Introduction to intelligent robotics

Project report

Nicolay Pierre, Germay Antoine

Academic year 2016–2017

Contents

1 Intro

In this project we were asked to program a robot to make it behave intelligently. Some milestone were listed to target our work. Form those we implemented the milestone A1, B4 (with only one table), C1 ,C2 ,D1 and D2. We also tried to separate the work and Pierre worked more on the vision part and Antoine on the control part.

2 Holonomic controller + A1 changes

2.1 Description of the algorithm used

To use at most the capabilities of the youbot a controller able to give a longitudinal and transversal speed is needed. A convenient solution to control a robot is to use a pure pursuit method. This method computes the speeds from the robot's position and a look-ahead point on a path to follow.

In the framework of this project, this algorithm is implemented in a class `HolonomicPursuit`. This class really differs from the approach that was taken for the first milestone. This class also implements several methods to control the youbot position around tables or baskets.

2.2 `step` method of `HolonomicPursuit`

This is the most important method of the class as many other methods of the class rely on it.

`step` takes as arguments `obj`, a reference to an instance of the class and `position`, a vector containing the x y positions of the youbot and its angular position, all in the map reference frame. The output of the method are the longitudinal, transversal and rotation speed of the youbot.

Ideally the path to follow should be continuous. However, the path to follow takes form of collection of points in the map reference frame that are contained in `obj.pathPoints` (set with `setPathPoints`). Thus, linear interpolations are performed if needed.

2.2.1 Longitudinal and transversal speeds

Firstly, the method search for the closest point of `obj.pathPoints` to the youbot. This point is the starting point from which the look-ahead point is obtained. The distance on the path, between the closest point and the look-ahead point is the look-ahead distance. This distance should not be too small otherwise oscillations may occur. On the other hand, if this distance is too large the real trajectory of the robot cuts the path to follow and the robot may hit walls.

Then the algorithm adds the length of the different segments of the path starting from the closest point. When this sum is greater than the look-ahead distance, the algorithm knows that the look-ahead point is somewhere on the last segment whose length has been computed. In most cases the look-ahead point does not correspond to a point of `obj.pathPoints` but is an interpolation of two of its points. The problem to find the interpolation parameter is explained here after.

The segment containing the look-ahead point is delimited by A and B that both belongs to the path. The equation of that segment in the map reference frame is:

$$y = m(x - x_A) + y_A \tag{1}$$

Where $m = \dfrac{y_B - y_A}{x_B - x_A}$ is the slope of the segment[1]. The look-ahead point is the intersection of that line with a circle centred in A of radius d, d being the remaining distance to cover so that the sum of lengths discussed before is equal to the look-ahead distance. The equation of the circle is:

$$(x - x_A)^2 + (y - y_A)^2 = d^2 \tag{2}$$

Substituting equation 1 in equation 2, equation 2 becomes:

$$(x - x_A)^2 + m^2(x - x_A)^2 = d^2 \tag{3}$$

Which is a second order equation that has two solutions[2], one such that the interpolated point is in between the two points used for the interpolation and one such that the interpolated point is not in between the two points used for the interpolation. The first solution corresponds to the look-ahead point.

If the look-ahead point is beyond the end of the path or if the closest point is the end of the path, then the look-ahead point is simply the end of the path.

Once the look-ahead point is located, the direction of the line connecting the youbot's position and the look-ahead point is known. This direction is the direction in which the youbot should go to follow the path. Thus, the resulting speed vector has this direction.

Finally, the resulting speed is projected on the longitudinal and transversal directions to get the longitudinal and transversal speeds respectively. In addition to the projection, an instance variable (`obj.acceleration`) is used to create an acceleration factor on the speeds (`obj.acceleration/obj.maxSpeedReached`). This variable is incremented a each call of the method up until it has reached `obj.maxSpeedReached`. Then the acceleration factor is no more taken into account.

[1]When the slope is infinite the problem becomes trivial (one dimension).

[2]y of the look-ahead point is obtained by injecting the solution of equation 3 in equation 1.

2.2.2 Rotation speed

The algorithm described in section 2.2.1 is sufficient to move the youbot everywhere in the room. However, because the hokuyo sensor angular range is limited, the robot should also turn around its vertical axis.

The rotation speed is controlled to align the longitudinal axis of the youbot with the path. To achieve that, the rotation speed is simply set to the angular difference between the longitudinal axis direction of the youbot and the path direction.

To let time for the youbot to align itself with the path, the controller anticipate by using the path direction of four path segments[3] in front of the closest point. If there is not enough path segments in front of the closest point, then the the direction of the last segment is used.

2.3 `stop` method

Every time this method is called the youbot decelerates a little bit up until it is stopped. In that case the method outputs a true boolean value (`isStopped`).

Before doing anything, this method gets all the speeds by calling `step` then it decreases those values. The mechanism behind the deceleration is the same as the acceleration mechanism in the `step` method but reversed.

2.4 `destuck` method

The idea behind this method is the following: because the direction in which the youbot goes made it stuck, the opposite direction make it de-stuck.

Thus, this methods outputs the speeds of the `stop` method but with the opposite sign up until the youbot is stopped. In that case the method outputs a true boolean value (`isStopped`).

2.5 `rotate` method

The rotation speed is simply set to the angular difference between the longitudinal axis direction of the youbot and the desired orientation set with `setDesiredOrientation`. Translation speeds are set to zero. When the youbot is stopped the method outputs a true boolean value (`isStopped`).

2.6 `stayNearTable` method

This method make the youbot hold an angular position and a radial position relative to a table centre (those coordinates are set in the instance with the `setRobotPositionNearTable` method and the table position is set with `setTablePosition`). The orientation of the youbot

[3]Path segments are used here as unit of length because at this state of the project the resolution of the map is considered as fixed. A more robust way to set this parameter would have been to look for the segment x meters in front of the closest point on the path.

in that position is such that its longitudinal axis is tangent to the table circumference and such that its always the same side of the youbot that is the closest of the table.

The `step` method is tricked by setting the path to a single point which is the position to hold. Thus, both translation speeds are recovered and multiplied by the distance in between the current youbot's position and the point to hold. They are also multiplied by a coefficient which is zero when the rotation speed is greater than a maximum rotation speed and is one when the robot is not rotating. The rotation speed is simply set to the angular difference between the longitudinal axis direction of the youbot and the desired orientation discussed above.

2.7 `changeAngleNearTable` method

This method is used to change the angular position around a table without hitting the table.

The `step` method is used whith a set point that circumvents the table. This set of point is generated with `setNewAngleNearTable`. Speeds are obtained by calling the `stayNearTable` method. Finally, the position of the robot near the table is updated to the new angular position.

2.8 `setNewRadiusNearTable` method

This method is used during object manipulations to change the robot's radial position around a table without changing the angular position.

3 Robotic arm controller

3.1 Description of the strategy

The strategy is to not use the V-REP IK. To be able to do that, a few rules are followed during the objects manipulation.

First, the youbot is placed such that the axis of its first joint (a vertical axis) is contained in a vertical plane containing a point to grab on an object and the center of the table on which the object to grab lies. Thus, the control of the arm is a two dimensional problem and not a three dimensional problem anymore.

Then, the holonomic capabilities of the youbot are used to reach objects. Hence, there is no need to control every joints to produce a translation of the gripper.

The last rule is not to use to much space near the objects. Indeed moving the arm near the objects could make some cylinders roll and fall on the ground.

The code to control the robotic arm is implemented in a class `ArmController`.

The next section discuss the kinematic problem that is solved to move the arm in position to grab an object.

3.2 `positioning` method

This method finds the joints position to pick-up an object by a specified point to grab (set with `setPointToGrab` method).

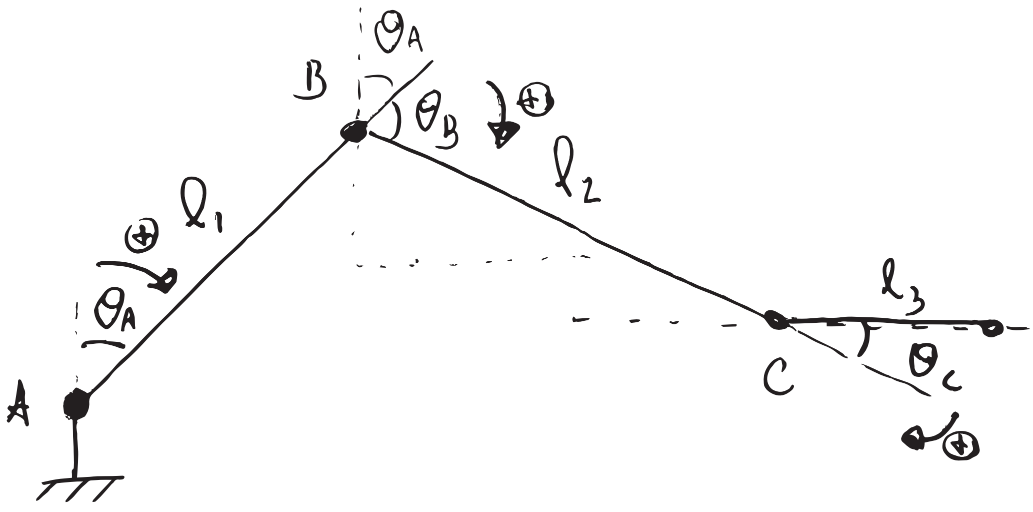

Figure 1: Robotic arm with its joints angle

As discussed before, the arm positioning can be reduced to a two dimensional problem. Fig.1 shows a scheme of the joints and bodies of the robotic arm. The coordinates of the point C are given by:

$$x_C = l_1 sin\theta_A + l_2 sin(\theta_A + \theta_B) \tag{4}$$

$$y_C = l_1 cos\theta_A + l_2 cos(\pi - (\theta_A + \theta_B)) \tag{5}$$

Equation 4 and equation 5 form a system of two nonlinear equations for θ_A and θ_B. With the strategy that is used y_C is set to the height of the point to grab of the object and x_C is equal to $0.75(l_1 + l_2)$ so that the arm is not fully extended. This problem has two solutions, one with the first kink of the arm upwards (as seen of Fig.1) and one with the kink downwards.

To solve that system, the solver `vpasolve` of the Matlab's symbolic toolbox is used. This solver has the particularity to search for a solution in a specified range of θ_A and θ_B values. The intervals used are $(-1.57; 1.30)$ for θ_A and $(0; 2.28)$ for θ_B. Those intervals respect the joint ranges and make sure that the solution of the problem is the one with the kink upwards.

To have the last part of the arm horizontal, the condition on θ_C is:

$$\theta_C = 0.5\pi - (\theta_A + \theta_B) \tag{6}$$

3.3 `lift` method

This method tricks `positioning` by changing y_C to lift object. Joints positions are obtained by calling `positioning`.

3.4 `objectPickup` method

This method is used to get the successive joints position through an entire object pick up sequence.

The sequence consists in:

- First, the youbot gets in position not to close of the table.
- An initial rotation of $-pi/2$ of the first joint. The arm is perpendicular to the longitudinal axis of the youbot.
- A positioning of the arm to place the gripper in the correct position to pick up the object.
- A translation of the youbot to reach the object.
- Once the youbot is stopped, the gripper is closed and the object is lifted above the table.
- Then, the youbot backs off a little bit before moving the arm in its initial position.
- Finally, the arm is lowered a little bit on the youbot so that the object can not slip out of the gripper and a boolean value `isDone` is set to true.

The transition in between the different states of the sequence are made when conditions on timers and on the different velocities are verified.

3.5 `objectDrop` method

This method is used to get the successive joints position through an entire object drop sequence.

The sequence consists in:

- First, the youbot gets in position not to close of the basket to not interfere with nearby objects.
- Then, the arm is lifted in its initial position, so that subsequent arm movement does not hit the youbot with the object which could make the object fall.
- The arm moves up and rotate to have its position to drop the object. Then, the youbot translates near the table to have its arm above the basket (specified by `setPlaceToDrop`).
- Once the youbot is stopped, the gripper is opened and the object is dropped in the basket.
- Then the youbot is backed off, the arm goes in its initial state and a boolean value `isDone` is set to true to signal the code calling the method that the routine is over.

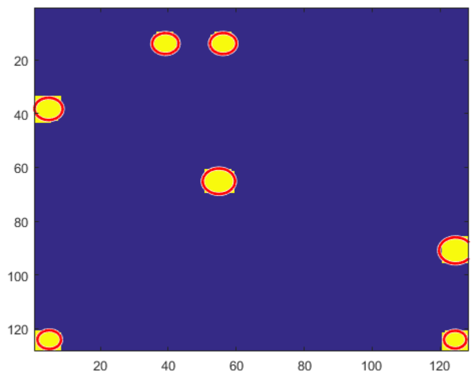

Figure 2: tables and basket detection

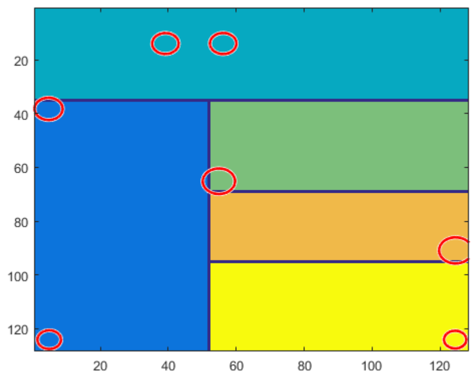

Figure 3: watershed result

The transition in between the different states of the sequence are made when conditions on timers and on the different velocities are verified.

4 Milestone C1 and C2

4.1 C1

In order to find the baskets and the tables we used some morphological operations. First we try to eliminate everything that isn't a circle with an closing followed by an opening with a round structure element of a radius 3px (resolution of the map is of 128px). One improvement we need to do is to change this value with respect to the resolution and the size of the tables. After that we use a hough transform with circles element to find the tables, their approximated center and their radius (*imfindcircles* fig 4.1). The important thing we needed was to find some interesting points on the map were we could move the robot and which were in the same room as the table. We closed the rooms doors with an opening followed by closing with one time a horizontal rectangle and one time a vertical rectangle. After we used a watershed (fig 4.1) on this map to find all points in the same room as a given basket or table. We now have found the tables and the baskets.

4.2 C2

The next part we had to recognized the landmarks given by a set of images. To do so, we used the points in the same room (discussed in the previous paragraph) as a given basket to go next to it. Then we captured an image of the landmark. We used a bag of feature and a S.V.M algorithm to classify the images. The first tries were really bad. However thinking about it with Antoine Wehenkel we made as simple observation that a single landmark can only belong to one image. Thus as we got the score for each image classification, we looked for a global optimum by computing the global score of each permutation of image. This lead to a perfect classification an also a quite robust also (got 1 error on all the tests).

4.3 D1

To do this part we had to detect the objects, their shape and their colors. To do so we first needed a good idea of the table location. Thus using the depth camera we turned around the table. Using point of cloud fitting algorithm from matlab we were able to detect the table and its center. Next we are bulding a 3d map of the objects located on the table. Once we have a good map we project every z point on the XY plan. Here we made 2 strong assumptions. First we are only trying to locate the objects on the table on wich they are unpacked. Secondly we consider 5 objects per table. Thus we use a k-means algorithm to find the center of each object. Next with this center we defined regions of interest of the objects. Those are defined as follow : the center of one [cluster + 50mm, cluster - 50mm, cluster + 50mm, cluster - 50mm, height of smallest object] . Thus we only consider object of the same size. This makes the differentiation between cylinders and rectangles easier. The differentiation works as follow : for each ROI, we try to detect a plane with a RUNSAC algorithm. Looking at the number of points in this plane we can decide with a threshold if it's a box or not. As every object is either a box or a cylinder, we classified correctly each one of them. To find the color of each object we just take a picture of the object and with some threshold we find its color.

4.4 D2

This step was quite straightforward with B4,C1 and C2. We just had to match the location to drop an object in the right basket

5 Remark

In its final state, the code is not calibrated in a perfect way. In addition, the localisation of the basket's centers revealed itself not precise enough in its current state. On some simulations, objects are dropped slightly outside of the basket or the youbot collide with them. A method similar to the one that has been applied on the left table can be applied to the baskets to locate their center more accurately.

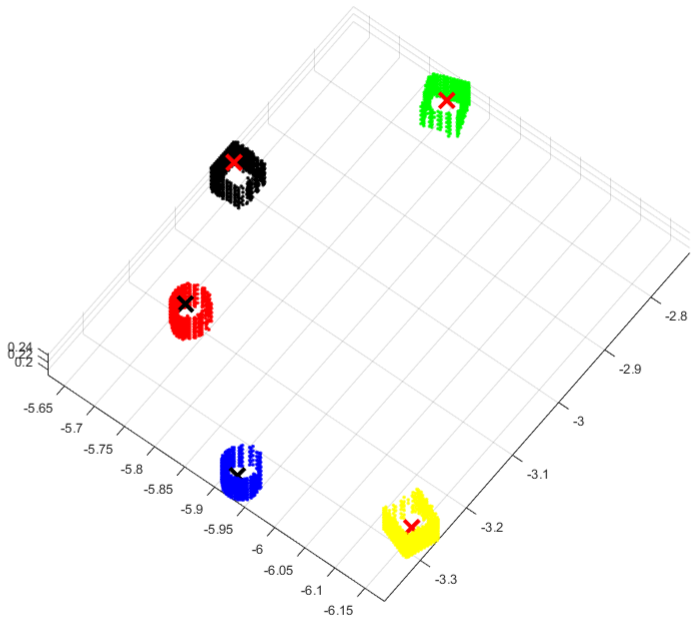

Figure 4: clustering

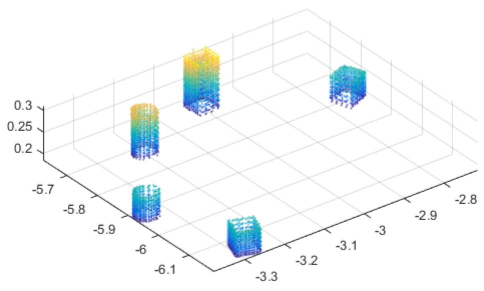

Figure 5: point cloud