Report Assignment 2 AI4RO

Group Participant:

- Ermanno Girardo ID: 4506472
- Enzo Ubaldo Petrocco ID: 4530363
- Alessio Roda ID: 4458313
- Laura Triglia ID: 4494106

1. Introduction

The aim of this assignment is to become acquainted with the motion planner (in particular popf-tif), the external solvers and semantic attachments between pddl and c++. Starting from the previous assignment of the bar waiters scenario, now we focus on the minimization of the cost of the motion, in order to make them optimal to complete the tasks.

What we have implemented for this assignment are some modifications to the VisitSolver.cpp, the dom1.pddl and prob1.pddl. We also added an external library to evaluate the Extended Kalman Filter, so to do that we also had to modify the MakeFile in the “build” folder and the CMakeList in the “src” folder.

2. Preliminary steps

In order to download the planner, please download it from this link https://github.com/popftif/popf-tif , here you can see all the steps required to compile the source code.

Warning: In order to execute the planner with the ExternalSolver.cpp the syntax is the following:

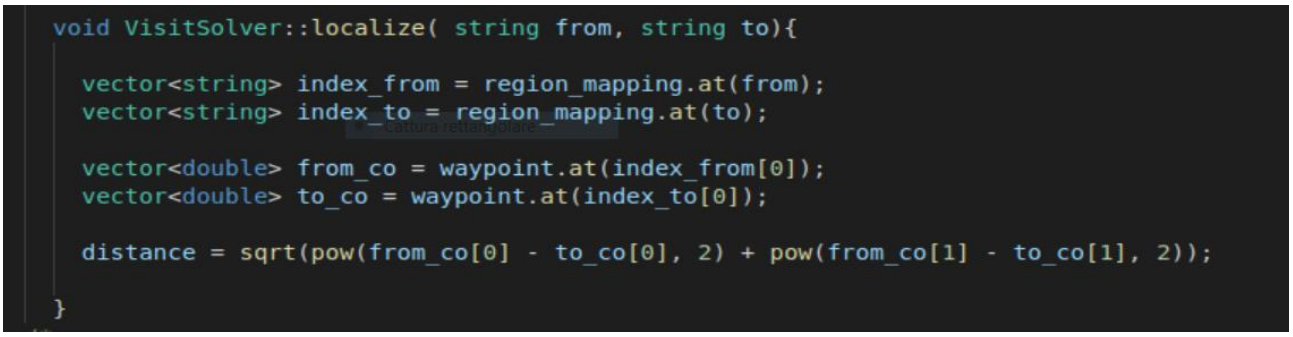
```
./popf3-clp -x domainfile problemfile externalsolver [inputexternalsolver]
```

Where domain file is dom1.pddl, problemfile is prob1.pddl, the externalsolver is the libVisits.so that is the compiled executable file of our c++ code and the inputexternalsolver is region_poses.

3. Localize function and distance evaluation

First of all we declared the localize function into the VisitSolver.h library, then we implemented it into the VisitSolver.cpp file. This function takes as input the regions (as string parameters) in which robot comes from and in which it has to go.
The region_poses file maps each region with the waypoint associated; in order to serve a table, the waiter must reach exactly the three components associated to each waypoint [x, y, theta components].

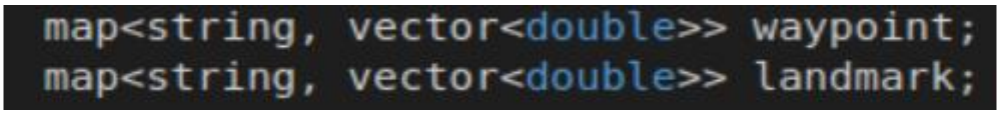
Once the two regions are taken, the localize function computes the Euclidean distance between the two regions and stores it in a global variable (distance).
In the function calculateExtern the cost associated with the motion of the robot is updated with respect to the evaluated distance in the localize function.



```
void VisitSolver::localize( string from, string to){

  vector<string> index_from = region_mapping.at(from);
  vector<string> index_to = region_mapping.at(to);

  vector<double> from_co = waypoint.at(index_from[0]);
  vector<double> to_co = waypoint.at(index_to[0]);

  distance = sqrt(pow(from_co[0] - to_co[0], 2) + pow(from_co[1] - to_co[1], 2));

}
```

Figure 1: localize function

In general when the solver is loaded with the loadSolver function, the files waypoint.txt and lendmark.txt are taken into account by the c++ compiler and the data are stored into two map structures:



```
map<string, vector<double>> waypoint;
map<string, vector<double>> landmark;
```

Figure 2: Map structures for waypoint and landmark

4. Durative-action localize

This is a durative-action which as to update the act-cost function in the dom1.pddl file.
In order to do this it is triggered by a flag that is activated in the goto_region durative-action.
It also set the triggered function to 0, that was previously in the goto_region durative-action

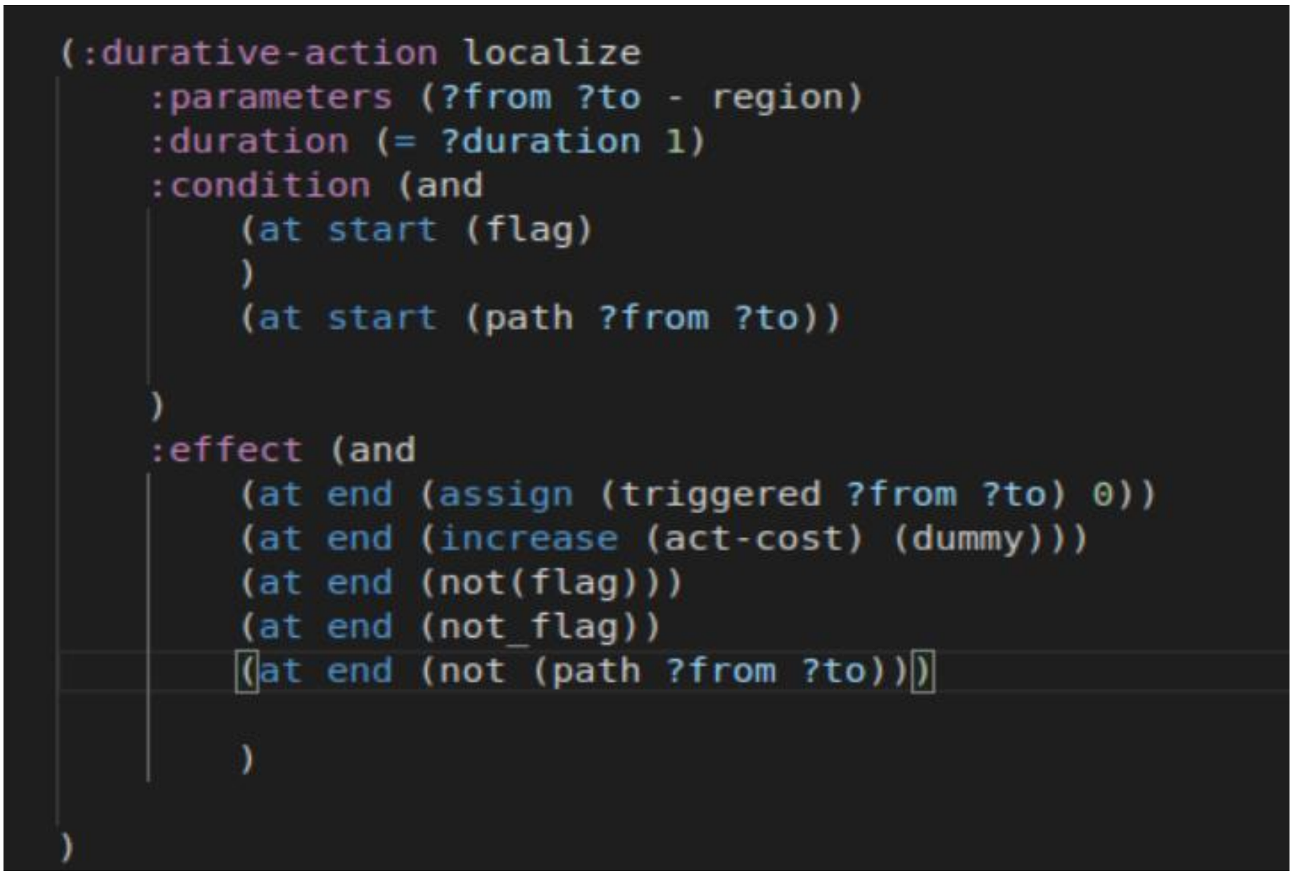

```
(:durative-action localize
    :parameters (?from ?to - region)
    :duration (= ?duration 1)
    :condition (and
        (at start (flag)
        )
        (at start (path ?from ?to))

    )
    :effect (and
        (at end (assign (triggered ?from ?to) 0))
        (at end (increase (act-cost) (dummy)))
        (at end (not(flag)))
        (at end (not_flag))
        (at end (not (path ?from ?to)))

        )

)
```

Figure 3: durative-action localize

The cost of the motion in the c++ file is stored in the dummy variable, so when this durative-action is executed it increases the act-cost of dummy.

5. Extended Kalman Filter

To compute the EKF we used a pre-existed library that you can find at this link https://github.com/shazraz/Extended-Kalman-Filter .

From the folder you can download there, we have included the Eigen folder and the kalman_filtr.h and kalman_filter.cpp files into the src folder. So, in the VisitSolver.h and VisitSolver.cpp files we implemented the startEKF function, in which are initialized all the matrices and the position vectors in order to compute the model and measurements uncertainty within the Extended Kalman Filter, that are added to the cost of the motion. Inside the startEKF function after the declaration we called the methods of the KalmanFilter class that evaluates the prediction and the updating of the EKF's matrices.

We initialized 5 matrices, F, P, H, Q and R.

- F= transition matrix

- P = the second one the initial covariance
- H = the Jacobian matrix
- R = the process noise covariance matrix
- Q = H*R*Ht

In order to evaluate the measurement (z vector) we wrote the function nearestLandmark, that takes as input a string which contains the actual position and returns a vector<double> with:

0) The value of the distance from the nearest landmark
1) The heading
2) The x coordinate of the nearest landmark
3) The y coordinate of the nearest landmark

In the end, in order to make the simulation more realistic, we added some gaussian distribution zero-mean noise to the measurement (z vector).

The trace of the resulting covariance matrix is added to the motion’s cost.