University of Ljubljana
Faculty of Computer and Information Science
Development of intelligent systems

Final report

Team Sigma

Anže Štritof
Ana Mangravoska
Anastazija Kovachevikj

Ljubljana, 2023

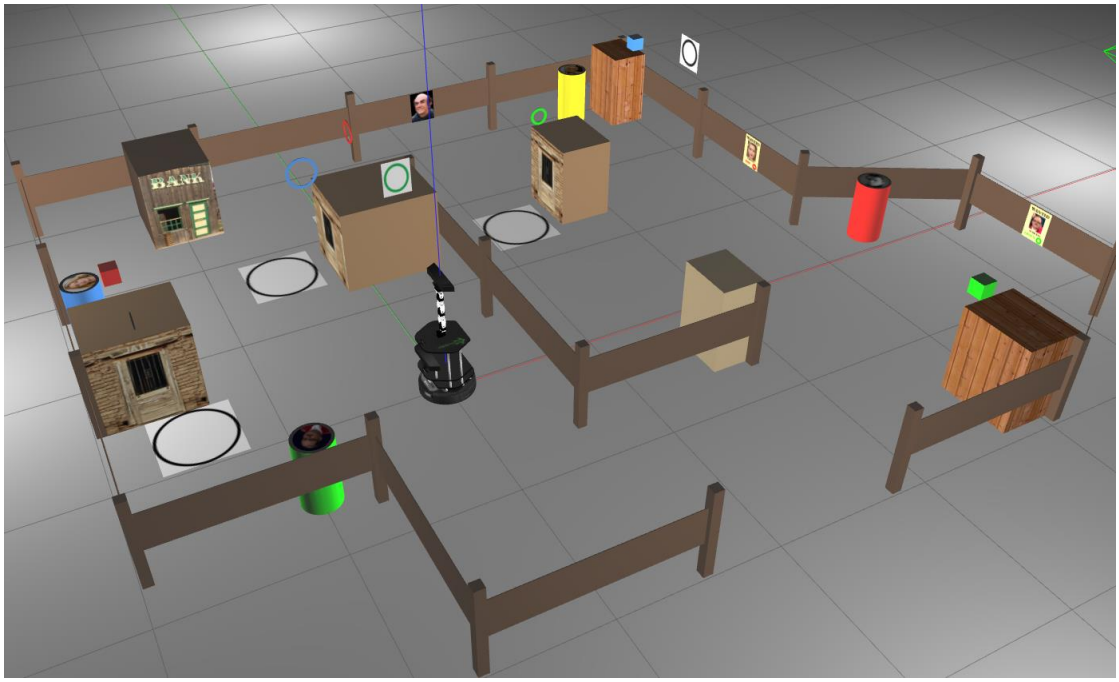
Contents

1. Introduction

Throughout the "Development of Intelligent Systems" course, we explored various aspects about robots, how they work, and how they interact with their surroundings.
For this project we had to develop a robot using the Robot Operating System (ROS). The objective was to enable our robot to move autonomously while also recognizing its surroundings so that it can accomplish its mission of finding and imprisoning the most wanted robber in the polygon. The polygon includes cylinders, rings, faces and wanted posters. When the robot detects a wanted poster, it can recognize the robber's face and obtain information about the offered prize as well as the color of the prison to which the robber should be brought. Cylinders symbolize possible hiding places for robbers, whereas rings represent prisons. While detecting the objects the robot should also detect their color. Lastly the faces represent civilians which need to be approached and interrogated about the robber's whereabouts.

In the next chapters, we will discuss the methods and algorithms we used to accomplish our goal, how we implemented those methods and the obstacles we encountered.

Figure 1: Task polygon



2. Methods

The methods we used to complete the tasks and the theoretical principles they were based on are explained in the chapter that follows.

2.1 Ring and Cylinder detection

For this part we expanded the template provided to us during the course.

2.1.1 Ring detection

We achieve ring detection by preprocessing the arm camera image and arm camera depth image to improve detection accuracy. Following that, we use the cv2 library to detect contours in the image, map any elliptical shapes to the image, and then provide additional logic to determine if it is a torus before subsampling for color.

2.1.2 Cylinder detection

The overall technique for this part was to use the Kinect depth camera point cloud representation along with PCL (point cloud library). We preprocessed the input to reduce the size of the cloud and speed up the processing. On this we then calculate the normals, group planar inliers, and use this information to check for potential cylinders. We then subsample the detection for color.

2.2 Face detection and recognition

2.2.1 Face detection

Face detection is a general method that we perform on regular faces and posters, as well as the faces on the cylinder. The approach is to take an image, process it with a neural network, and determine whether the confidence of our custom deep neural network is greater than a given threshold of an image that contains a face.

2.2.2 Poster detection

After ensuring that the robot is looking at a face, the following step is to see if the image appears on a wanted poster. By analyzing the leftmost and rightmost padded pixels of the image for color, we may draw the conclusion that the robot is facing a wanted poster if the color is a shade of yellow. After this is confirmed, we can perform facial embedding, which involves running the face on the poster through the dlib library. We use this to detect facial features using a trained custom face feature prediction model, and then using those detected features to run through another model that transforms that information into a vector. We need this for comparing faces on cylinders to the posters.

2.3 Text recognition

The way we decided to tackle this is by using the Python-tesseract optical character recognition (OCR) tool whose job is to recognize and "read" the text embedded in images. OCR uses sophisticated algorithms and machine learning techniques to find and recognize text regions within an image and then transform them into machine-readable text data. It does this by analyzing the visual properties of the text such as shape, size and spatial arrangement.

2.4 Speech recognition

To enable speech recognition our system uses the Google Speech-to-Text API which is designed to convert spoken language into written text. The library works in a way that it analyzes speech-containing audio inputs and recognizes the spoken words using machine learning models and algorithms. To provide accurate transcriptions, it considers numerous elements such as linguistic context and background noise. Lastly as output, the API returns the transcribed text.

2.5 Autonomous exploration

The goal was to create a robust system that will allow the robot to perform its goals regardless of any map changes. We achieved this by relying on internal map information obtained from ROS services and topics, to have an idea which points are unoccupied, then implemented K Means Clustering, to generate the desired number of goals within the points that are unoccupied. Because establishing a path through these points is a computationally difficult issue (see Traveling salesman problem), we implemented a basic greedy algorithm that always selected the nearest goal, with the extension of line of sight checking penalty.

3. Implementation and integration

The robot starts in search mode, the point of which is to move through the polygon autonomously and detect as many posters, rings, and cylinders as possible, as accurately as possible. In our implementation we want the robot to find everything, before we move onto the cylinder approach.

After finding all the posters, and letting the masternode know who is the most wanted, and where his prison is, the robot can also approach civilians (regular faces), at which point we expect at least one of them to give some information about the whereabouts of the highest bounty criminal. Once all the elements have been discovered, we lower the arm, and initiate the apprehension procedure. Which involves us approaching the cylinder we think the robber is hiding on, comparing his face embedding to the target, and if it's not the optimal target, and the civilians lied to us, we check the next potential cylinder. We repeat this until the correct person is discovered (relying on high cosine distance value), at which point the robot will move him, and himself to the correct ring, and initiate parking procedure.

3.1 Ring and Cylinder detection

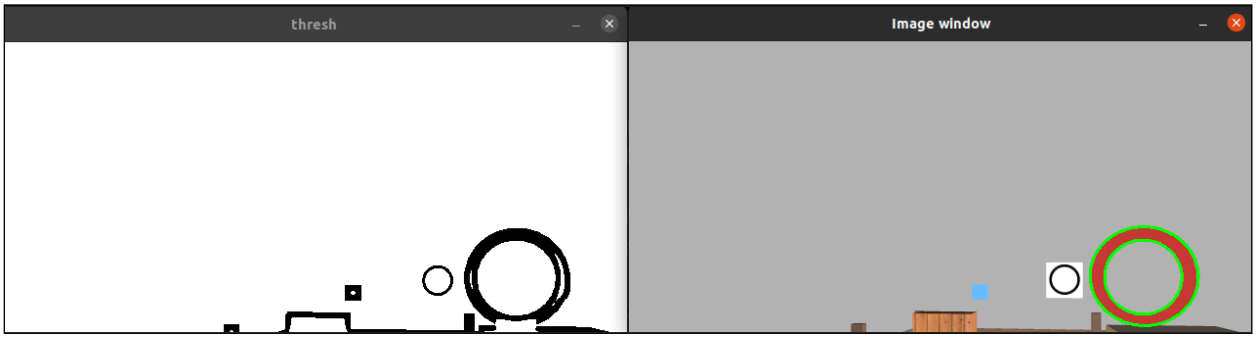
3.1.1 Ring detection

Firstly, we extract a smaller upper part of an image because the elements on the ground do not interest us. On this we then do preprocessing, contrast boosting, and mild brightness reduction, conversion to grayscale, grayscale histogram equalization, and then run cv2.adaptiveThresholding.

When contours are discovered, using the cv2 library, we try to map ellipses to them. If these ellipses share the approximately same center [<0.5] we can be almost certain that we are looking at a torus. To distinguish it from printed paper we perform additional checks. If the center has a measurable depth or the background in the middle is the ambient gray we know that it's a torus.

To improve accuracy of the detection we rely on color based non-maxima suppression (NMS) which greatly improved the location and color accuracy of our detections.

Figure 2: Detected ring



3.1.2 Cylinder detection

Figure 3 : Parameters used for cylinder detection

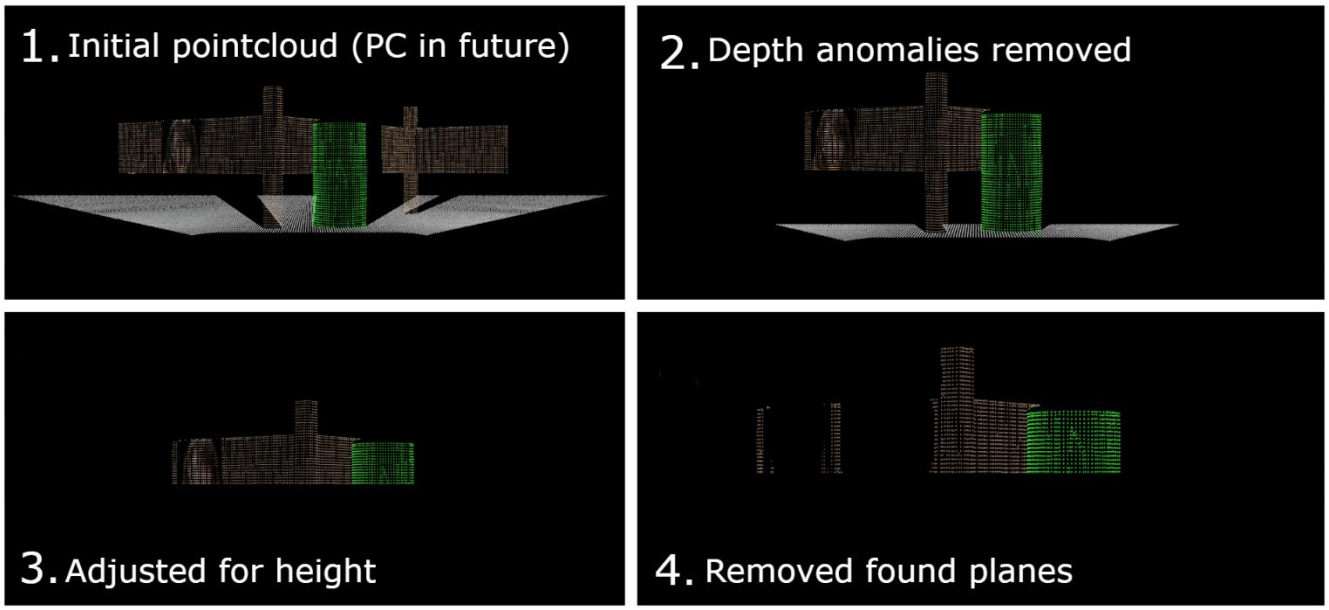
```
// 1) segmentation for planar model
float plan_normal_distance_weight_ = 0.4;
int plan_max_iterations_ = 70;
float plan_distance_threshold_ = 0.03;
// 2) segmentation for cylinder model
float cyl_normal_distance_weight_ = 0.3;
int cyl_max_iterations_ = 10000;
float cyl_set_distance_threshold_ = 0.020;
float cyl_rad_min_ = 0.11;
float cyl_rad_max_ = 0.13;
```

We improved upon the baseline code by first experimenting with some parameters and in the end we found the values worked well with our robot.

To make it more reliable we added some preprocessing to it. Besides removing spurious NaN values, we also removed the bottom part of the point cloud, since this guaranteed we will not detect cylinders on the other side of the wall, and that the marker placements are going to be in more open areas.

To this base we then added NMS, the requirement being 12 accurate detections before we are certain it is an accurate detection. In the end to make execution faster we created a launch file called combined.launch, which merged detect_rings, and added voxelgrid downsampling as the input to our cylinder detection.

Figure 4:Stages of point cloud visualization used for cylinder detection



3.2 Face detection and recognition

3.2.1 Face detection

Using the color image from the Kinect camera and the cv2.dnn approach in combination with the caffemodel network, we obtain information on potential face detections, including how confident the network is in the detection. This confidence represents our threshold for accepting a detection as a face. Since the majority of face detection is done in this manner, the only other addition to this section is the approach to the face, which is detailed later.

3.2.2 Face recognition

Face recognition is used in two tasks: poster detection and cylinder approach. We use the bounding box obtained from the face detection algorithm to extract key features from the face using the dlib library and a custom deep neural network [1]. With these features obtained we can fix the orientation of the image with dlib.get_face_chip and then use these two information points, with another deep neural network [2] to create an embedded vector representation of that person's face. Then we can send this to the master node in case it is a poster or compare to the target face if we are in the cylinder approach phase. It is worth noting that dlib.get_face_chip is especially useful with cylinders because we can scan the face from a variety of angles. These embeddings are unit vectors of 128 floats that we compare using cosine distance. We consider a face to be a match if the cosine distance is greater than 0.75 .

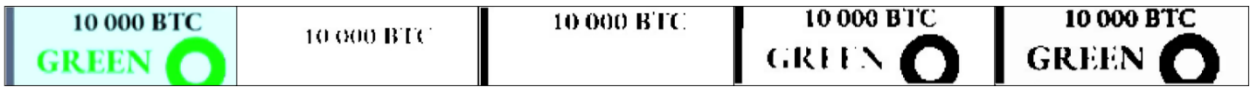
3.2.3 Poster detection

The first step is to determine the shade of yellow (the color of the wanted posters) we are looking for. To obtain the color around the face image, we first resize the image region, after which we compute the means of the top left and bottom left pixels, as well as the top right and bottom right pixels. This method is used to determine whether the face belongs to a poster or a civilian.

3.3 Text detection and recognition

The way we begin our approach is to first make sure that the robot is facing a wanted poster which is not yet marked. Once this condition is met, we first extract the section of the poster that contains information about the offered price reward and color of prison that robber should be taken to. OCR is applied to the processed face region to extract the text information. After this we apply techniques such as thresholding, pattern matching (FuzzyWuzzy library) and text analysis to retrieve the correct information from the poster.

Figure 5:General progression of increasing threshold values applied on leftmost image



3.4 Speech recognition

We used the SpeechRecognition library as well as the previously mentioned Google Speech-to-Text API for this section. For easier determination of possible hiding spots (colored cylinders) we predefine an array of possible colors, as well as an array of possible negation words. After we receive a response from the API, we store it as a string and iterate through it, searching for matching words in the arrays. Lastly colors with negation get stored in the invalid_colors[] array, whereas the other colors are stored in the valid_collors[] array.

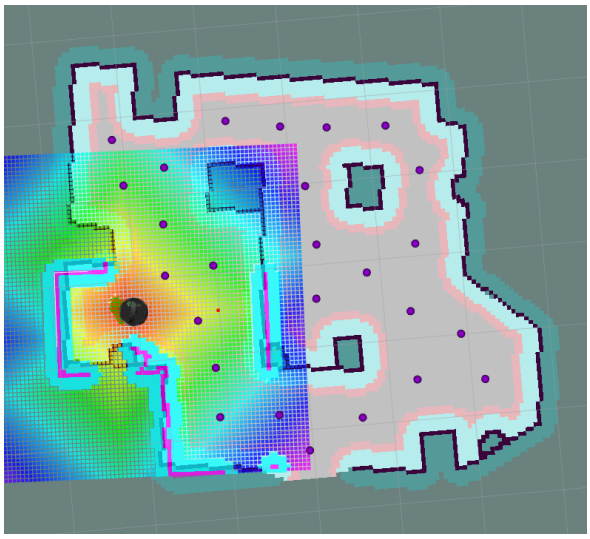
3.5 Autonomous exploration

The implementation of autonomous exploration uses four major components:

1. First component is the obtaining of all the initial state information, from /get_map service and from /move_base/global_costmap/costmap topic. We use this information as the input into our KMeansClustering implementation.
2. We use KMeansClustering to cluster all available unoccupied occupancy grid points into clusters, where their centers represent our goals.
3. Once the goals are created, we have to create a sensible path through them. However, since this resembles the traveling salesman problem, constructing an efficient algorithm for a large number of cases is challenging. Instead of calculating all possible permutations, our implementation prioritizes being autonomous and independence from human intervention. Since larger maps might require selecting 100 points, utilizing more complex algorithms would significantly slow down the process. Thus, we opt for a greedy solution that's "good enough" at small numbers of cases which we then expanded.

4. While planning our path we select the closest unvisited node from the remaining options. However, when calculating distance, we also apply a penalty for points that lack a line of sight from the previous node. This penalty is calculated as (euclidean_distance + 1) * 1.5. This approach works well on relatively small distances and relies on the line-of-sight checker which is a crucial component for the success of this implementation.

Figure 6:Rviz representation of created goals using KMeans



3.6 Color recognition

Figure 7: Formula used for color recognition

$$\bar{r} = \frac{C_{1,R} + C_{2,R}}{2}$$
$$\Delta R = C_{1,R} - C_{2,R}$$
$$\Delta G = C_{1,G} - C_{2,G}$$
$$\Delta B = C_{1,B} - C_{2,B}$$
$$\Delta C = \sqrt{\left(2 + \frac{\bar{r}}{256}\right) \times \Delta R^2 + 4 \times \Delta G^2 + \left(2 + \frac{255 - \bar{r}}{256}\right) \times \Delta B^2}$$

Color recognition is handled by transforming BGR images to RGB. We subsample a bounding box (in the case of rings) or a color information point cloud (PointXYZRGB in the case of cylinders). For rings, we first detect the majority color and then mean it to construct an accurate representation of the object's overall color. For cylinders, we further randomly subsample the detected cylinder, and mean the color from the points that are left in the point cloud. Then we hold a dictionary of several known RGB values, mapping these into different strings. In the case of rings we use the Webcolors library, whereas in the case of cylinders we had to manually create such a dictionary as no suitable library was found. We calculated the distance between colors using the "redmean" method [3]. This by itself was an accurate enough approach, and works especially well with a larger, predetermined map such as the Webcolor library, because then functionality can be expanded. We added a simplification layer to the output from the Webcolor maps, since that way we could guarantee color names that were more familiar.

3.7 Approaching face, posters, cylinders and parking

All approaches are done using a service-client relationship call that our map_goals_task2 node allows. This is good because it halts other processes the master node is currently dealing with, and also stops the client program, until the request is resolved. When such a request is published we allow for the service node to find a more suitable goal if it determines that the request is within the occupied territory of the global costmap. If the location is deemed occupied, it will add a padded value to it in different directions in steps of 0.05 until it finds a valid location.

3.7.1 Approaching faces and posters

We approach faces in two phases, the initial approach stage, mainly to reduce the distance to about ½ of the original distance, and the improvement stage, which rotates the robot to improve the facing direction. Because of the implementation choice of allowing the service to pick a better goal, we have the second stage to improve orientation after the translation.

3.7.2 Approaching cylinders

We use the same internal goal issuing logic, that checks the desired location, with the global costmap, to get us closer to the target. We then use cylinder_center_detector to turn the robot around until it discovers the desired published color of our cylinder. We scan across it to determine at what point the central depth camera value is the smallest, and consequently start moving towards it. We first rely on depth camera information to approach carefully using linear twist messages, but since it produces NaN values under distances of 0.4, but we still need to get closer, we have a short timed slower twist message approach, to safely get us closer. (measured from the minimum acceptable distance)

3.7.3 Parking

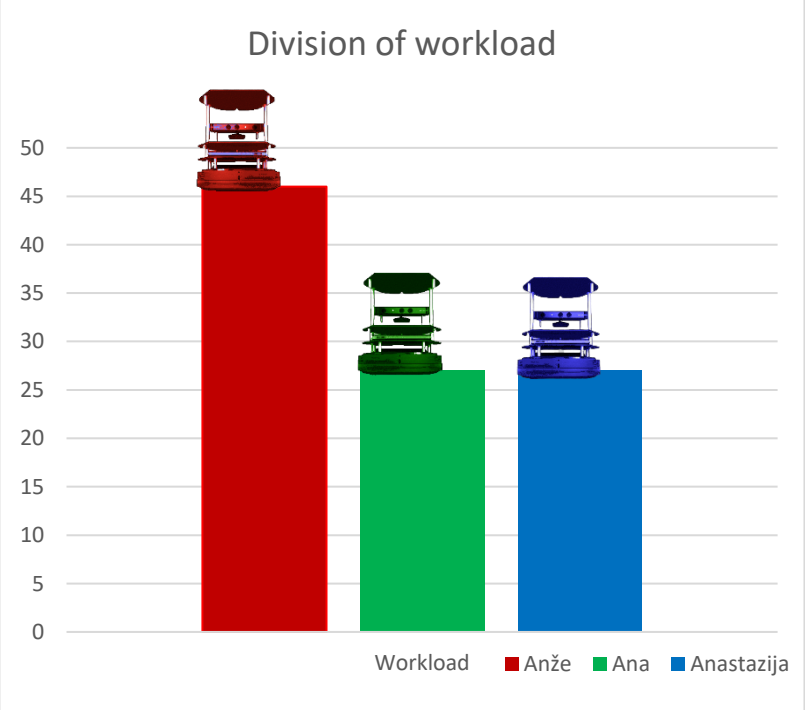
The initial approach is the same as the other two cases. When we reach the goal, we initiate the detect_floor node, which is responsible for trying to map a circle to an object on the ground. It controls the robot by spinning it around and seeing if it can detect any circle, using the same basic approach as with the ring detection. If it cannot after a full turn, it will stop. Because our ring detection and rough approach are accurate enough, it is not uncommon for the robot to be parked sufficiently well with these two elements by themselves. In the opposite case, it will approach the detected goal, and turn, until it finds the center of a circle on the ground, and slowly move forward, until it finds the front edge of the printed ground circle, where it stops, knowing it has parked successfully.

4. Results

As with any project, the road to a fully operating robot sheriff had its bumps. One of the issues we had early on was the autonomous navigation of the robot which in the end we fixed with implementing KMeans clustering and penalties. Another challenge was the accuracy of the OCR text recognition which we fixed with multiple samplings and a pattern matching.
We should mention that one area of potential improvement for us is the Google Speech-to-Text API which even though it should be able to recognize different dialects we sometimes had the issue of the API returning the wrong words.
Despite these setbacks, we had little issue implementing the functionalities for ring, cylinder, and face detections.
The following video-link demonstrates how the robot works: https://youtu.be/LeqyEPJ0Fko

5. Division of work

Even though we had just recently become colleagues, we collaborated on this project very well. During the whole process there was not a clear separation of the workload for some parts. Anže was responsible for the robot's ring and cylinder detection, color recognition, parking, and overall synchronization. Anastazija and Ana, on the other hand, focused on the implementation of speech and text recognition, as well as poster recognition. Where we collaborated, most was the autonomous navigation, along with face detection. If we have to estimate the work division we would give 46% to Anže, 27% to Anastazija and 27% to Ana.



6. Conclusion

Overall, we are extremely pleased with what we were able to achieve throughout the duration of this course. Although we encountered some challenges along the way, we were able to implement all the necessary requirements in the end and get the robot working almost perfectly. We must acknowledge that achieving synchronization among all the functionalities was quite a challenge, but the biggest obstacle was that some of the team members lacked the necessary prior knowledge which sometimes made the workflow more difficult. Nevertheless, we all agree that we have gained a lot of experience and insight into ROS which we intend to develop further.

References

[1] Deep neural network (face landmarks) https://github.com/MortezaNedaei/Facial-Landmarks-Detection

[2] Deep neural network (face recognition) https://github.com/ageitgey/face_recognition_models/blob/master/face_recognition_models/models/dlib_face_recognition_resnet_model_v1.dat

[3] "Redmean" method https://www.compuphase.com/cmetric.htm