Development of Intelligent Systems

Final Report

Team Eta

Biljana Vitanova, Trajche Krstev, Andjela Rajovic

University of Ljubljana, Faculty of Computer and Information Science

May 2024

Contents

1 Introduction **2**

2 Methods and Implementation **2**

2.1 Face detection and recognition: 2

2.2 Ring and Cylinder detection 3

2.2.1 Colour Recognition 3

2.2.2 Point Cloud 3

2.2.3 Preprocessing 3

2.2.4 Ring detection 3

2.2.5 Cylinder detection 4

2.2.6 Handling multiple detections 5

2.3 Arm camera 5

2.4 QR Code Reading 5

2.5 Parking 6

2.6 Speech detection and recognition 6

2.7 Anomaly detection 6

3 Integration **8**

3.1 Robot Commander 8

3.1.1 Navigation 8

3.1.2 Robot commander process 8

3.1.3 Markers 10

4 Results **10**

4.1 Diagram 10

4.2 Evaluation 11

4.3 Experimental results 11

5 Division of work **12**

6 Encountered problems **12**

7 Conclusion **13**

1 Introduction

In this report, we detail our work on the "Robo da Vinci" robot, developed as the final project for the Development of Intelligent Systems course. We discuss the challenges we faced, the methods we used to overcome them, and provide a concise overview of our approach and results.

We needed to adjust the robot to perform different tasks in a small Renaissance town. There were several goals for Robo da Vinci:

First, the robot needed to navigate around and find the genuine QR code containing the picture of the Mona Lisa. To achieve this, the robot had to ask people in the city to guide it to the QR code, specifically to the art gallery (cylinder) where it was located. The townspeople provided useful information about where this art gallery might be and around which parking place the robot could find it. Once Robo da Vinci found the QR code containing the Mona Lisa picture, it had to traverse the city and find pictures of genuine Mona Lisa paintings. If these paintings contained anomalies, the robot needed to mark where these anomalies were located. Otherwise, it needed to show a friendly gesture.

To solve this task, we first needed to understand working in a ROS2 environment and how communication between different nodes is done. Working with various types of sensors and the data they provide was crucial. Transforming between coordinate frames, navigation, computer vision methods, and many other aspects needed to be taken into consideration.

2 Methods and Implementation

In this section, we will explain how we tackled each part of the task, our approach, and the methods we used to achieve the desired results. For most of the tasks, as a starting point, we used the code given by the assistants and further modified it.

2.1 Face detection and recognition:

We used the YOLO-8 model for real-time object detection, specifically focusing on extracting objects classified as faces. Once a face is detected, we calculate its center to obtain the corresponding 3D point from the point cloud. Additionally, we extract a feature descriptor for each face using a pretrained AlexNet CNN. For each new detection, we measure the spatial distance between its 3D center point and the center points of all previously detected faces. If all distances are beyond a certain threshold, the detection is marked as a new face. Otherwise, we compare the feature descriptors using cosine similarity. If these descriptors are distinctive enough, it is still considered a new face. With this two-stage approach, we ensure flexibility, not relying solely on fixed distances.
With this task, we tested the efficiency and accuracy of the **YOLO model** by scanning the entire image once, dividing the image into a grid of cells, and proposing bounding boxes and class probabilities for each cell. We also used **AlexNet**, which is considered one of the first CNNs used for recognition tasks.

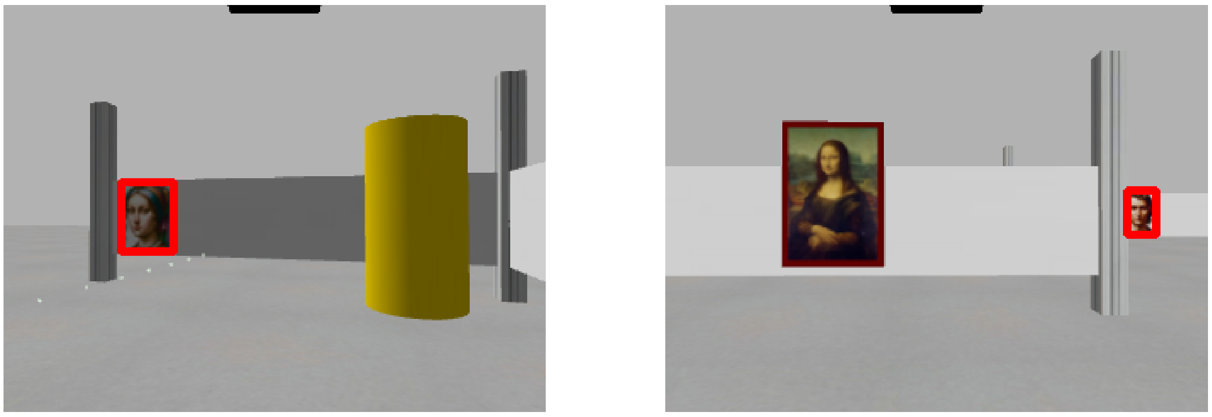

Figure 1: Face Detection

2.2 Ring and Cylinder detection

We used several methods to detect rings and cylinders, including color recognition, image pre-processing, contour detection, filtering to reduce false positive detections and utilized point cloud data. This section details how we combined these techniques to ensure accurate detections.

2.2.1 Colour Recognition

We used the **HSV colour space** to accurately recognize the distinctive colours of rings and cylinders. By defining specific intervals for the Hue channel while keeping the Saturation and Value constant, we create a simple classifier. This allowed us to easily classify each pixel in the image and only focus on the colours we were interested in.

2.2.2 Point Cloud

In processing point cloud data, we first reduce the number of points to speed up analysis and improve accuracy. We use a PassThrough filter to focus only on key areas, removing points that aren't needed. Then using SACSegmentation, we identify and remove flat surfaces like floors and walls. This let us focus on more complex shapes like cylinders and rings. For extracting 3D points of rings and cylinders, we worked with point cloud data in C++, which significantly sped up the process.

2.2.3 Preprocessing

To identify potential cylinders and rings in an image we first need to extract the contours of all objects. The pipeline includes converting the image to grayscale, applying Gaussian blur to reduce noise, and using the Canny edge detector to create a binary mask of all edges. Afterwards OpenCV's contour detection function is used to locate object boundaries.

Once the setup is complete: point cloud data is processed, contours are extracted and assigned color. We can perform further checks based on the type of object.

2.2.4 Ring detection

For a certain contour to be considered a ring, we check if the ring is placed in the upper third of the image and try to fit an ellipse in it. The ratio between the axes of the ellipse should not differ significantly, and its center should be at infinity, using the information from the depth image. If all conditions are fulfilled, we extract corresponding points from the point cloud and average them to calculate the center of the new ring.

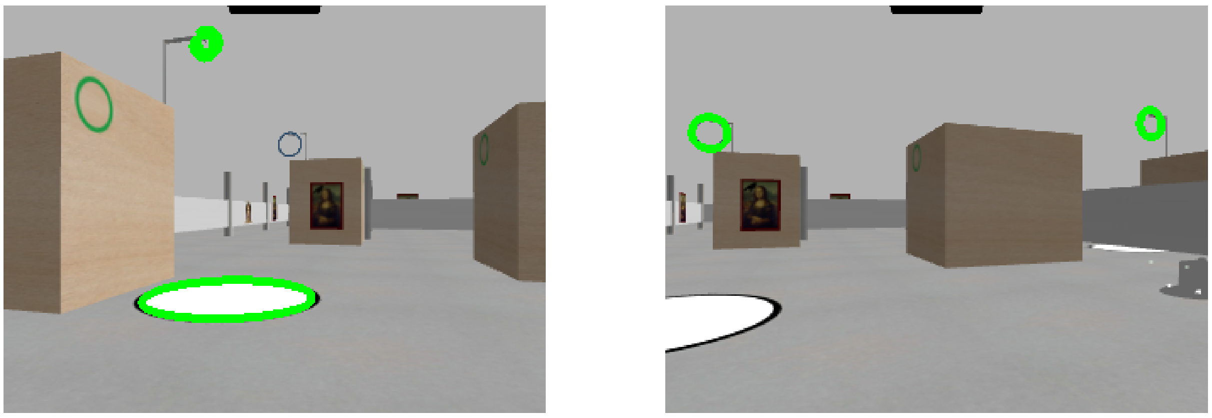

Figure 2: Ring detection

2.2.5 Cylinder detection

For a certain contour to be considered a cylinder, its area must surpass a certain threshold. After that we try fitting a plane using RANSAC: this method involved fitting a plane to the points of the contour obtained from the depth image. Each point's deviation from the plane was calculated, and if the maximum deviation exceeded a predefined threshold, the contour was further processed. The center point of the contour is transformed into 3D.
Next, all possible cylinders in the point cloud within this region are segmented, again using RANSAC but trying to fit cylindrical model. The centers of each detected cylinder are then obtained. If the distance between the 3D point and any cylinder center is approximately equal to the radius of the cylinder, the contour is confirmed as a cylinder.
RANSAC allowed us to robustly segment potential cylinders by considering the inliers of the model fit. This approach helped in identifying and distinguishing cylindrical shapes within the point cloud.

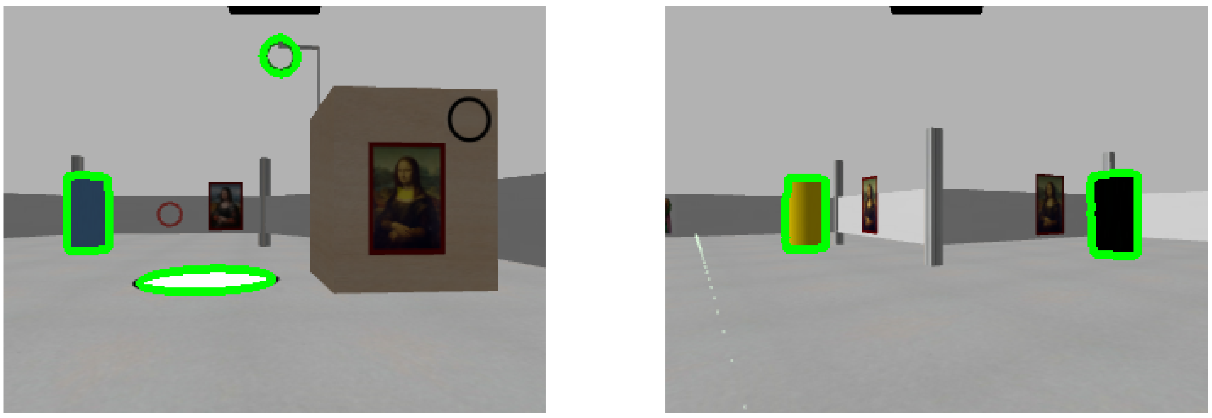

Figure 3: Detected rings, cylinders, parking spots

2.2.6 Handling multiple detections

To eliminate multiple detections of the same object, a clustering approach is implemented. All detected ring centers are stored. If the center of a newly detected ring falls within a predefined distance of an existing center, it is considered the same ring. Otherwise, it is added as a new detection. The same approach is used for cylinders.

2.3 Arm camera

For anomaly detection of the Mona Lisa, locating parking positions, reading the QR code from the top of the cylinder, and waving, we used the arm camera. For each different task, we determined predefined positions for the robot arm:

- **Mona Lisa:** `[0.0, 0.5, 2.3, -1.2]`. The camera is positioned so that it looks directly forward at the Mona Lisa painting. We have also used a higher resolution for this arm camera to better capture potential anomalies (Figure 3).
- **Waving:** Waving is executed by asynchronously moving the robot arm camera left and right based on the coordinates written below:
  - *Wave 1:* `[1.5, 0.5, 2.3, -1.2]`
  - *Wave 2:* `[-1.5, 0.5, 2.3, -1.2]`
- **Parking Detection:** `[0.0, 0.4, 1.5, 1.2]`. For parking detection, we have used a full bird's eye top view so we can apply the Hough transform onto the parking circles (Figure 4).
- **QR Code Reading:** `[0.0, 0.2, 0.3, 2.0]`. For cylinders, we have used an extended bird's eye top view so we can see the QR code clearly (Figure 5).

All the initialization and methods for setting up the arm camera, changing its position, and properties are integrated into the Robot Commander.

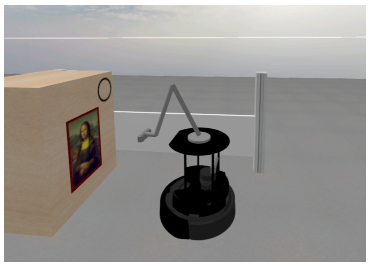
Figure 4: Anomaly Detection Arm Positioning

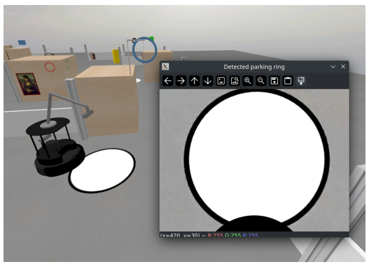
Figure 5: Parking Arm Positioning

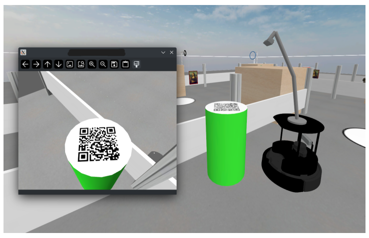
Figure 6: QR Code Scanning Arm Positioning

2.4 QR Code Reading

For the QR code scanning capabilities, we used the robot arm's camera and the Pyzbar library for QR code decoding.

1. **Camera Positioning**: Following the approach to a cylinder, the robot precisely aligns its arm-mounted camera above the cylinder to capture the QR code positioned on its top surface.
2. **QR Code Decoding**: The Pyzbar library decoded the QR code
3. **Data Utilization**: Upon successful decoding, the library extracted relevant data embedded within the QR codes, such as text strings or URLs.

2.5 Parking

Our initial attempt was to use the additional arm camera attached to the top of the robot for detecting parking. When approaching the goal parking spot, we used the Hough transform to look for a 2D ring on the floor. The captured images were converted to grayscale and then passed through a median blur to reduce noise. We applied the Hough Circle Transform to detect circular shapes in the image, which represented potential parking spots. Upon detecting circles, we calculated their bounding boxes and the centers of these circles. We then tried to center the robot based on these coordinates using velocity adjustments.

Unfortunately, this approach proved unsuccessful due to it being slower and causing the camera to lag, and we were not always able to center it perfectly, which is why we opted to navigate the parking using goalPose with a proximity of 0.1 to the parking spot. This method provided very accurate results for our use.

2.6 Speech detection and recognition

The speech dialog in the provided code uses the `speech_recognition` library for converting speech to text and the `pyttsx3` library for converting text to speech. In the `RobotCommander` class, `Recognizer` and `tts_engine` are set up to handle these tasks. The `speak` method uses `pyttsx3` to make the robot say text out loud, while the `listen` method uses `speech_recognition` to listen to what the user says and turn it into text. There's also a `preprocess_text` method that looks at the recognized text to find useful information, like color names.

This speech dialog functionality is primarily activated when the robot interacts with detected faces. After moving to the face, the robot asks the user about the next goal, such as a color for a parking spot. It listens for the user's answer, processes the response in the way that it extracts only the colors from the speech (using the already predefined colors: red, green, yellow, black and blue) while ignoring other unnecessary words. After it gathers information from two people who said two different rings, it takes the intersection of their answers which is the color of the final rings. Robot then proceeds to look for goal ring.

2.7 Anomaly detection

We solved the anomaly detection task in an **unsupervised manner**, training an autoencoder only on normal images.

In developing the autoencoder, we experimented with various architectures, from simple configurations with fewer layers and filters to more advanced setups using skip connections, batch normalization, and variational techniques. The optimal architecture we utilized consisted of four layers, with the final layer having 256 filters. Instead of defining stopping criteria based on the convergence of the loss function, our objective was to intentionally overfit the model to the training data. This was achieved by increasing the number of training epochs to 1000. By doing so, we allowed the model to learn highly detailed and specific representations of the training images.

Training stage: We trained the model on 100 images. We did not introduce a lot of variability in these images. As only changing the brightness, contrast and slight roatations. The uniform training set allowed us to easier train the model.

Evaluation stage: For evaluating the model on the detected Mona Lisa paintings, preprocessing was applied. First, the image paintings were resized to a predefined size. Then, we performed homography, where we transformed the geometrical perspective of the image, and histogram matching, where we adjusted the colors of the image using one reference image. These two steps helped in more easily detecting anomalies in the image. Once the autoencoder reconstructs the image we compare the reconstructed image and the input image. Estimate the difference and define threshold to extract the binary mask.

Here are the results obtained by driving the robot around and analyzing paintings of Mona Lisa. From the reconstructed image we can see that the model has learned the features of the normal images without anomalies, while for anomaly images fails with reconstruction. The binary mask represents the difference between the input image and the reconstructed, based on the binary mask we decide whether the image contains anomalies or not.

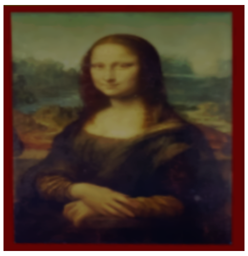
Figure 7: Image without anomalies

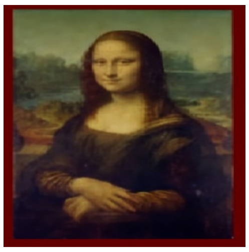
Figure 8: Reconstructed image

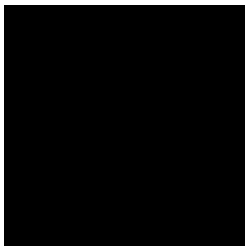
Figure 9: Binary mask

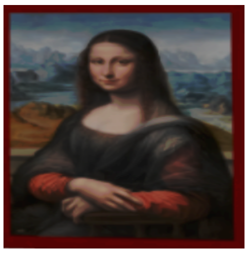
Figure 10: Image with anomalies

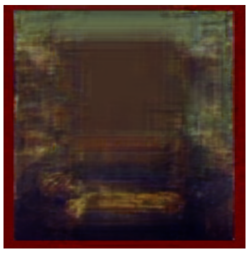
Figure 11: Reconstructed image

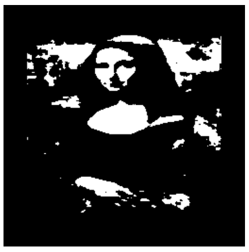
Figure 12: Binary mask

And other examples where the anomalies where successfully detected.

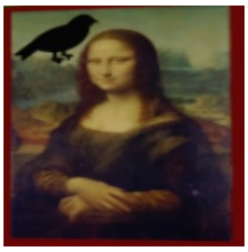
Figure 13: Image with anomalies

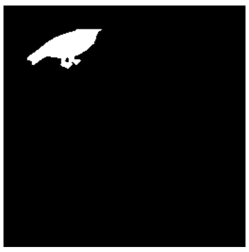
Figure 14: Binary mask

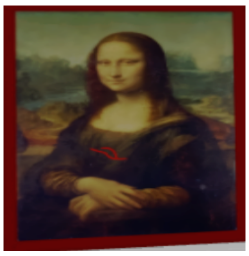
Figure 15: Image with anomalies

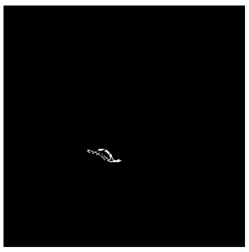
Figure 16: Binary mask

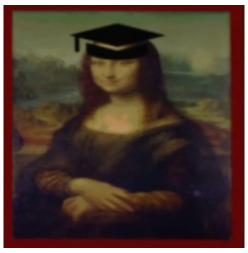
Figure 17: Image with anomalies

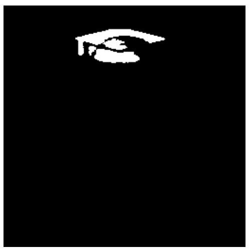
Figure 18: Binary mask

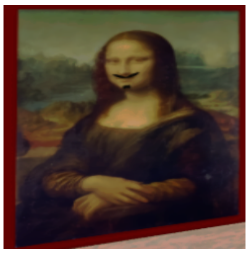
Figure 19: Image with anomalies

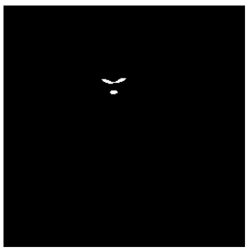
Figure 20: Binary mask

Note: We attempted image augmentation techniques such as cropping, introducing color variability, and increasing the number of training samples. Despite these efforts, the models did not perform better, possibly due to our limited experience and knowledge in effectively training the models. Generally, overfitting should be avoided, but for our object-centric task, where all the images were very similar, it proved to be beneficial.

3 Integration

Given that in the first two projects, we have successfully implemented all the required tasks, we had only minor set ups for the final integration. We had to update our `RobotCommander` so it would include ring, cylinder detection, arm parking camera and Mona Liza anomaly detection. The adjustments we made were:

- Differentiating between Mona Liza paintings and other people in the simulation. For that we used the average color present, paintings of Mona Lisa were darker in comparison to human faces.
- Parking rings were not right below the detected rings, so we had to adjust our logic for the parking. After detecting a goal ring, we went to the closest goal points around the ring until we found the parking spot.

3.1 Robot Commander

The *RobotCommander* node is the heart of our robot's system. It handles moving the robot around and ensuring that all the other nodes and tasks work together smoothly. We divided the robot's tasks into different stages to keep things organized and efficient.

3.1.1 Navigation

Navigation through the map is accomplished using predefined hard coded position points. These points are distributed in a way that covers the entire map. Each time the robot discovers a new face, a new ring the cylinder or a parking spot it should go to, the loop that iterates through goal points is paused, and the intervening tasks are executed. After that, the robot resumes navigating through its goal points.

3.1.2 Robot commander process

- **Movement/Detection Stage**:
  - The robot follows a predefined set of waypoints on the map.

  - It uses its built-in camera to look for faces, rings, cylinders, and parking spots.
  - Utilizes the *detect_rings* and *detect_people* nodes for detection.
  - When a ring, cylinder, or parking spot is found, the *RobotCommander* receives the object's 3D location. Using the *tf buffer*, we transform the new marker's coordinates from the */base_link* frame to the */map* frame and place the transformed point as a marker in RViz.
  - If a face is detected and recognized as a villager, the robot switches to the *approaching* stage.

- **Approaching Stage**:
  - The robot stops further face detection to focus on the current face.
  - Approaches the villager and asks for directions to the right ring for parking. The robot approaches the face using the Euclidean distance between its current position and the detected face's position. This distance determines whether the robot should start approaching the face.

    In the main loop, the robot checks if a face has been detected and if it is not already moving towards another face. The robot calculates an offset based on its current position and the detected face's position to avoid getting too close. This calculation takes different wall orientations into account.

    Once the robot reaches the vicinity of the face, it stops and recalculates its orientation to face the detected face directly using the *math.atan2* function for angle calculation. It then calls the *spin* function to rotate towards the face.
  - Returns to the movement/detection stage after receiving directions.
  - This process repeats until the robot has received answers from two villagers and has gathered the intersection of ring color.

- **Parking Stage**:
  - The robot determines the goal ring based on the common ring mentioned by both villagers.
  - Adjusts its arm camera for parking.
  - If a nearby parking spot is known and is discovered during the detection stage, the robot parks there.
  - Otherwise, it searches for a parking spot by moving to the nearest points around the ring until one is found.

- **Cylinder Detection and QR Code Reading**:
  - After parking, the robot rotates around in place to find the specified cylinder.
  - When a cylinder is detected, robot puts the marker on that position and moves towards it while positioning its arm camera for QR code scanning.
  - It moves close and rotates around the cylinder until the QR code is read.
  - After it scans the QR code, it moves to the Anomaly Detection Phase.

- **Anomaly Detection Phase**:
  - The robot sets its arm camera for anomaly detection.
  - Navigates to specific predefined points while only looking for Mona Lisa paintings.
  - Ignores other objects like villagers, rings, and cylinders.

- **Painting Interaction**:
  - Upon detecting a painting, the robot approaches it in the similar manner it approached to the villagers.
  - By using the arm camera, set to a higher resolution, it captures a detailed image.
  - Processes the image with the anomaly detection algorithm.
  - Marks the image if an anomaly is found.
  - If no anomaly is detected, the robot "waves" at the painting with its arm camera.
  - Moves to the next navigation point, repeating this loop as needed.

By organizing the tasks this way, the *RobotCommander* node effectively manages the robot's complex activities, ensuring it performs well and accurately detects anomalies throughout its tasks.

3.1.3 Markers

In RViz, we are visualizing the markers for each detected object like rings, parking spots, cylinders and detected faces. Each time we identify one of the following objects in the simulation, we use the `TransformPoint` method to transform the point from `/base-link` to `/map frame`. The markers are then created based on the transformed points, using the different sizes and colors for each object.

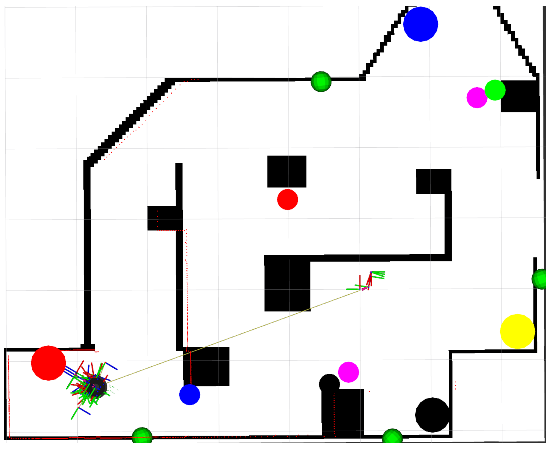

Figure 21: RViz Markers

The topology of the displayed markers:

- ● **3D Green markers:** Detected faces
- ● **Pink Markers:** Display 2D parking spots
- ● **Smaller color markers:** Display each detected ring and its corresponding color
- ● **Bigger color markers:** Display each detected cylinder and its corresponding color

4 Results

4.1 Diagram

In this section, we prepared the diagram to approximately illustrate how our nodes are communicating with each other and what topics are we sending among them

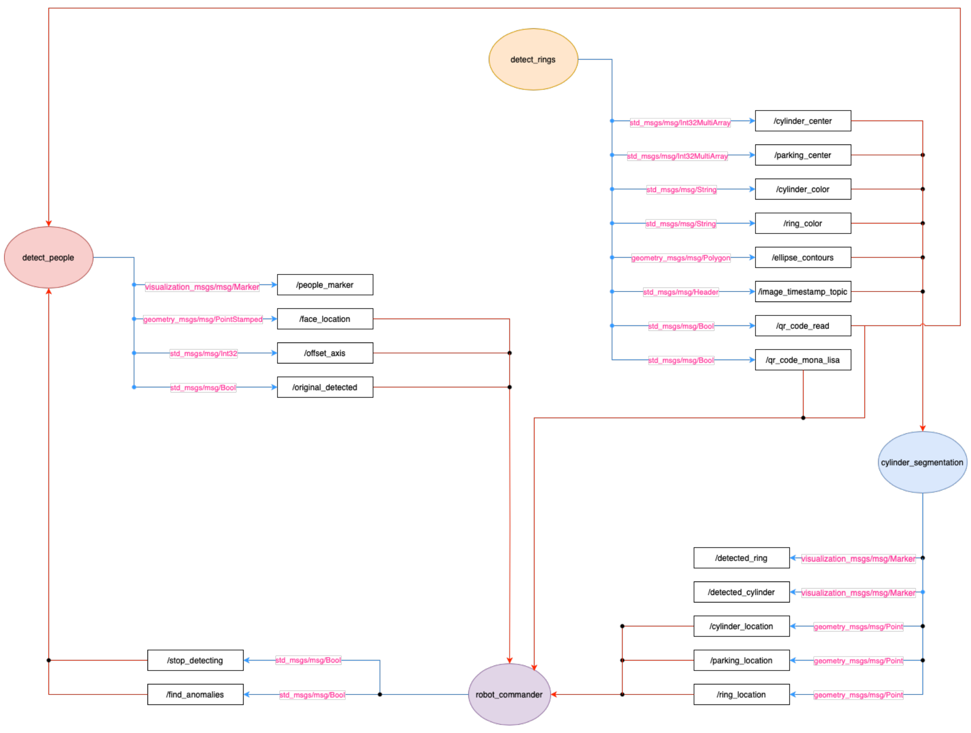

Figure 22: An illustrative diagram

4.2 Evaluation

In the final evaluation, our task performed very well. Among the required criteria, the only one we were unable to fulfill was the autonomous movement of the robot. The rest of the simulation worked flawlessly. The integrated parts were well-coordinated: the robot approached villagers, communicated with them, found the goal ring, parked underneath it, and located the correct cylinder. After that, it proceeded to look for Mona Lisa anomalies, displaying the anomalous ones and waving to the correct, non-anomalous Mona Lisas.

4.3 Experimental results

Apart from the current results, we have tried some other methods for solving particular parts of the task:

- We also experimented with using the **SIFT** (Scale-Invariant Feature Transform) and **HOG** (Histogram of Oriented Gradients) algorithms to calculate feature descriptors for face recognition. However, we found that the results obtained from SIFT were not as accurate or reliable as those from the pretrained AlexNet CNN. The AlexNet model provided more distinctive and robust feature descriptors, leading to improved accuracy in identifying new faces and distinguishing between previously detected faces.

- For anomaly detection, we tried constructing a **PCA** space and using **Normalized Cross Correlation**. However, PCA was not the most effective method, as it introduced significant noise into the reconstructed images. To account for this noise, we applied median filtering and then identified the differences. The Normalized Cross Correlation performed better but was again harder to properly identify regions with anomalies. Nonetheless, the best results were achieved using Autoencoders.

- We started implementing **autonomous navigation** by first optimizing the path through predefined goal points using a nearest neighbor heuristic, ensuring minimal travel distance.

The navigation algorithm operates on a grid consisting of 1s and 0s, where 1s represent obstacles and 0s represent free space, allowing the **A* algorithm** to accurately plan paths by avoiding obstacles. For detailed path planning between consecutive points, the A* algorithm is employed, leveraging a heuristic based on Euclidean distance to find the shortest path while considering traversal costs and obstacle positions. Although the algorithm may not be fully autonomous, it worked well for the robot's basic movement. However, when additional tasks were introduced, the movement was not as smooth and efficient as it was when using only predefined points with goalPose. That is why we stuck with that in our final evaluation.

5 Division of work

We are confident that our team performed well together, as each of us equally contributed. Andjela had the best understanding of operating the physical robot and integrating different parts of code, eg. including arm camera and execution of its tasks. Trajche provided insightful observations about navigating the robot, task planning and ring detection. Biljana brought a strong understanding of computer vision that was crucial for face, anomaly detection, and working with point cloud data. Throughout the project, we were only able to run simulations on the lab laptops, which was a factor in enforcing us to work together most of the time. Each of us proposed various ideas on how to solve problems and assisted each other in debugging, even the smallest errors.

In the table below, it is a rough estimate shown who contributed to which part of the task the **most**.

Task	Biljana	Trajche	Andjela
Face detection	✓	✓	
Ring detection	✓	✓	✓
Cylinder detection	✓		
Navigation logic		✓	✓
Real life robot			✓
Speech detection		✓	✓
Anomaly detection	✓		
QR reading		✓	

Total approximate contribution of each member:

- **Biljana:** 34%
- **Trajche:** 33%
- **Andjela:** 33%

6 Encountered problems

Most of the problems that we encountered during our project, and what took most of our time, were due to hardware difficulties. The simulations did not always compile on the first try, Gazebo and RViz did not always work flawlessly, and the real-life robot also faced several challenges.

Logic and implementation problems we faced in the conversion between frames, using PointCloud, and using the C++ language, which we had not worked with before. The lag between the frames in Python was another issue. Anomaly detection also gave us a hard time.

Not being able to work from our laptops at home made the whole process much harder, leading us to work in the faculty laboratories for a large part of the week and often during the weekends.

7 Conclusion

To conclude, we believe that we have successfully implemented all the requirements.

Although this task maybe is considered as simple, it provided a good starting point for us to learn what programming a robot involves, how many factors need to be considered, and how debugging in dynamic environments can be quite challenging. The project took some time, but we understand that acquiring new skills requires time. Looking back, we learned a lot and believe it was well worth it.

What we appreciated most about the course was the focus on hands-on experience, rather than just going through theoretical material without understanding it. We also liked that we could apply the knowledge gathered from other courses to real tasks.