much as possible to avoid interruptions, thus improving overall efficiency. This creates for us three key requirements that we should fulfil (in order of importance):

1 Humans must not be exposed to any danger at any moment.
2 Existing safety measures must not be overridden (even if this provokes machinery to stop working)
3 Operations, even if fully automated, should not be interrupted whenever possible, if adequate and guaranteed permitted situations are present. Halting the machines should be always the last option.

Current deliverable, result of the *Task 6.7* ***Implementing safety control in robotic (planning)***, is the second of a three-document series. It will describe the actual decisions and the software built in CoRoSect for these safety related issues.

Section 1 will describe our software approach to the problem taking these requirements into account. Section 2 summarizes the document results in the conclusion.

For the theoretical discussion about standards and possible software solutions, please refer to *D6.7 Safety concept for robotics systems (planning)* already delivered on M12.

1 Software solution built in CoRoSect

Following all the requirements described in D6.7, we have finally built three separate but complementary software products:

1. Map creator (common component)
2. Obstacles Detector
3. Route manager

These three elements share a common map incorporated in a database written in PostgreSQL[1] with the GIS plugin (the result is called PostGIS[2]). This will be the "factory map" and will be also coordinated with the maps used in any other software or hardware element (for example in the AGV and the shop floor manager).

These three tools are independent. They communicate between them via JavaScript Object Notation (JSON) messages (using MQ Telemetry Transport MQTT) and the PostGIS map.

The common Map

This is a common map in PostGIS that includes the 2D representation of the factory taken into account:

- Areas allowed for pass of the AGV: these areas are the "aisles" that are allowed for passing
- Named points of interest like "parking station", "rearing station", etc. These points will be used for navigation proposes (e.g. "go from Parking station to Maturing station") complementing the coordinates.

This map will be created by the map creator tool based on SLAM (Simultaneous Localization And Mapping), and then will be corrected manually if it is needed (for example to indicate small details), additionally it will incorporate the points of interest of the factory under consideration.

We will need to coordinate or even share this map with other software and hardware components like the AGV and the Shop Floor Manager in order to: 1) stablish the same origin of coordinates relative to the factory map (typically in the lower left corner chosen arbitrarily as zero point), 2) coordinate the "size" of the factory between maps so that the coordinates will be identical, and 3) set the same points of interest so we can use them later for the routes communication flow (for example in "go from Parking station to Rearing Station").

1.1. Map creator (common)

This tool is not specifically a direct software outcome of CoRoSect (created as result of a specific task). The intention of the tool is to support the creation of the common map of the factory to be shared by all our components and coordinated with AGV and other tools.

For this product, we have explored two versions of SLAM algorithms and cameras (monocular and binocular).

[1] https://www.postgresql.org/
[2] https://postgis.net/

1. ORB-SLAM2 with monocular camera

We built a software in Python based on ORB-SLAM2 (https://github.com/raulmur/ORB_SLAM2) with the use of monocular camera (a "normal" IP camera). We checked the results and added some complementary modules to 1) create the 2D projection of the result, 2) filter and improve the results, 3) complete the "lines" by joining points together (interpreted as walls) and 4) put the result into PostGIS database.

The results were good but could be improved. We encountered limitations in our use cases (for a factory-like layout) due to the use of a monocular camera (thus without depth information and with a limited angle of view). We also found limitations in the case of "elements without features" like continuous white walls that don't provide features to be used by the algorithm thus making it difficult to detect the shape of them (we can see some real results in Figure 1).

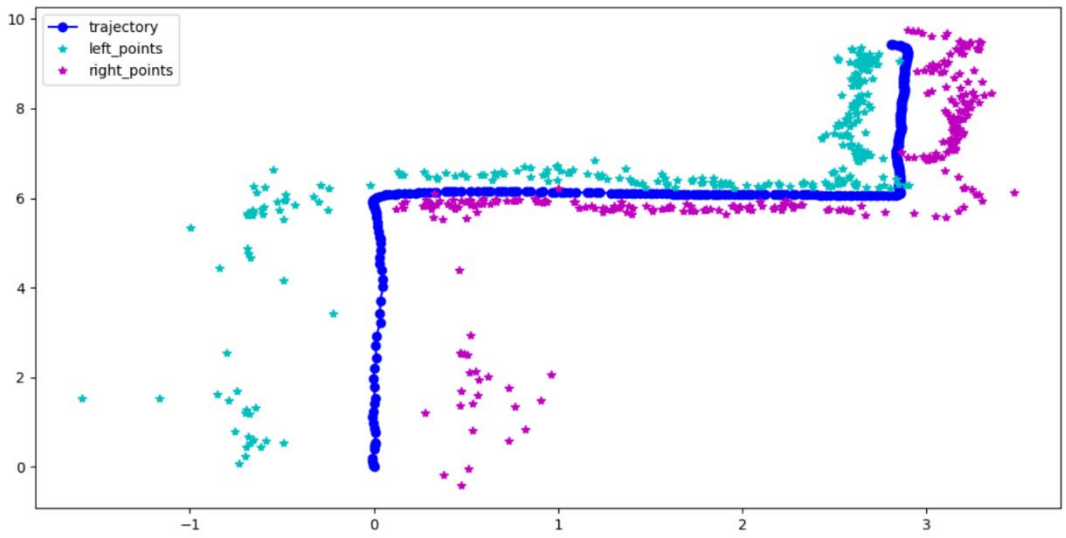

Figure 1 - ORB-SLAM2 results

2. ROS-ISAAC with binocular camera

We next moved to the use of another SLAM approach (ROS-ISAAC: https://github.com/NVIDIA-ISAAC-ROS) with the addition of a binocular camera with IMU (Inertial Measurement Unit) to help complete the map. For that propose, we have used a Zed2i stereo camera (https://www.stereolabs.com/zed-2/) (Figure 2)

Figure 2 - zed 2i stereo camera

This approach is more robust than the previous one in three key elements:

- Camera angle is now 120º. This allows us to add more “visual features” in each “pass”
- Camera has Inertial Measurement Unit (IMU). This IMU can be used by SLAM to complete and complement the images
- Adds depth information that can be used to determine distances

The use of ROS-ISAAC implied the creation of an “adaptor” or data-bridge needed for adding the results of ROS to the PostGIS database. For that, we created a DDS (Data Distribution Service) subscriber (this is the protocol used by ROS2) to read the “costmap” (Figure 3) (in NAV2 format https://navigation.ros.org/index.html) created by the tool and put it in our PostGIS database.

Figure 3 - NAV2 costmap

Note that this process doesn’t need to be “real time” and will be performed for each factory before the real pilot action. The results, in PostGIS can be then annotated (using a specific annotation tool)

for setting the “points of interest”, used for synchronization of the map and corrected manually if deemed necessary.

1.2. Obstacles detector

The tool called “obstacles detector” is a Python program that, using a monocular camera footage in “near” real time is able to detect up to 22000 categories of objects. These objects are then considered as obstacles if they are in the “passing” areas of the factory. Obstacles can be of any kind, static (i.e. not moving) or dynamic (moving). Finally, the current position of these obstacles and the future projection of their movement are sent to the “routes manager” to detect and avoid any possible collision.

Step by step, the program does (Figure 4):

- Reads the common map from PostGIS database. This map is used to know the “passing areas” and to position the obstacles detected.
- Reads the live footage of a camera and, for each frame:
    - Corrects the image (using camera correction data created from a previous calibration pass)
    - Detects the possible obstacles and segment them adjusting the “mask” to the detected element
    - Tracks (i.e. follows) the object(s) as obstacles
    - Detects the depth of these obstacles
        - Corrects this depth to the real distance
        - Smooths the distance “jumps” between frames using Kalman filter
    - With the depth information, places the obstacles in the 2D map
    - Creates a “future” projection of the movement of the obstacles (can also be static) for the next seconds using a different Kalman filter
    - Sends these information to the “routes manager” using Mosquitto (MQTT)

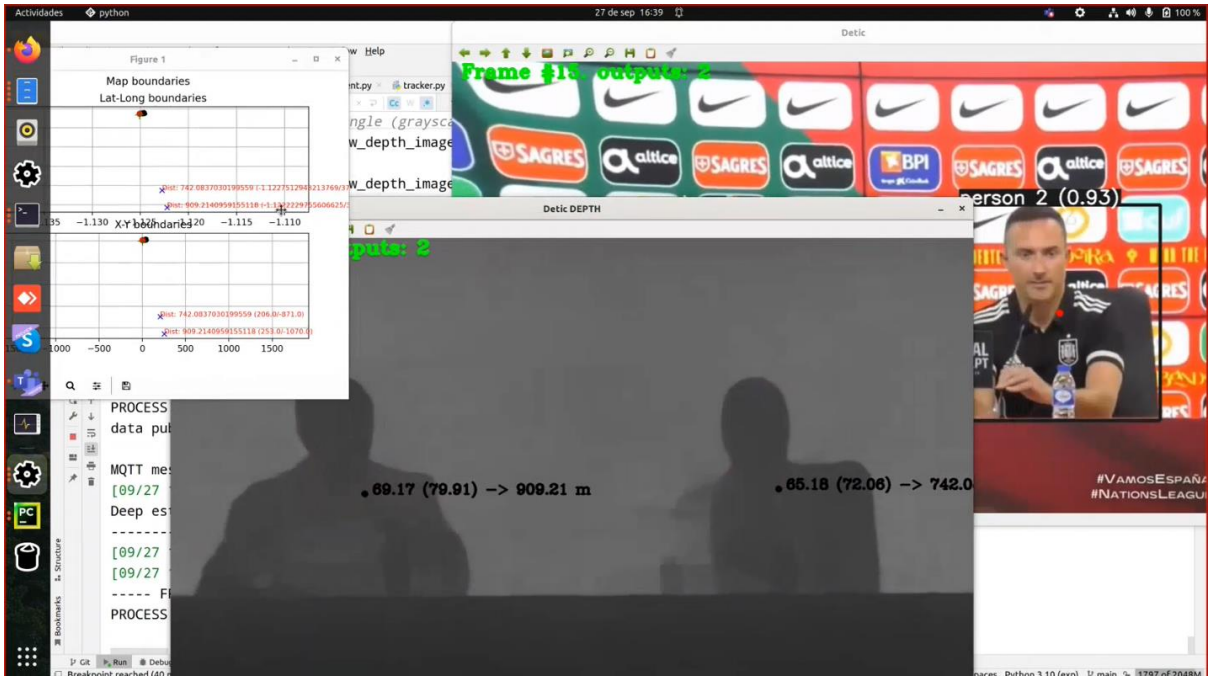

Figure 4 - Detection of obstacles and depth map

This detection is not limited to people (as described in Task 5.2) but also to any possible object in 22000 categories that can be a potential obstacle.

The camera used for this detection is not binocular, but monocular. The reason of this election is to take advantage of the most typical cameras already present in factories (IP cameras). On the other hand, this election makes the use of “depth” channel (Figure 5) impossible forcing us to use a CV (Computer Vision) algorithm for estimation of depth. The camera that has finally being chosen is a “commercial” one with the constraints determined by the scenario conditions of “low light”, “high humidity” and “high temperature” (Figure 6).

Figure 5 - Depth map example using CV from a monocular camera

For each of the features described we have tried to use the best available algorithms in terms of accuracy with the less latency added.

- Detection: has been made using Detic (based on Facebook Detectron2): https://github.com/facebookresearch/Detic
- Fine adjustment of segmentation has been made using PointRend: https://github.com/zsef123/PointRend-PyTorch
- Tracking: has been made using deepsort ported to cuda (own development)
- Depth estimation: we have used DPT transformer: https://github.com/isl-org/DPT
- Estimation of trajectory was made using Kalman

Note that our approach of use of monocular cameras is very innovative and different to current approaches that typically uses binocular cameras attached to the robots. With this approach we try to re-use the usual cameras already existing in most factories. In Table 1 we can see a summary of some innovative aspects of our solution

Table 1 - Innovative aspects of the solution

Our approach	**Traditional approach**
Use the cameras already present in the factory (monocular) connected to the common net	Use specific cameras (binocular) attached to the robots. Use of specific connectors
Detect people and as much categories as possible (up to 22000)	Detect only people or a limited set of objects (80)
Separation between analysis and camera, allowing processing of multiple cameras with the same software	Cameras and software are included in a common block (usually with the robot). Extension to new cameras is limited

Figure 6 - IP camera for detection of obstacles

Rationale of the election of the obstacles detector algorithm

In the core of the detection of people and obstacles, there is an object detector algorithm. A complete study of the State of the Art (SOTA) will be included in *D8.2 Autonomous and human aware robot trajectory planning for safe and efficient HRC* due to M32. Here we will discuss briefly the rationale of the election of the algorithm taken into account the current SOTA.

Detection of people and objects has been led, from 2016 (which in computer vision is a long term) by the YOLO (You Only Look Once) family of detectors. All these (and their successors) are trained using COCO dataset (https://cocodataset.org/#home) with 80 categories (list taken from https://github.com/srebroa/awesome-yolo)

- **Yolo v1** (2016) Joseph Redmon 'You Only Look Once: Unified, Real-Time Object Detection'
- **Yolo v2** (2017) Joseph Redmon 'YOLO9000: Better, Faster, Stronger'
- **Yolo v3** (2018) Joseph Redmon 'YOLOv3: An Incremental Improvement'
- **Yolo v4** (2020) Alexey Bochkovskiy 'YOLOv4: Optimal Speed and Accuracy of Object Detection'.
- **Yolo v5** (2020) Glen Jocher - PyTorch implementation (v1 to v4 Darknet implementation).
- **PP-Yolo** (2020) Xiang Long et al.(Baidu) 'PP-YOLO: An Effective and Efficient Implementation of Object Detector'.
- **Yolo Z** (2021) Aduen Benjumea et al. 'YOLO-Z: Improving small object detection in YOLOv5 for autonomous vehicles'
- **Yolo-ReT** (2021) Prakhar Ganesh et al. 'YOLO-ReT: Towards High Accuracy Real-time Object Detection on Edge GPUs'
- **Scaled-Yolo v4** (2021) Chien-Yao Wang et al. 'Scaled-YOLOv4: Scaling Cross Stage Partial Network'
- **YoloX** (2021) Zheng Ge at all. 'YOLOX: Exceeding YOLO Series in 2021'.
- **YoloR** (You Only Learn One) (2021) Chien-Yao Wang et al. 'You Only Learn One Representation: Unified Network for Multiple Tasks'
- **YoloS** (2021) Yuxin Fang at all. 'You Only Look at One Sequence: Rethinking Transformer in Vision through Object Detection'
- **YoloF** (2021) Qiang Chen at all. 'You Only Look One-level Feature'
- **YoloP** (2022-v7) Dong Wu at all. 'YOLOP: You Only Look Once for Panoptic Driving Perception'.
- **Yolov6** (2022) Hardware-friendly design for backbone and neck, efficient Decoupled Head with SIoU Loss,
- **Yolov7 not official** (2022)
- **Yolov7 official** (2022) Chien-Yao Wang at all. 'Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors'
- **Yolov8** (2023) developed by Ultralytics

Accuracy and speed in detection has increased with each new version.

On the other hand, in the category of detection of "many" categories we find the Facebook family of algorithms called Detectron. Detectron is a software system that implements SOTA object detection algorithms, including Mask R-CNN (*Marr Prize at ICCV 2017),* RetinaNet *(Best Student Paper Award at ICCV 2017),* Faster R-CNN, RPN, Fast R-CNN and R-FCN. The timeline of Detectron is:

- Detectron (2018): https://github.com/facebookresearch/Detectron

- Detectron2 (2022): https://github.com/facebookresearch/detectron2

As a evolution of Detectron2, Detic [3] (https://github.com/facebookresearch/Detic) released in 2022 is able to detect 22000 categories of objects.

Detic is not, by any means the most accurate or fast algorithm of detection of objects, we have chosen it based on the following reasons:

- Possibility of detecting up to 22000 different categories of possible obstacles and people. This should include all possible real obstacles that can be found in a real insect farm
- Possibility of detecting people in a effective way in the distance range considered of 15 meters (limited by the depth estimation algorithms and the camera)
- Accurate of detection, if not the best of breed, is good enough for the task
- Real time requisites for the detection does not need strict real time. We can take some few seconds for the analysis considering the relative speed of people and the "projection" of trajectories that we apply (projection for the next seconds)

Inclusion of Human motion prediction in the detection of Obstacles

Task 5.2 explicitly mentions the use of human motion prediction in the task description. We have included this motion prediction included inside the obstacles detector tool. We could have detected the people and the rest of obstacles using separate algorithms (in the same or different tools), but we decided not to do that.

The reasons for this decision were:

- We have included the detection of people included with the detection of "objects". With this mixture we can detect more categories of objects, but, at the same time, can lose precision in the detection of people. We are aware of this issue, but since the detection range will be limited (in our estimation) in real application to a range of less than 15 meters to the camera there is no real difference in the detection of people. Note that the idea, in final deployment is to combine multiple cameras (with the corresponding detectors) to cover the entire space. The estimation of the range of 15 meters is due to the use of "depth" detection algorithms that are not trustable after this distance and limitations in the calibration of the monocular camera used. As an additional reason, the most popular solutions for detection of people are not "pure" but include, in the training, many other categories of objects. For example a typical approach is to have the full COCO dataset that includes 80 categories of objects.
- All objects detected as obstacles have their motion "projected" (using Kalman) thus covering the motion prediction for people.
- Arguably, we could have included the detection of the "pose" or people or even the movement of limbs and hands. We have consciously not done that because we are interested in the detection of potential collisions in a longer range (meters). We are not including "fine" detection since i) this is already included in current security of the robots (AGV for example stops if it collides with an obstacle, robotic arms stop if someone enters their "cages") and ii) the intention is to detect people far from the robot, project their trajectories and avoid the potential collision before it happens in order to keep the robot moving at all times. With these considerations in mind, detecting the limbs position or even fingers do not add any advantage to the task.

[3] Detecting Twenty-thousand Classes using Image-level Supervision,
Xingyi Zhou, Rohit Girdhar, Armand Joulin, Philipp Krähenbühl, Ishan Misra,
ECCV 2022 (arXiv 2201.02605)

- The detection of “every possible object” as an obstacle, is not clearly included in T5.2 nor in T6.3 but we think that by including it we can make the use-case broader and more ambitious. We think that this is also a good extension of the description of Task 6.3 that completes the task to include every possible object that can be found in a real situation.

1.3. Route manager

The final tool in the set is the route manager tool. This tool is a separate module that has the responsibility of creating and managing the routes of any moving element (i.e. a robot). In our specific case this will be the AGV. The module can (Figure 7):

- Create a new route under Shop Floor Manager demand
- Send the route to the robot using Mosquitto MQTT
- Follow any number of routes by receiving the new position of the robot periodically
- Receive any detected obstacle from the “obstacles detector”, check it against the trajectory foreseen and, if needed recalculate and send the route again to the robot.

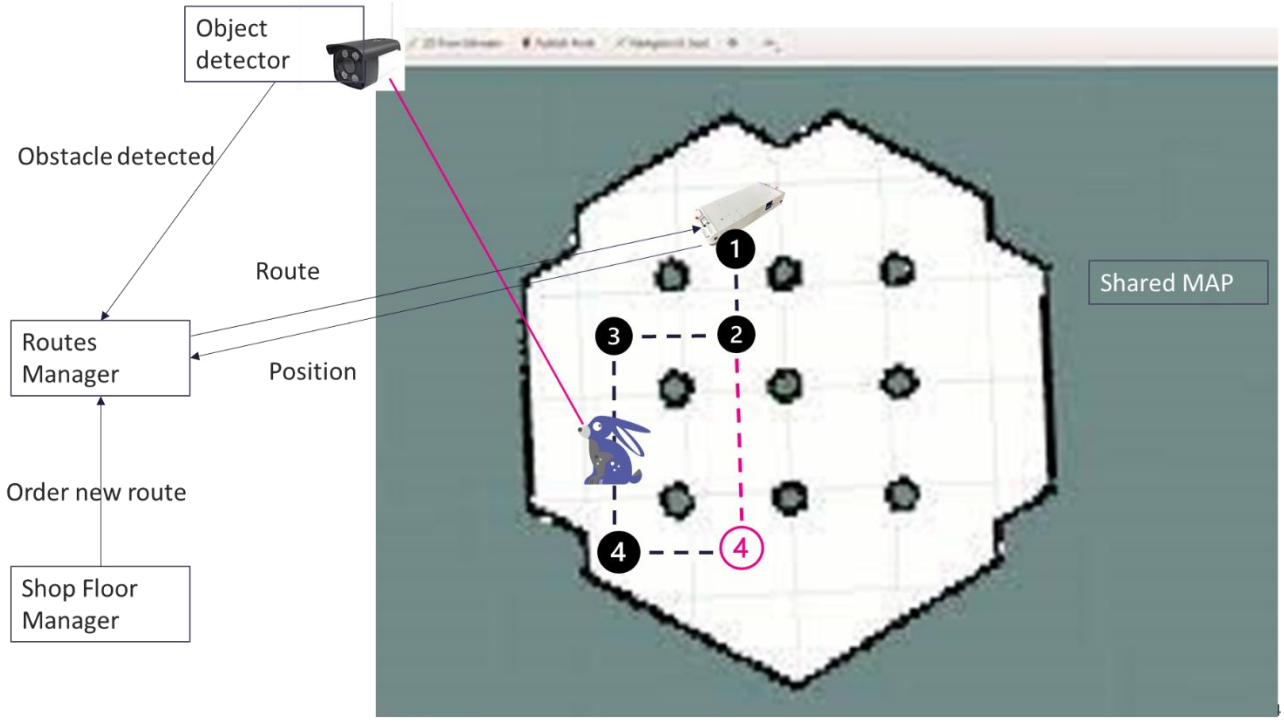

Figure 7 - Routes manager data flux

For all these missions, the module first loads the common map from PostGIS database. Then, it waits for the Shop Floor Manager to order a new route for a named robot (e.g. “AGV1”) from a position to a another (e.g. from (33,14) to “Rearing Station”). After that, it calculates the route taking into account the walls and a “safe” zone around them to avoid collisions. For the route calculation it uses D*

algorithm.[4] After that, it sends the starting points of the route to the robot (the AGV). In this point, the module starts tracking the AGV position in a continuous loop:

- Sends new set of points to the AGV (the "next" positions)
- AGV sends periodically the position
- If AGV reaches one of the "next" positions, we send new ones
- Until we reach the end of the route
- Then, it informs to the Shop Floor Manager that the route has ended

If during this process, the "obstacle detector" module detects a new obstacle, then the Route Manager checks if the new obstacle detected is in the path of the robot. If the new obstacle is in the path of the robot, then the algorithm recalculates the route (using the updated space) and start sending the new route to the robot (Figure 8).

This process can be done for any number of robots simultaneously.

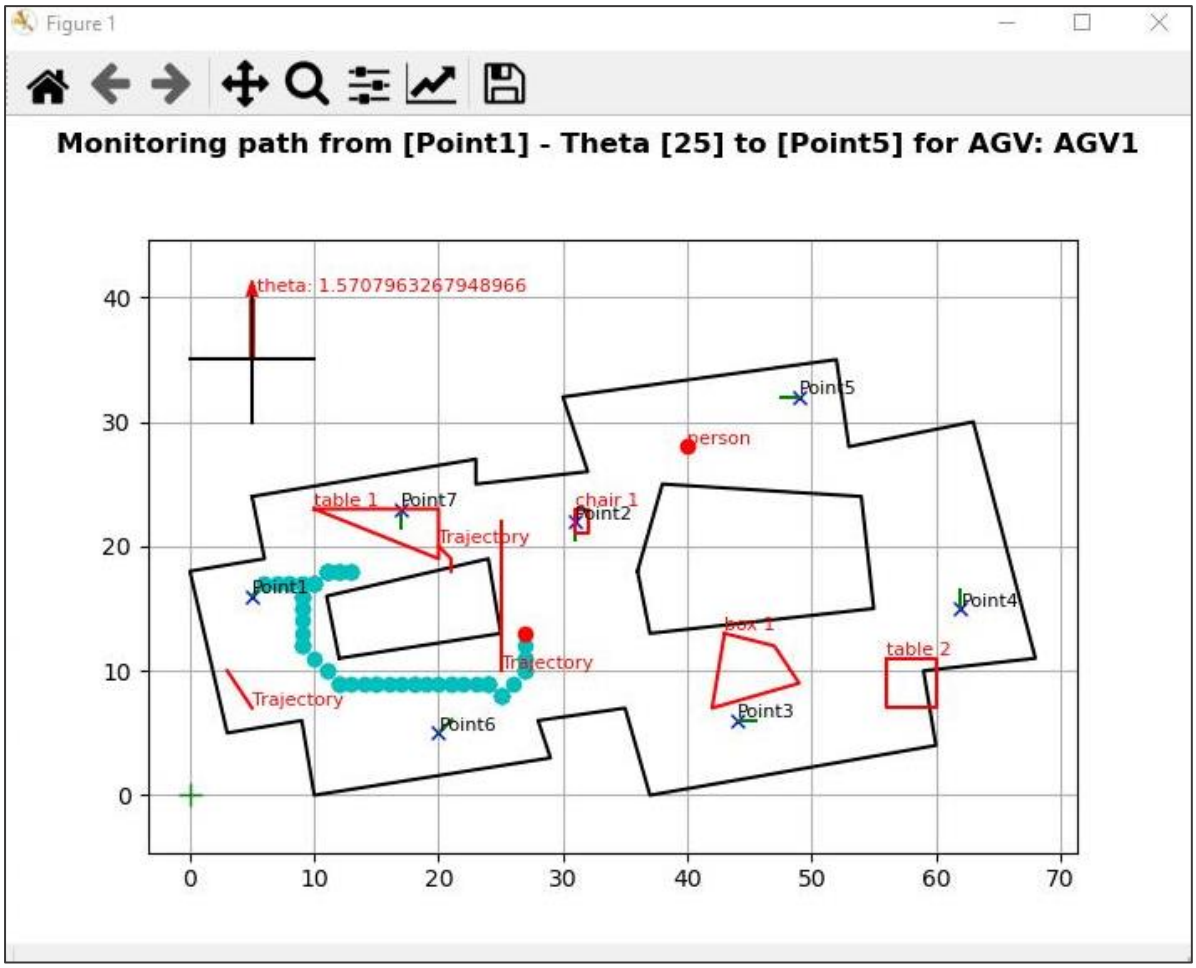

Figure 8 - Simulation of the route followed by a robot

[4] Stentz, A. (1997). Optimal and efficient path planning for partially known environments. In *Intelligent unmanned ground vehicles* (pp. 203-220). Springer, Boston, MA.

2 Conclusion

This deliverable is the second of the three dedicated to the safety planning for robotic systems. In the first deliverable, we described the requirements and what we planned to create for the task. In the current document, we have described what we have been building for the tasks in three separate software modules. In the third and last version of this document *D6.9 Safety concept for robotic systems (final)* due to M36 we will describe the behaviour of these software modules during the real pilots and the subsequent adjustments we will have had to make.

CoRoSect

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101016953