AI FOR ROBOTICS II

ASSIGNMENT 2: AI PLANNING

June 28, 2021

GROUP B :
Marco Staiano, Jacopo Ciro Soncini,
Lorenzo Morocutti, Riccardo Lastrico.

GitHub Repository for the assignment.

Contents

Chapter 1

Introduction

1.1 The problem in a nutshell

In the second assignment of the Artificial Intelligence for Robotics II course, it was asked to the students to deal with a mobile robots in a constrained environment. In particular, it has been requested the insertion of an action, called *localize*, in a PDDL domain file, provided by Anthony Thomas, that could compute the Euclidean distance between two regions of the environment. This action, that students have defined, also computes the act cost, that is the total distance traversed by the robot. This step allowed us to develop a plan where we could minimize the motion of the robot such that it could reached the goal (that is to visit all the locations of the environment) in the shortest way.
Thanks to the aforementioned Euclidean distance computation, it was also possible to compute an Extended Kalman Filter capable of estimating the robot state on the environment and the covariance matrix associated with it.

1.2 The Planning Engine

It was given to the students the *popf-tif* as planning engine: chosen thanks to its ability to deal with numerical fluents, temporal planning and semantic attachments, this provide us the task planning part. For the motion planning and cost computation, we use and external solver, defined by the library *libVisits.so*

COMPILATION STEPS

Pay attetion, you need to change the paths in some files, like *run_planner.bash* and *VisitSolver.cpp*, to match your directories.

- **POPF-TIF**
  Install dependencies to compile popf-tif

```
sudo apt-get install coinor-libcbc-dev coinor-libclp-dev
   coinor-libcoinutils-dev coinor-libosi-dev coinor-libcgl-
   dev
```

Read the readme inside ‘./popf-tif‘ to compile it

- **VISITSOLVER**
  Install dependencies to compile visitsolver

```
sudo apt-get install liblapack-dev
sudo apt-get install libblas-dev
sudo apt-get install libboost-dev
sudo apt-get install libarmadillo-dev  # this is the
   necessary one, the other 3 might be already installed
```

To compile the module

```
run ‘./visits_module/buildInstruction.txt‘
```

- **RUN THE SOLVER**
  Open the bash script ./run_planner.bash with a text editor, change the necessary path and name, then run it

Chapter 2

Domain and Problem Files

Going into the details of the code, some changes have been introduced in the domain file and in the problem files.
In the domain file, a durative action, called *localize_simulation*, has been added that receives as parameters the types *robot* and *region* (associated, respectively, to the robot and the two regions between which the distance has been calculated). The condition of the action is, trivially, that the robot is in the location from which it wants to calculate the distance, while the effects are to keep track of the total distance travelled and to activate a trigger when it wants to communicate to an external module which are the two regions of start and end.

```
(:durative-action localize_simulation
    :parameters (?v - robot ?from ?to - region)
    :duration (= ?duration 1) ;simulation duration is fixed as 1
    :condition (and (at start (robot_in ?v ?from))
                    (at start (= (computed ?from ?to) 0)))
    :effect (and (at start (increase (triggered ?from ?to) 1))
                 (at end (assign (triggered ?from ?to) 0))
                 (at end (increase (distance ?from ?to) (actcost)))
                 (at end (assign (computed ?from ?to) 1))
)
```

We made some changes to the *goto_region* and now this action just moves the robot to the new location, following the simulated path computed by the *localize* action. Since the robot moves at a speed of one unit/second we assumed that the duration is the same as the distance calculated.

```
(:durative-action goto_region
    :parameters (?v - robot ?from ?to - region)
    :duration (= ?duration (distance ?from ?to))
    :condition (and (at start (robot_in ?v ?from))
            (at start (= (computed ?from ?to) 1)))
    :effect (and
      (at start (not (robot_in ?v ?from)))
      (at end (robot_in ?v ?to))
      (at end (visited ?to))
      (at end (increase (totalcost) (distance ?from ?to)))
      (at end (assign (computed ?from ?to) 0))
    )
)
```

In the problem file, the only modification is related to the addition of two initial conditions: the *totalcost* and *actcost* that are set to 0. This allow us to optimize the totalcost, that is the goal of the assignment.

```
(:init
    (robot_in R2D2 r0)
    (=(totalcost) 0)
    (=(actcost) 0)
)
```

Chapter 3

External Solver

In order for the domain and problem files to work we had to modify the file VisitSolver.cpp.
Here we added a function called *localize*. This function to be called needs to know the regions from where and to where the robot is moving. The value that the function returns will be the total cost, which will be passed to the planner.
We also used a *Landmarkcheck* function that takes the X and Y position of the robot, which are used for checking the distance between the robot and each landmark; if the distance is below a certain threshold the robot is considered above the landmark and its ID is returned to the calling function.

3.1 Localization Implementation

The reader must keep in mind that we assumed that the robot is omnidirectional and X, Y and θ evolve independetly from one another,so that is why θ was ignored.
The *localize* function has different uses:

- First of all it calculates the distance between the regions where the robot is moving. This is a simple euclidean distance, calculated for every coordinate. The distance is used to calculate the evolution of X and Y for each step and the number of steps.; we considered each step duration as 0.2 seconds.

- After it is done calculating the steps and steps' sizes we built a very simple EKF-like filter; we implemented a measurement error, based on the evolving P matrix (from a starting covariance one), that updates at each step if a landmark has not been found. This adds a Gaussian noise the the measurements (the pre-computed update steps) with a PDF that depends by the P matrix.

- Using the robot real position, not the simulated one with introduced errors, we measure the distance between the robot and each landmark, at each step. If the distance between the robot and a landmark is below a certain threshold, the robot

position will be set to the one of the landmark with some added noise depending on the initial covariance matrix; the P matrix also will be resetted to the initial value.

- While navigating from the initial to the goal waypoint, at each step, the position of the robot and the P matrix will evolve, unless a waypoint is encountered. The simulation still keeps track of the real position of the robot, beside the one that uses the simulated odometry sensor with noise addition.

- The system is able to keep track of the P matrix, even between different calls of the *goto_region* action from the planner, similarly to how a real KF would behave during navigation.

Behaviour of the robot

The robot starts in the initial point with an error introduced by the initial covariance matrix. It then calculates the distance to the next point and after that it starts moving. Once it reaches the estimated position, which will differ from the target position because of the noise introduced, it calculates the distance again from the next point using the updated the covariance matrix. It keeps updating the covariance matrix until he meets the last target. In case it meets a landmark the covariance matrix would reset to the initial one.
The planner keeps track of the distances estimated by the robot at each intermediate goal and sums them to have the total cost of the plan.

3.2 Results

We can see that the estimated distances vary from the actual ones and so does the cost of the plan. This is the uncertainty introduced by the noise.

```
Number of literals: 10
Constructing lookup tables: [10%] [20%] [30%] [40%] [50%] [60%] [70%] [80%] [90%] [100%]
Post filtering unreachable actions:  [10%] [20%] [30%] [40%] [50%] [60%] [70%] [80%] [90%] [100%]
No semaphore facts found, returning
No analytic limits found, not considering limit effects of goal-only operators
Not looking for earlier-is-better time-dependent rewards: no goal limits
None of the ground temporal actions in this problem have been recognised as compression-safe
Initial heuristic = 16.000
b (14.000 | 1.000)b (13.000 | 3.002)b (12.000 | 3.002)b (10.000 | 4.003)b (9.000 | 7.037)b (8.000 | 7.037)b (6.000 | 8.038)b (5.00
0 | 11.056)b (4.000 | 11.056)b (2.000 | 12.057)b (1.000 | 15.201);;;; Solution Found
; States evaluated: 35
; Cost: 11.194
; External Solver: 0.000
; Time 0.04
0.000: (localize_simulation r2d2 r0 r4)  [1.000]
1.001: (goto_region r2d2 r0 r4)  [2.001]
3.003: (localize_simulation r2d2 r4 r3)  [1.000]
4.004: (goto_region r2d2 r4 r3)  [3.032]
7.038: (localize_simulation r2d2 r3 r2)  [1.000]
8.039: (goto_region r2d2 r3 r2)  [3.017]
11.057: (localize_simulation r2d2 r2 r1)  [1.000]
12.058: (goto_region r2d2 r2 r1)  [3.143]
rick@RIKY-DESKTOP:/mnt/c/Users/rick/Desktop/visits$
```

Figure 3.1: Results of the planner