31391 Software Frameworks for Autonomous Systems

Semester Project

Authors:

Student Name	*Student Id*
Anders Thuelund	s164057
Hans Kristian Bech Hansen	s164025
Mads Frimann Madsen	s134109
Mustafa Sidiqi	s153168

November 15, 2021

1 Introduction

This report is part of the documentation of the semester project in the course *Software Frameworks for Autonomous System*. The projects main focus is to design and develop software for a simulated autonomous robotic system, which is capable of navigating in a predefined environment and identifying randomly placed targets. The targets are in the form of *QR codes*, where each QR code contains part of a secret message. A successful outcome of this project would be that we were able to find the secret message. The secret message is found by combining the letters, associated with each QR code, in the correct order. There is a total of, $N = 5$, QR codes which are scattered about a virtual classroom setting. The software is written in python and will use the Robot Operating Software (ROS), software libraries to build our autonomous robot application.

2 Details

This section contains a small description of the setting in which the robot operates and how the QR codes work.

2.1 Operational Space

The operational space in which we are to locate these QR codes is a classroom, see figure 1, with the following objects.

- Tables and chairs (Static position)
- Obstacles (Dynamic position, changes every time you start)
- QR Codes (Dynamic position, changes every time you start)

Figure 1: Scene of the project

2.2 QR Codes

Each QR codes holds the following information.

- X, Y Coordinates of the QR target
- X. Y Coordinates of the next QR target
- The QR target ID (1 - 5)
- A letter associated with the QR target, once all target have been examined, the letters construct a word.

Example of data from a QR target:

```
Data: "X=2.35\r\nY=3.24\r\nX_next=5.3\r\nY_next=5.9\r\nN=3\r\nL=M"

Simplified Data: "X=2.35, Y=3.24. X_next=5.3, Y_next=5.9, N=3, L=M"
```

Figure 2: The robot and targets (QR Code)

3 Project Implementation

3.1 Project Setup

The navigation's backbone is an offline map of the operational scene. The map was created by manually moving the robot around the operational scene, using the teleoperate function, creating the map seen in figure 3. How this map is used will be explained in section 3.6.

Figure 3: Mapped area of the operational scene

3.2 Initialization

Starting the code implementation we define a number of variables and functions that are absolutely necessary for the workings of the program. The variables which we define are for storing the information we get from the QR codes as well as for controlling the movement of the robot.
We then go on defining three callback functions, one for the laser scanner, one for the code reading capabilities and one for the object position. We will go further into some of these functions later in the report. Then we have a function called goal_pose(x,y,z,dir), which generates geometry_msgs.msg pose. This function is called in the move_to(x,y,z,dir) which is the function we have created to actually move the robot to a specific location when we have found the transformation from the hidden frame to the odometry frame. Lastly we have made the kabsch(A,B) function which we use to find the transformation from hidden frame to odometry frame. This we will explain further later in the report.
As said above we have defined three callback functions, which are used for the following three subscribers.

- scan_sub = rospy.Subscriber('scan', LaserScan, scan_callback)
- rospy.Subscriber('/visp_auto_trackercode_message', String, code_callback)
- rospy.Subscriber('visp_auto_trackerobject_position',tf2_geometry_msgs.PoseStamped, vispObj_callback)

Notice that we here use the tf2_ros library instead of the tf library, which is a nice package for the transformations we make. We also have the following publisher which we use for moving the robot in wander mode.

- cmd_vel_pub = rospy.Publisher('cmd_vel', Twist, queue_size=1,latch = True)

Following all of this we then initialize the ros node and then the "real" part of our program can begin.

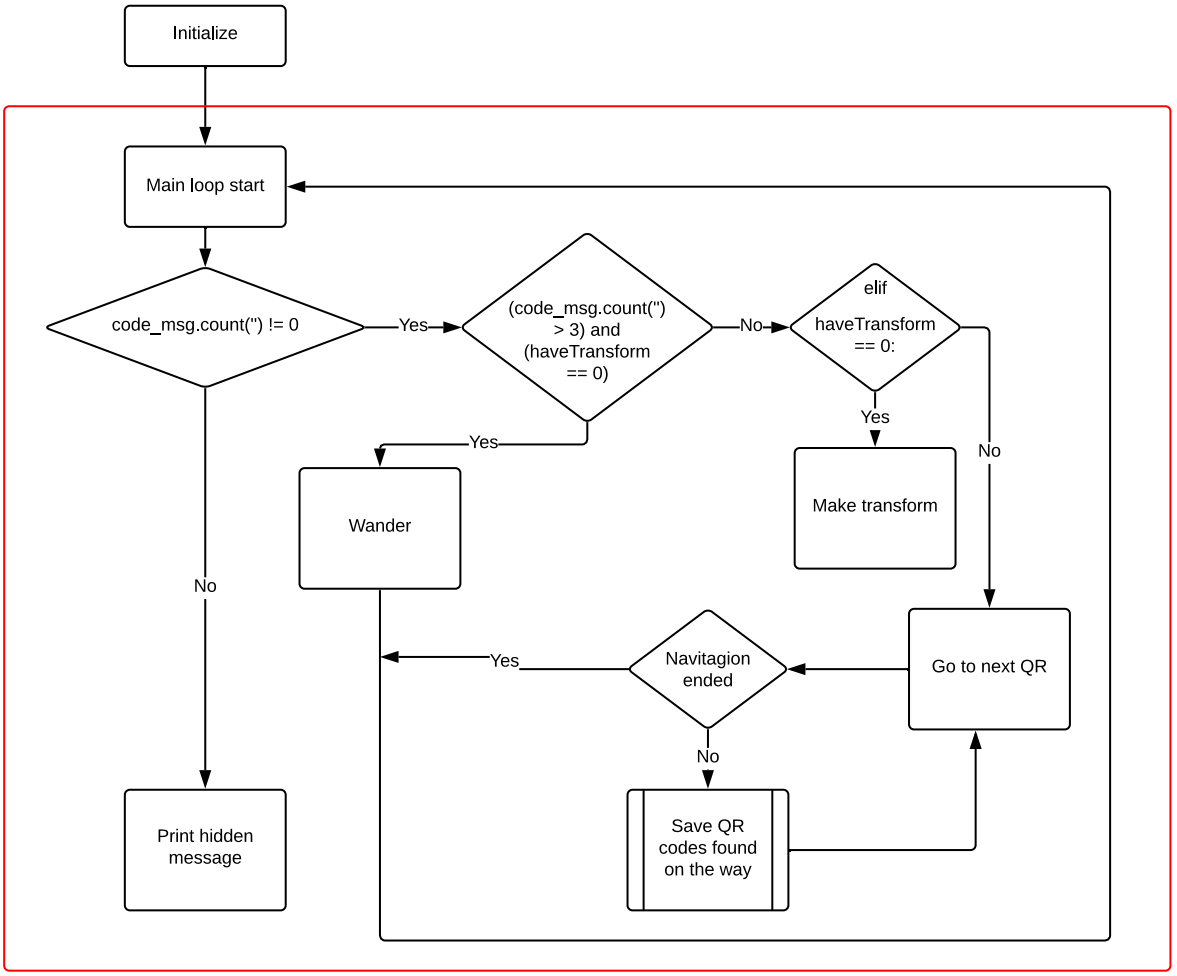

Figure 4: Simplified flow diagram of the main loop

3.3 Wander

The robots starts off by "wandering" around the room. It continues to just drive around aimlessly until it spots a QR code. Then it proceeds to read the QR code. After finding two QR codes the wander can stop and a transform be made.

The wandering is implemented using the robots build-in laser scan, in order to avoid being stuck or just driving into a wall. By subscribing to the laser scan, the distance to any object in front of the robot is found. Whenever this distance is less than 0.8 meters, the robot stops and turns until there's nothing in front of it, and then goes back to drive forward. This piece of code keeps running until we have found two QR codes. It will at random turn left or right around it self when getting to an obstacle. This is important to keep the time the wander takes down, as it could easily find a path around and around where it would not find any QR codes.

3.4 QR code reader

The reading of the QR codes is done using the ros-melodic-visp-auto-tracker*, which is a ROS package created for scanning reading QR- and flash-codes with the robots build-in camera. This package also contains computer vision algorithms for position and orientation estimation of objects in images. For this task QR reader capabilities, which to get the information contained in the QR codes as well as the

computer vision algorithms for position and orientation estimation of the QR codes. This part is taken care of for us, we simply have to use the qr_visp.launch file provided to us. The readings are published to the the topics /visp_auto_tracker/object_position and /visp_auto_tracker/code_message. We can get the information needed about the QR codes from subscribers to these topics.

When a QR code is spotted, the robot stops and reads the QR code. The message is then saved into an array, which is created to save all the messages, so they can be utilized later. At the same time, pose estimation of the QR code in the camera_optical_link frame is found and this is then transformed into a pose given in the odometry frame. This transformation is found using the transform function found in the tf2_ros package.

After the first QR code is scanned, the robot goes back to "wandering", since the transform from the odometry frame to the hidden layer can only be determined after 2 QR codes have been scanned. After scanning the second QR code, we have the information necessary to calculate the location of the next QR code, given in the odometry frame can be found and the robot is navigated there. It continues to do so, until all QR codes have been scanned and added to the array containing all the QR messages.

The array with the QR messages is used for a couple of things. Firstly is it used to detect whenever a spotted QR have already been scanned, in order for the robot not to scan the same QR code all 5 times. It is also used, as stated earlier, to find the next QR code, since the scanned QR code contains the location of the next QR code, in respect to the hidden QR frame. Lastly it also contains the coded message. However this is also saved in another array, which makes it easier to display it in the end.

3.5 Navigation of robot

For navigation, the robot uses 3 different maps: A pre-scanned map of the room, the global cost map and the local cost map. The pre-scanned map is created, by driving the robot around the room, without any obstacles or QR codes, by teleoperating the robot. The map is then saved, so the robot can call it for the "real" run around the room. Using this offline, static map, a global and local cost map can be found. These cost maps are calculated using the default values for the inflation and are used for navigating the robot, and therefore used to plan the route for the robot. The cost maps are also continually updated using the laser scan, in order for the robot to reach its destination while also taking the shortest path.
The difference between the two cost maps are their operating area and resolution. The local cost map provides a local navigation path and are updated more frequently. The global cost map provides an overall global direction while only being updated when the robots detects a new object. Both cost maps are however used interchangeably, which improves the robots navigation.

3.6 Transformation from hidden frame to odometry frame

The transformation it self is found using the Kabsch algorithm, link, which is used for finding an optimal rotational matrix between two sets of points. The sets needs to be paired meaning that both sets contain the positions for the same points but given in two different reference frames. For us to be able to use Kabsch algorithm we need the positions of at least two different QR codes given in both the hidden QR frame and in the odometry frame. putting this in to two matrices we have the information.

$$Q = \begin{bmatrix} x_{QR1,hidden} & y_{QR1,hidden} & 0 \\ x_{QR2,hidden} & y_{QR2,hidden} & 0 \end{bmatrix} \quad P = \begin{bmatrix} x_{QR1,odom} & y_{QR1,odom} & 0 \\ x_{QR1,odom} & y_{QR1,odom} & 0 \end{bmatrix}$$

Here Q is the position of the first and second QR codes in the hidden reference frame and P is the position of the same QR codes but given in the odometry frame. The algorithm then consists of the following steps.

1. Translate the points in Q and P so that their centeroids coincide with the origin of the coordinate system in which they are defined. Their centers are stored in $centQ$ and $centP$. This corresponds to subtracting the mean of each column from each column.
2. Compute the covariance matrix, $H = P^T Q$
3. Compute the singular value decomposition, $H = U\Sigma V^T$
4. Compute the $[3 \times 3]$ rotation matrix, $R = V^T U^T$
5. Determine if we need to perform a correction to our rotation matrix so that we have a right handed coordinate system, in case of mirroring.
6. Calculate the $[3 \times 1]$ translation matrix, $t = -R\ CentP + centQ$

With the rotation matrix R and the translation vector t we can compute the the next QR code as.

$$QR_{next_odometry} = R\ QR_{next_hidden} + t = R \begin{bmatrix} x_{QR_next_hidden} \\ y_{QR_next_hidden} \\ 0 \end{bmatrix} + t \tag{1}$$

3.7 Finding the remaining QR codes

In order to move the robot to the location of the next QR code, the robot utilizes the move_to function, which uses the navigation described above. However this function doesn't just move the robot to the desired location. When the robot moves toward a goal location, it scans for QR codes on the way. If the robot spots a QR code it haven't scanned before, the robot saves it. When reaching the goal or being unable to, it will jump back and set a new goal now use these new information. We proceed in this fashion until we have found all five QR codes, we then print the secret message and the program terminates. It is implemented that the robot is not moving to the location of the QR codes but a bit closer to the center of the map. This have been introduced because we are not wishing to go to the QR code but to scan it, there for a bit of distance is want we need.

4 Results

The following section will highlight and present the result of the implementation part and how they align with the initial goals for the project.

4.1 How well does our implementation perform

There are 3 different aspects to the project implementation,

- Robot navigation (Finding the two first QR code)
- QR Code reader
- Robot guidance (Being able to direct the robot to a specific point)

The "Wander" function handles the initial part of the program, where the robot is using its laser scan to safely wander around in the operational scene. As soon as the first QR code is found, the information is extracted, processed and stored. In our implementation, we need to find a total of 2 different QR codes, before we are able to perform the transformation from hidden frame to odometry frame, which is needed to find the remaining QR codes. Once the transformation is performed, the robot can navigate directly to the next QR code. Overall, based on the tests and final video, the implementation lives up to the project requirements. The robot is able to navigate and locate randomly placed QR codes, process and extract the data and finally use the data to calculate where the location for the remaining QR code.

4.2 How have we tested and result of videos

There are a few things to notice in the video, first of all, in the beginning of the video where the "wander" section of the code is running. As the robot spots the first 2 QR codes within a short timeframe, it doesn't wander a lot, before it starts finding the remaining QR codes. the twist are short and makes the robot drive either left or right, without much hassle.
The robot is navigating around based on its laser scan, until two different QR codes are located. We can see this in the video, where the behavior of the robot changes once the two QR codes needed for the transformation have been found. Before the transformation, the robot is driving more randomly and turning round in various directions until it finds a direction where there is enough space, however, after the transformation the robot is more direct and can drive straight to the next QR code. This can also be seen in the RViz section of the screen, where a path is displayed when the transformation is complete. The robot and video stops when all the QR codes have been located and the secret message have been printed out.

The results of the project (videos) have been published to the following Youtube video.
https://youtu.be/-EEP-hHf_b4

5 Conclusion

The goal of the project was to be able to navigate and find a hidden message through QR codes randomly placed in a room. This is reached. It was allowed to do a offline mapping first, but after that randomly placed obstacles is introduced. This means that the robot does not have complete information and have to rely on logic and randomness to find the QR codes and navigate the room. The randomness is used in the first section where less then 3 QR codes are found. This can be seen done in the video from the result section, but it can also be seen that the first part of the challenges is a lot harder and more time consuming if the robot happen to wander around without finding anything. This could be improved using the offline map, so a predefined search route could be used instead.

$$Dice(A, B) = \frac{2|A \bigcap B|}{|A| + |B|} \tag{2}$$

$$IoU(A, B) = \frac{A \bigcap B}{A \bigcup B} \tag{3}$$