Artificial Intelligence For Robotics 2
Assignment 2

Iacopo Pietrasanta, Alice Nardelli, Giovanni Di Marco

June 2021

1 Abstract

This document is a report on our solution of the second assignment of Artificial Intelligence 2 course. The initial files and planner are kindly provided by Professor Antony Thomas. The report is divided in two sections : one regarding the PDDL files and one about the implementation in cpp.

2 PDDL

Initially the PDDL domain implemented only the durative action **goto_region** which managed both the movement of the robot and the cost calculation. The PDDL domain file has been modified adding a new durative action called **localize** in order to compute the cost of the robot moving from a region to another one. According to this the **goto_region** has been also modified to now manage only the movement of the robot.

The function **(act-cost)**, which represent the cost of a movement, is updated through **(dummy)** function which iscomputed in CPP scripts and linked to the PDDL. Consequently the planner can find a plan the minimize the total cost for the task.

In order for everything to work also two fluents were added:

- **(moved ?from ?to - region)** is put to true as **end** effect of the **goto_region** durative action, moreover it is a precondition for the action **localize**. The main purpose of this predicate is to trigger the **localize** action ONLY once the movement has ended.
- **(localized)** that is a precondition for the action **goto_region** is immediately put to false once the **goto_region** action starts as a start effect. It is then put again to true as an end effect of the action localize. The main purpose of this predicate is to avoid parallelism in action execution and to enable robot movement ONLY once the robot has localized himself.

The Problem file has been modified adding in the **init** part the predicate **(localized)** , meaning that the robot starts from a known position and it is able to start moving.

3 CPP

The class **VisitSolver** has been modified adding two variables of type double to store the distances calculated at each step. These are **distance** and **unc**: they are respectively related to the distance travelled and the cost related to uncertainty of localization. The sum of these two will be the cost for a certain movement between two regions. Notice that the trace, once calculated, is multiplied by 20 meaning that we give additional weight to the uncertainty in the determination of the total cost.

- **void localize(string from, string to)**

  It is the function that computes the euclidean distance between the two regions the robot has travelled between. It stores the value in the global variable **distance**

- **void localize2(string to)**

  It is the function that computes the trace of the uncertainty matrix implementing a Kalman Filter algorithm. It stores the value in the global variable **unc**.

  - The KF algorithm works like this: each time a region is reached, the robot localize itself. All the four landmarks are taken as measurements: this generates 8 measurements equations. C matrix is an 8x3 obtained deriving measurements equations with respect to robot state (x,y,theta).
  - Pinit is a 3x3 diagonal matrix initialized accordingly to the values provided. It represents the initial uncertainty, which is uniform over the 3 states variables.
  - Matrix Q_gamma, representing the measurements noises, is an 8x8 diagonal matrix with all values set to 0.2.
  - Matrix K, obtained by the expression :

    K= Pinit * C' * inv(C * Pinit * C' + Q_gamma).

    It is a 3x8 matrix.
  - Finally the uncertainty matrix is updated accordingly to this expression :

    Pinit = (I - K * C) * Pinit.

    The uncertainty returned is then the sum of diagonal elements of this updated Pinit multiplied by 20.

- **double getDeterminant(const std::vector<std::vector<double>> vect)**

  Is a function that takes as argument a matrix, computes and returns its determinant.

- **std::vector<std::vector<double>> getTranspose(const std::vector<std::vector<double>> matrix1)**

  Is a funtion that takes as argument a matrix, computes and returns its transpose.

- **std::vector<std::vector<double>> getCofactor(const std::vector<std::vector<double>> vect)**

  Is a funtion that takes as argument a matrix, computes and returns its respective matrix of cofactors.

- **std::vector<std::vector<double>> getInverse(const std::vector<std::vector<double>> vect)**

  Is a funtion that takes as argument a quadratic matrix, computes and returns its inverse. This function obiously calls getDeterminant,getCofactor,getTranspose.

- **void printMatrix(const std::vector<std::vector<double>> vect)**

  Is a simple funtion that prints a matrix.

- **std::vector<std::vector<double>> Multipl(const std::vector<std::vector<double>> first , const std::vector<std::vector<double>> second)**

  Is the function that computes the multiplication of two matrix, of course the dimensions of the matrices needs to be adequate to perform matrix multiplication.

- **std::vector<std::vector<double>> Sub(const std::vector<std::vector<double>> first , const std::vector<std::vector<double>> second;**

  Is the function that computes the multiplication of two matrices of the same dimensions.

- **std::vector<std::vector<double>> Add(const std::vector<std::vector<double>> first , const std::vector<std::vector<double>> second)**

  Is the function that computes the sum of two matrices of same dimensions.