Robot
Final Submission Report

Alexandre Pieroux

May 2018

1 Introduction

In the context of the course "Introduction to intelligent robotics", it was asked to program a small robot in a simulation to perform certain tasks. This report explain in details the milestones implemented with their algorithms and their results. All goals couldn't have been reached, but examples of possible implementations will be given.

2 Motion

One of the first task to perform is to find a way to tell the robot to move from a pose to another. Depending of the physical constraints, the robot can have certain degrees of freedom. The provided robot is equipped with omnidirectional mecanum wheels that make it holonomic.

2.1 Kinematic

We define the following variables:

- x_R is the robot velocity according to the x axis of the robot frame.
- y_R is the robot velocity according to the y axis of the robot frame.
- θ is the angular velocity according to the robot frame.
- φ_i for $i \in \{1, 2, 3, 4\}$, is the angular velocity of the ith wheel of the platform.
- ϵ is the measured angular velocity error.

As the kinematic is based on the physical properties of the robot, we need to take into account the length of the robot, its width and the radius of its wheels.

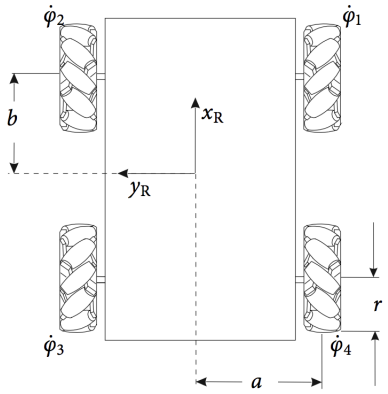

Figure 1: "Omnidirectional platform with Mecanum wheels (top view)" [1]

Following [1], the augmented foward kinematic[1] can be expressed as:

$$x_a = J_a^{-1}\varphi$$

where $x_a = \begin{pmatrix} x_R \\ y_R \\ \theta \\ \epsilon \end{pmatrix}$, $\varphi = \begin{pmatrix} \varphi_1 \\ \varphi_2 \\ \varphi_3 \\ \varphi_4 \end{pmatrix}$, $J_a^{-1} = \frac{r}{4}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ \frac{1}{a+b} & \frac{-1}{a+b} & \frac{-1}{a+b} & \frac{1}{a+b} \\ \frac{4}{r} & \frac{4}{r} & \frac{-4}{r} & \frac{-4}{r} \end{pmatrix}$

And thus the inverse augmented kinematic can be derived in:

$$\varphi = J_a x_a$$

where $J_a = \frac{1}{r}\begin{pmatrix} 1 & 1 & (a+b) & 1 \\ 1 & -1 & -(a+b) & 1 \\ 1 & 1 & -(a+b) & -1 \\ 1 & -1 & (a+b) & -1 \end{pmatrix}$

It is then easy to compute the individual wheel velocities knowing the angular velocity, the x and y axes wise velocities. These equations are implemented in the source file "Robot.m" in the source folder of the project.

2.2 Pose Interpolation

The pose interpolation implemented for the given solution, aim to provide the smoothest way to travel from pose A to pose B.
Depending on the task to perform, an algorithm provide a path to follow. This

[1]The following equation take into account the velocities errors that occur when performing the motion. This allow to compensate and regulate the motion of the platform.

path is then interpolated respect to the physical constraints of the mobile platform to perform it. The pose interpolation is divided in two parts: the translation interpolation and the rotation interpolation. Then the two interpolations are used to manoeuvre the robot.

2.2.1 Translation Interpolation

The translation interpolation use the quintic polynomial method that allow to respect the six boundaries on the initial and final positions, velocity and acceleration. The function *mstraj* is used by specifying the time per segments of the path, the reason will be given in the rotation interpolation subsection. The parameters used for the acceleration and sample interval can be consulted in the file "Brain.m" and the maximum velocity of the robot is set in the *startSimulation.m*. But these constants are subject to changes depending on the situations. For instance, if a payload have to be considered on the platform, the values will be adapted to keep it safe onboard.

2.2.2 Rotation Interpolation

The rotation interpolation is implemented thanks to the spherical linear interpolation (slerp) using the unit quaternion class. As we privilege the completion of the rotation of the robot between two way points, the speed of the translation need to be adapted in certain situations. In order to make it possible, the time needed to finish the rotation on a segment is computed. If the time for the rotation is greater than the one needed for the translation, then the time of the segment is the one of the rotation and thus slow down the translation on the segment. Otherwise, the rotation is done before reaching the point and the time of the translation is used on the segment.

2.2.3 Path Refinement

Path given by algorithms can vary. For instance the Dstar algorithm give a "cell path" and PRM return "way points" that correspond to a node in the random tree. In order to have a smooth path, a refinement is done to keep only the way points of the path, according to a small angle tolerance of 0.1 rad. This is done through the *reducem* function that use the Douglas-Peucker line simplification algorithm. Once the path is refined, it is interpolated and performed.

2.2.4 Driving

In order to drive the robot to its pose, the solution loop while the destination is not reached. The pose to reach is retrieved with the *interp1* function that retrieve the rotation and the point in space to reach at time t. This interpolation

according to the time is done slightly in the future to avoid stopping along the path.
As the loop is not regular and some iterations can be longer than others, its is allowed to the robot to speed up to 20% faster than its max velocity to catch up. The maximum velocity of the robot at the initialization have to be set in consequence.

3 Map

The map is an important component of the navigation for the solution provided. It impact a lot of the aspects of the robot navigation such as:

- Computation time for path.
- Updates times.
- Precision of the displacements.
- Location of obstacles.
- ...

The map is implemented with sparse matrix. The original guess was to maintain a map that would contain values between 0 and $+\infty$. The process of updating the map consist of reading the data from the sensors and to increment the value of the cell where an obstacle has been detected. According to this, a simple threshold is applied when path planning a direction to detect obstacles, but it lead to errors and imprecision. To avoid these, the update mechanism was updated to diminish the value of the cells where nothing were detected by the sensors[2]. This lead to a map that constantly update the values of the cells on which the sensors swipe.
In order to save space, the map only grow column and row wise according to the data input in the map.
At the initialization phase of the map, the robot is inserted in the map by setting the cells on which it stand to $-\infty$.

4 Exploration

The exploration phase is performed in order to produce a relatively precise map of the environment of the mobile platform.

[2]The update process "trust" more an obstacle than the "void" detection. This mean that when a sensor detect an obstacle, the value of the cell is incremented by two but when nothing is detect it only decrease it by one. This was implemented after noticing that sometimes, due to rounding on scaled data, some walls tended to disappear.

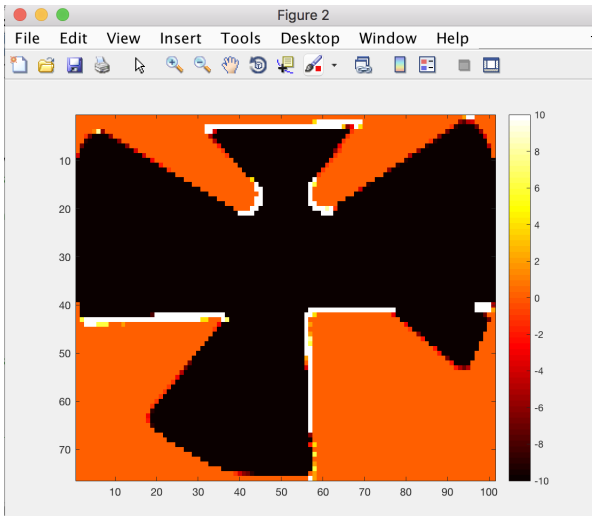

Figure 2: Map under construction after a barrel roll of the robot.

4.1 Algorithm

4.1.1 Map Processing

The algorithm chosen to perform this task is *Dstar*. In order to product a path to be used by the algorithm, a threshold for the obstacle is applied on the data of the current map.
The produced map is then inflated according to the proportions of the robot in order to avoid it to bump into the obstacles. The inflate is performed by dilatation of the binary image of the map using a disk shape.
The robot position is inflated and carved into the map to ensure that it is never in an obstacle before planning the path[3] When this process is done, the map is ready to be processed by Dstar.

4.1.2 Dstar

The Dstar algorithm is chosen for its ability to produce a distance map that can be computed by setting the goal as to be the robot pose[4]. Once the distance map is retrieve, the map threshold is then applied to only select the point in an

[3]This was observed several times while trying to explore the house, even by using the *inflate* option of the Dstar function, due to rounding caused by the scaling of the map, the robot find its pose inside an obstacle.

[4]A small tweak was performed in the code of the library in order to retrieve the distance map.

"undetermined" state by the thresholds[5].
The closest unknown point is then chosen to be the goal.
The point is then queried to produce a path. The path returned is reduced by 30% to avoid the robot to bump into obstacles that would be unknown on the map. An efficient way to navigate would be to constantly query an unknown point in parallel of the driving and setting the new destination once a new point to explore is found[6].
This process is repeated until no "undetermined" point is found in the distance map.

4.2 Result

The first phase of the exploration is a barrel roll performed to detect nearby obstacles and to produce a first map. The result are quite precise and avoid the robot to hit obstacles, but the computation time required by Dstar increase as the map increase or if the resolution is augmented. However, complete exploration can be achieved and the following map is produced:

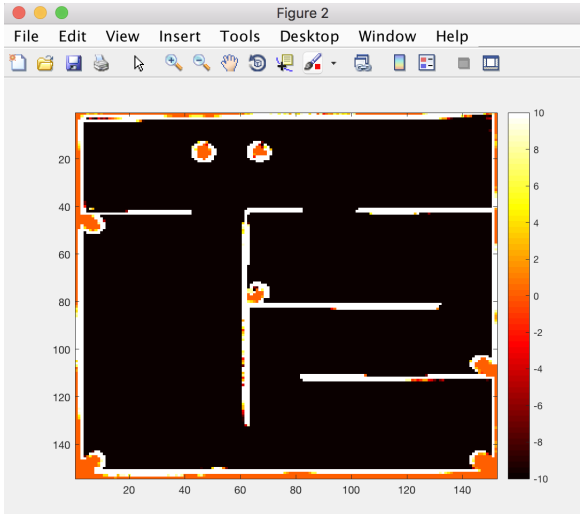

Figure 3: Map build after exploration.

[5]The points that are between the two thresholds that determine a "void" cell and an obstacle.

[6]However, as multi-threading is hard to achieve within Matlab, this was not implemented and the new destination is computed every time the robot reach its previous goal.

5 Bins Detection

The bins detection occur after a map has been built. This phase consist of determining points in the map where the centers of the bins might be located. As the bin appear as circles on the map, a circular Hough transform is used to detect circles in the map.

5.1 Algorithm

As the produced map contain a lot of imprecisions, a morphological closing is performed to enhance details. Then the function *imfindcircles*, that use circular Hough transform, is used to detect potential bins[7].
Then each point is set as a goal for the robot to check.
To navigate in the explored environment PRM[8] (Probabilistic Road Map) is used for its simplicity, efficiency and ability to build a connected graph in the map. The main advantage of PRM is that there it support re-planning and dynamically setting goal without rebuilding anything[9].
Another advantage is that it is easy to get the nth closest point of the bin candidate point in the tree. This allow to find a goal that is not in an obstacle and at least at a given distance from the point to check in order to take a good picture.
Once the path is planned, the robot travel to the point to check and correct its orientation to face it in order to take a picture.

5.2 Results

This allow the robot to move close to the point and to take picture of the bin candidate. One of the drawback of PRM is that some of the paths are far from efficient and lead to unnecessary long path to get to a point. But each executions lead to different results but always succeed in getting to identify the bins, with few false positives, and to get to them at a desired distance.

6 Robot Vision

In order to check that a circle detected on the map is a bin and match it with one of the provided picture, several tempt were made but no convincing solution

[7] And also the tables.

[8] RRT (rapidly-exploring random tree) could have been more interesting but the library required a "vehicle" class as argument, to take into account the physical limitation of the vehicle. But some difficulties into implementing the mecanum wheel kinematic into a "vehicle" class were hard to do. This idea was abandoned.

[9] Dstar would have been more precise and smooth but each time the distance map have to be recomputed for a given goal.

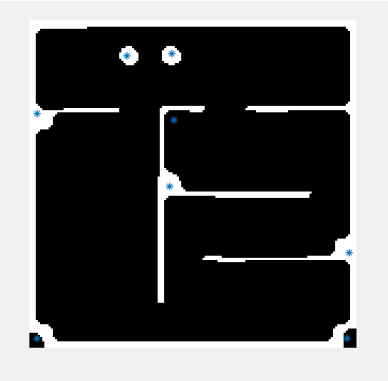

Figure 4: Morphologicaly closed map with detected bins using Hough transform.

was adopted. This section describe what has been tested and what solution are available to complete this task. But at the moment this report is written, this was not achieved yet.

6.1 Features Matching

One of the guess that was made was to use SURF (Speeded Up Robust Features) feature for matching. An estimation of the geometric transform is then performed using the *estimateGeometricTransform* that exclude outliers in the matched features by using the M-estimator SAmple Consensus (MSAC) algorithm. The MSAC algorithm is a variant of the Random Sample Consensus (RANSAC) algorithm. Then the transform is applied to the taken photo in order to recover the original picture and compare it to the original in order to get a matching score.

6.2 Results

The drawback is that, even if SURF features are robust, if the robot take a photo with a too difficult angle[10], no strong matching can be done and this end up in no bins matched during that phase.

6.3 Improvements

Several improvement are going to be explored:

[10]This is very likely to occur as the bin detection is not perfectly accurate and the closest point might be behind a wall or on a close angle to the wall.

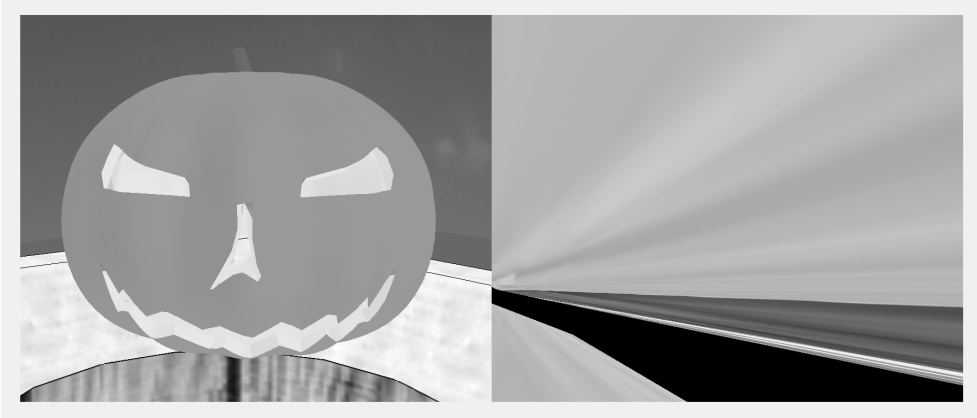

Figure 5: Left: pumpkin original picture. Right: recovered image after geometric transform estimation of a picture of the pumpkin take by the robot.

Figure 6: Pumpkin picture taken by the robot.

- An improvement would be to pick the n closest points to the potential bins and take pictures to try to get a strong match.
- Another possible solution would be to cycle between the points where no strong match were found until each bins is found. The idea is to "come back later for another check from another point of view".

Another way to perform the match would be to use "bag of features" but as it require a data set to be robust this idea was not retained as the goal that is seek in this solution is to match the bins based only on the sole picture provided. Although the classification would be stronger but would require to produce several picture from the simulation under different angles and illumination.
One of the last solution would be to use ICP (Iterative Closest Point) to find a robust transformation if possible.

7 Other Milestones

Other tasks such as object detection on the table and object dispatching were not treated in this submission but an active work continue to reach these goals until the final presentation.

References

[1] Röhrig, C., Heß, D., Künemund, F. (2017, August). *Motion controller design for a mecanum wheeled mobile manipulator*. In Control Technology and Applications (CCTA), 2017 IEEE Conference on (pp. 444-449). IEEE.