COMP3631: Robotic Cluedo

INTELLIGENT SYSTEMS & ROBOTICS

Ilyass Taouil [sc14it]

University of Leeds

ABSTRACT

The aim of the project is to design and implement a software solution that instructs a robot on which moves to take in each circumstance, with the final outcome of playing the famous board game **Cluedo**, with the player in case being the robot itself.

The presented result will work in a scenario where two images are pinned to the wall (with no previous knowledge about the exact locations) and where the robot will have the target to detect and recognise the images within a 30 minutes' time frame. The images have an AR-marker printed on them to allow an easier detection. Moreover, a snapshot of the detected images and their position in the map should be stored in memory for a prove of success.

Given the above problem, we can break the robot actions into three, which are quickly presented below:

1. Object Search
2. Object Detection
3. Object Recognition

1 Object Search

The object search enables the robot to perform an exploration of the environment given, with the aim of detecting the images on the wall during the former's motion. The search has to be such that all the sections of the surroundings are visited, allowing the robot to detect the images even if these are placed in difficult positions.

2 Object Detection

The image detections process has two components to it. The first one is the detection part, which is possible thanks to the usage of AR-markers. However, once the image is detected and its position in the map is retrieved a correct positioning in front of the object is required to obtain a successful recognition result.

3 Object Recognition

The final step in the chain is recognising the image and store its map position as well as its snapshot with a boundary around the image. The recognition has been implemented such that it could be as robust as possible using computer vision techniques that enable a powerful feature realisation.

2. DESIGN

In the following section a detailed description of the three components previously described is presented. For each element, whenever possible, a portrayal of the initial idea, the actual solution and its limitations is provided.

2.1 Object Search

The first considered approach for the object search consisted in sampling random points from the map and initiate a movement towards the sampled position. Once the given location has

been reached, adequate procedures are called to determine if an image is in the reachable field of vision of the robot. The same process would be repeated until both images are found.

However, after an evaluation several faulty points have been found. The method in fact is not an efficient one, as the probabilistic approach could take several minutes before reaching an appropriate position where the image is visible and several others before the next image is detected and processed.

Moreover, the random sampling would require an additional filtering layer where the points that are not reachable should be filter out as to avoid unnecessary planning by the robot.

Hence, the decision of using an obstacle avoidance approach, which in this proposed scenario tries to follow the walls using laser scan values coming from the depth sensor readings. The scan readings provide a measure of how far the robot is from nearby obstacles. The readings have been split into three ranges such as to handle obstacles to the left, in front and to the right of the robot.

A logic based on these sensor readings decides whether the robot should remain in the trajectory of movement or change its direction by either turning to the left or right to avoid crashing into walls or other obstacles.

Furthermore, bump tolerance is an integral part of the searching system in case the robot slightly bumps into the walls or any other obstacle, this helps avoid loosing localisation which is important for the object detection logic (explained in the following section).

The “follow the wall” navigation system is not entirely deterministic when compared to the initial probabilistic method, as the former’s actions are based on random angle and side (clockwise or anticlockwise) rotations for some searching cases, as for example when the robot is facing a corner.

The choice of implementing such randomness enables the robot to get unstuck from looping portions in the map as well as giving the robot a chance to discover previously unexplored portions, which otherwise would have been tedious to add to the logic itself, necessitating a higher number of splits in the laser scan data and a larger coverage in the logic.

The limitations of the search method are mainly three, its non-deterministic behaviour, the nature of the laser scan and narrow entrances. The non fully deterministic nature of the navigation might make the robot visit multiple times the same section of the room, hence increasing the amount of time needed to complete the full task. The usage of laser scans in environment where certain obstacles are made of glass can defect the functionality of the search, although there exist specific flags that can enable laser scans to work in such situations while the existence of narrow entrances within the room can as well limit the discovery of certain sections if these are too narrow (width slightly larger than the robot width).

Overall the search algorithm should perform well, being able to accomplish the task within the 30 minutes’ time frame given, exploring all the sections of the environment, given that the main limiting condition is not met (narrow entrances). A typical search movement is shown in the image below:

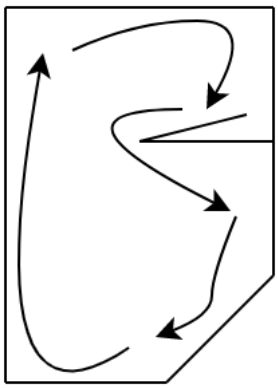


Figure 2.1: Object search movement

2.2 Object Detection

The detection process comprehends two parts; these are enlisted below:

1) Detecting the presence of an AR-marker.
2) Position the robot in front of the AR-marker.

The initial approach to detect the presence of an AR-marker consisted in calling the transform routine (**lookupTransform**) after every movement of the robot to see whether an AR-marker, and hence an image, was visible to the robot. However, the following approach does redundant checks which are unnecessary most of the times and ultimately slow down the object search.

Therefore, a different design solution has been implemented. In fact, the robot offers a ROS topic (***ar_pose_marker***) which publishes continuous data with the pose of the AR-marker if this is visible. Hence, a subscription to the service is made and a background data retrieving procedure is set such that as soon as the condition fires (AR-marker visible) the robot seamlessly stops the navigation to start the image detection process. The condition is based on the existence of the **markers** object in the list of objects published by the service.

The second part to the detection process is to actually position the agent in front of the AR-marker, and hence the image, as this would facilitate the image recognition step. To achieve this the AR position in the map is used to compute the new pose for the robot. The transform and the quaternion of the AR-marker obtained via the transform routine **lookupTransform** is used to build a rotation matrix which provides the values needed to compute the new vector pose containing the x and y coordinate that would place the robot in front of the AR.

Nonetheless, the robot is not necessarily going to have the correct rotation to face directly the AR-marker, therefore a correct theta rotation has to be computed via the same rotation matrix presented before. Further explanation and mathematical formulations are conferred in the “Implementation & Results section”.

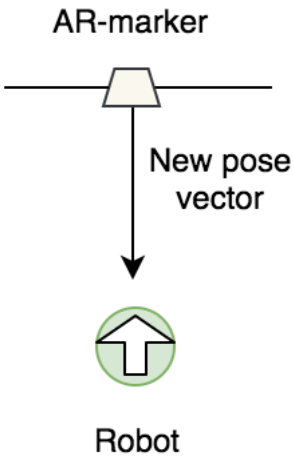


Figure 2.2: AR-marker positioning (1)

The designed object detection solution is probably the best reachable for the given constraints, although not the only solution, nonetheless the non-optimality. In fact, by knowing the position of the robot and the AR-marker and using the properties of the triangle, the pose in front of the marker could be easily found using trigonometry, however the former approach would not be as robust to the noise as the one presented before, given that slight variations in the **yaw** angle could introduce a big positioning mistake.

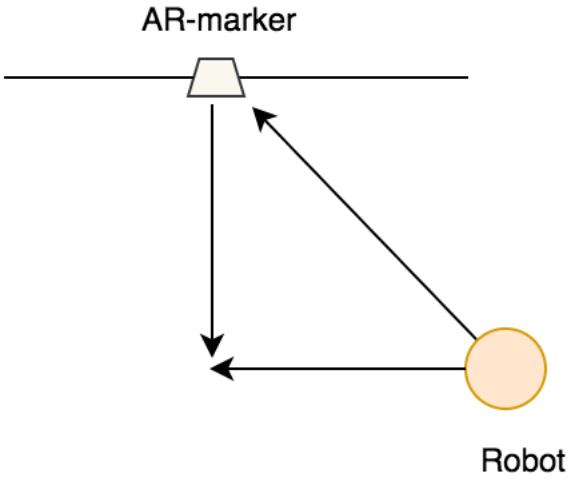


Figure 2.3: AR-marker positioning (2)

2.3 Object Recognition

The object recognition uses features to recognise the object. This is based on the idea that working with patches is easier than working with full images, due to shape, texture and lighting variances that these might have. Hence, by using parts of the image we make the whole process more robust to possible variations.

These key points or salient points, are unique points that can be found with high reliability in multiple similar images.

In the designed solution a feature descriptor such as ORB is used, which is a similar algorithm to the other main two feature descriptor SIFT and SURF but computationally less expensive and hence faster. In fact, ORB uses the FAST key point descriptor to find the salient points in the image, to then apply the Harris corner detector to find the top M points. The FLANN

matcher is then used to compare the salient key points in both the template image and the input image.

The strength of the method is its robustness to light variation when the same intensity is added as a constant to the entire matrix of the image as well as its rotation invariance and spatial invariance. Nonetheless, the method would not be able to scale to thousand if not millions of images as in this case the computational resources needed would raise up. The FLANN matcher and the ORB descriptor were already implemented in the **PlanTracker** class, which has been borrowed from the **OpenCV** library. The **Recognition** class which differently has been designed from scratch just offers the interface to the main logic to feed in the live image data and obtain a result back from the **PlaneTracker**. Moreover, template matching has been used to draw the boundary around the image whenever the image is recognised.

Machine Learning methods could have been used as well to accomplish the task, for example training a neural network with seven output neuros, with each neuron being a possible output object. However, using such an approach would certainly consume more resources than the one really needed for the task.

Hence, the final design solution combines the ORB descriptor, the FLANN matcher and template matching to accomplish the recognition and the image boundary drawing as using only template matching for both tasks produced a not so performant result. Moreover, template matching doesn't offer a robust behaviour to noise, such as slight rotations or image cluttering (which is discussed more in the results section).

3. Implementation & Results

In the following chapter we present how the different logic modules have been implemented in the software solution as well as their hierarchical structure which can be seen in figure 3.1 below. Moreover, the results for the three main section of the program (search, detection and recognition) are presented.

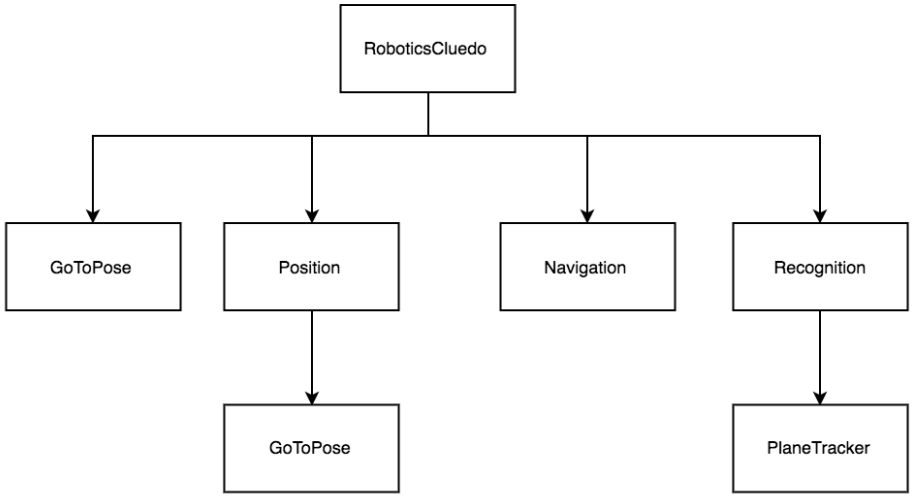


Figure 3.1: Program structure.

3.1 Planning & Search

Implementation

The planning and search behaviours are implemented in the RoboticsCluedo and Navigation modules respectively.

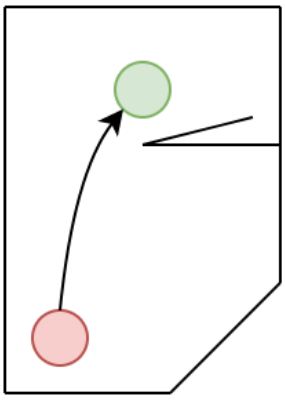
The first class (**RoboticsCluedo**) handles the overall logic including the initial positioning in the centre of the room. In fact, the x and y coordinates passed to the **run** method of the class are passed to the **GoToPose** class which via the ROS' **actionlib** package commands the **move_base** of the robot to move in the desired position.

The second class (**Navigation**) handles the object search behaviour using the in-coming laser scan data which the **RoboticsCluedo** class subscribes to and passes to the Navigation's method **navigate**. These values are then processed and a logic unit is run by the **navigate** method based on the result of the process, deciding whether to move forward, turn in either directions or to spin to avoid dead sections such as corners.

Planning results

Below are presented the results of how efficient the planning of the initial point positioning is. More precisely we evaluate how close does the robot get to the given pose starting from the initial position. The green dot in the figure represents the given pose, while the red circle represents the starting position.

The robot has always been given the same position to reach, in this case an x coordinate of (-2.53) and a y coordinate of (-0.32). The actual reached pose has been checked by running a **tf** transform of the base-link of the robot with respect to the map.

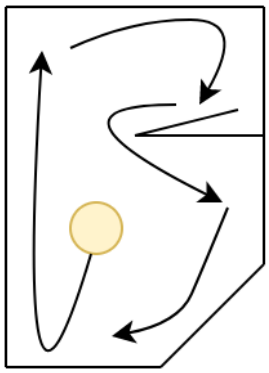


Given Pose	Reached pose	Abs. Difference
(-2.53, -0.32)	(-2.46, -0,45)	(0.07, 0.13)
(-2.53, -0.37)	(-2.46, -0,43)	(0.07, 0.06)
(-2.53, -0.37)	(-2.47, -0,44)	(0.06, 0.07)

We can notice how the motion to the initial pose where the search starts is really accurate and respects the given constraint of reaching the given point within a 0.5m difference. In fact, the maximum absolute difference met is only 10% of the given constraint which we can consider an optimal result.

Search results

After the initial positioning of the robot in the given pose, the robot will have to perform a search as previously stated. Below are presented the results of how much time does the robot require to scan the whole room, where by scan is meant a full cycle as shown in the diagram below. The singular tests have been run with the robot always starting at the same position (-1, 0) which is presented by the yellow circle in the diagram.



Laps	Time
Lap1	1:40
Lap2	1:33
Lap3	1:47
Avrg.	1.40

The agent is able to scan the whole arena in one minute and forty seconds in average. Moreover, the environment used in this test case had two sections to it, and the robot proved to be able to get out of inconvenient unopened spaces with ease thanks to the corner logic present in the navigation system.

However, some considerations have to be made. In fact, the arena itself is really small and given the non-deterministic behaviour of the search logic it happens that sometimes the robot does visit consequently the same section because of the random choice of the rotation side and angle. This approach can certainly have repercussions on the time needed for the robot to visit the whole space, especially when the testing environment is considerably large.

3.2 Object identification & Location accuracy

Implementation

The whole object identification logic is implemented in the Position class, which is used directly by the main logic (**RoboticsCluedo** or **RC**) and called accordingly. The identification process itself, where by this is meant the detection of an AR-marker and the frontal positioning is called as soon as the AR becomes visible. The following diagram illustrates a typical example:

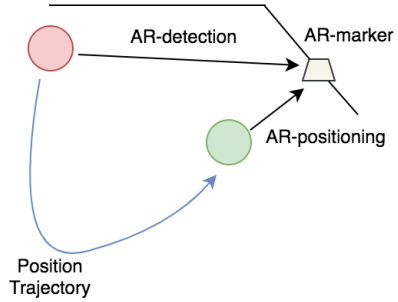


Figure 3.2: Object detection flow.

Below we present the mathematical formulation of how the pose in front of the AR-marker with the correct rotation angle are computed. These are the steps of the computation:

1. Compute the transform and quaternions representation of the AR-marker in the map frame (obtained via the **lookupTranform** using the **tf** package).

2. Build the rotation matrix from the transform and the quaternions result obtained in point (1). The following is obtained by using the **tf** package routine **fromTranslationRotation** which returns a 4x4 numpy matrix with the values needed for the pose and theta computation.

3. Identify the z axis entry in the rotation matrix, which is the third column from the left and its first two row values defining our z-axis vector.

4. Compute the new pose by adding the transform values obtained in point (1) with the z axes vector obtained in point (3) where this is multiplied by a certain distance representing how close to the AR-marker the robot should be. The equation looks as follows, where (Px) and (Py) are the new pose, (Xt) and (Yt) are the transform values and (Zx) and (Zy) the z axis vector components:

$$\begin{pmatrix} Px \\ Py \end{pmatrix} = \begin{pmatrix} Xt \\ Yt \end{pmatrix} + \begin{pmatrix} Zx \\ Zy \end{pmatrix} * distance$$

5. Finally compute the theta value of the robot to make sure the robot is facing directly the AR-marker. The angle is given by the inverse of the tangent of the quotient of the flipped (Zy) and (Zx) normalised components, and this is because the orientation required is the opposite direction of the z axis vector while the normalisation is needed because the third entry in the columns is disregarded. The equation is as follows, where $-Zy'$ is the normalised and flipped y component and $-Zx'$ is the normalised and flipped x component:

$$(\theta) = \tan^{-1}(-Zy'/-Zx')$$

After computing the pose and the theta values the position logic feeds these values to the **GoToPose** class that sends a motion command via **actionlib** to the **move_base** module which will try to reach the given position in the map.

Results

Below we present the object identification accuracy. The test has been run three times with the same objects in the same position and with the robot starting in the same location every single time. The test took in consideration the correct position of the images in the map obtained with a manual check on RVIZ and the position stored by the robot. We show below the position of the images in the testing arena (red circle is Scarlett and grey circle is Wrench) and a table comparing the different values.

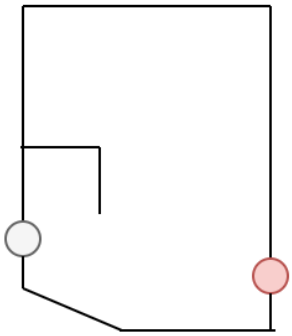


Image	Actual Position	Detected Positions		
Wrench	(0.79, -4.54)	(0.72, -4.37)	(0.55, -4.36)	(0.66, -4.4
Scarlet	(-0.22, -2.79)	(-0.26, -2.83)	(-0.29, -2.90)	(-0.37, -2.8

We can notice how overall the position detected by the robot is pretty accurate when compared to the actual position of the image, although this measurement is prone itself to errors given their estimated localisation in the map (manual check on RVIZ). We can as well associate the big differences between the actual positions and the detected ones with de-localisation problems that robot has due to imperfection in the motion (wheels slipped, fast spins, etc.) or noise coming from the sensors.

3.3 Object recognition

Implementation

The image recognition is bases on two classes, the **Recognition** class and the **PlaneTracker** class. The decision of when to call the recognition routine is made by the **RoboticsCluedo** class based on a set flags that keep track of the overall process chain (AR-detection, AR-positioning and image centring).

The **PlaneTracker** has two parts to it. The first one being the loading of the images we would like to recognise, in our case the Cluedo characters and the weapons, hence, easily extendible to other objects as well. The second part is the tracking feature defined in the track method which finds features or **saliency** points thought the computation of gradients in the image.

Once the feature points have been found in both the loaded images and the image feed coming from the robot RGB camera a feature matching is run, and the image with the best match is returned as the recognised one.

The **Recognition** class makes uses of the **PlaneTracker** to run feature matching on the image feed passed by the main class when time for recognition arrives, mainly after the AR detection and positioning.

Results

The accuracy test for the recognition (correct recognitions have a boundary presence in the image) has been carried taking into consideration three main points that might alter the former, mainly:

1. Lighting condition (dark).
2. Image clutter.

3. Image rotation.

I. For the lighting condition test two different images (wrench and plum) have been used. The test consisted in turning off the lights above the arena used for the test in the staff room and run the program. The result is considerably positive given that the recognition module has proven to be able to recognise images even in extreme lighting conditions.

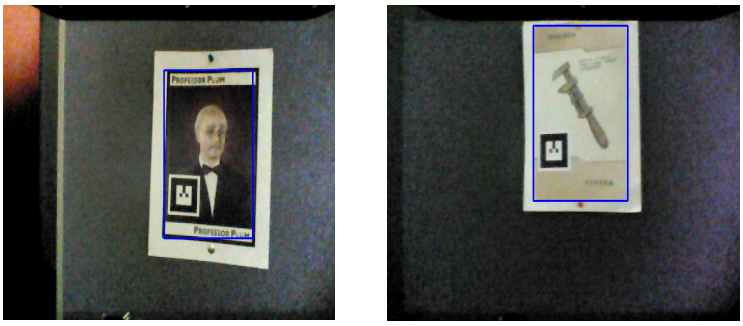


Figure 3.2: Dark detection (Plum & Wrench).

II. The clutter test consisted in pinning the image with other images present in the background, as well as obstructing the images with pieces of Lego. The result obtained is extremely positive as the recognition module was still able to detect the images with a correct boundary positioning as well.

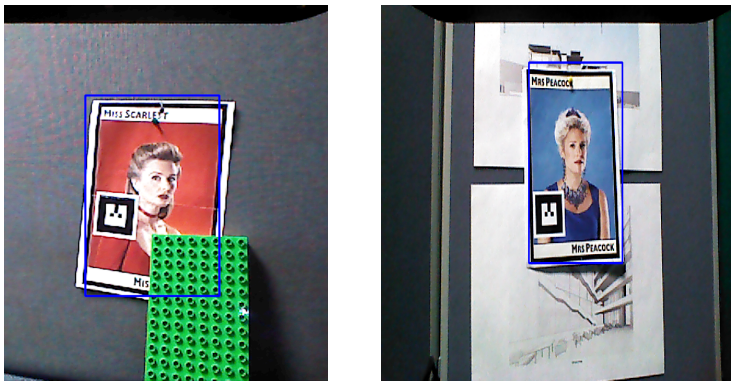


Figure 3.2: Cluttered detections (Scarlett & Peacock).

III. The last test consisted in positioning the images with a slight rotation (mustard example) and with a full 160-degree rotation (rope example). The recognition result is still extraordinary robust thanks to the rotation invariant feature of the ORB descriptor, however the template matching boundary system fails when the rotation is to high as can be seen in the left figure.

Figure 3.3: Rotation (Rope & Mustard)

4 Self & Team Assessment

Meetings & Labs

I took part in all the meetings the group had and took the initiative of organising most of them for each step in project process, starting from the initial design to the final testing of the solution. Moreover, I attended all lab sessions, both the individual and group ones being able to complete every single one of them, hence gaining full marks.

Solution Design

I designed the complete vision section including object detection and recognition by myself with minimal help, choosing which methods to use upon a trial and error approach thanks to quick prototyping. Moreover, I designed the software architecture for the final solution.

Solution Implementation

Implementation wise I wrote 100% of code for the vision module (excluding the PlaneTracker and the GoToPose) including the image detection, image recognition, image boundary and AR-marker positioning logic units. I implemented as well the main logic that merged together the previously scattered navigation and vision modules. The only piece which I haven't fully written myself is the navigation class, which nonetheless I entirely refactored to fit commons programming standards.

Testing

I carried the debugging process by adjusting the code and fixing the logical mistakes made which haven't been covered previously in the simulation environment.

Report

The following report has been fully written by myself, and no report section has been exchanged between me and the other members of the team, except for certain testing data.

Social Being

I always encouraged discussion and critics within the team and this has proven to have a great return, moreover I always complimented people for their idea independently of the success or failure of the former.

Other member's contribution

Overall the rest of the team did not contribute to the project as much as I did, nonetheless they seem to have tried to the best of their capabilities by attending the group meetings and offering help during the various phases of the project.

5 Appendix

The following are the links to videos demonstrating the functionality of the solution presented in this report:

1) Object search obstacle avoidance: https://www.youtube.com/watch?v=seu7PmEVW9Y

2) Perfect solution: https://www.youtube.com/watch?v=UxJlaxV9_yU&feature=youtu.be

3) Initial positioning: https://www.youtube.com/watch?v=RUzrUzInKP4&feature=youtu.be

4) Navigation, Detection and Recognition: https://www.youtube.com/watch?v=smiY0M0Eb4k&feature=youtu.be

5) Detection and Recognition: https://www.youtube.com/watch?v=KfGFNOKT0Tk&feature=youtu.be