Technion – Israel Institute of Technology



HW2

Autonomous Navigation and Perception

086762

|  |  |  |
| --- | --- | --- |
| Alon Spinner | 305184335 | alonspinner@gmail.com |
| Dan Hazzan | 308553601 | danhazzan@campus.technion.ac.il |

April 14, 2022

# Question 1 – Theoretical questions

Given:

And assume motion and observation models are:

{ where }

## Write the objective function explicitly for the above considered setting:

## Derive a recursive formulation that expresses :

Given:

## Provide similar derivations to the value function and discuss the differences

### Value function definition

### Recursive formulation

### Differences from the objective function

In the objective function formulation J, we assume a set of known actions. The value of uses a policy to create actions. As such, actions are generated online and can be optimized online.

# Hands-on tasks 1 – GPS

Given:

1. Mobile robot navigating in a 2D environment.
2. Motion and observation models
3. =

Where:

1. A given prior

## Implement a function *PropagateUpdateBelief*

We decided to go with a filtering approach and implement a Kalman filter (ProbabilisticRobotics chapter 3.2.1) for the following reasons:

1. is a linear function and corrupted by Gaussian noise.
2. is a linear function and corrupted by Gaussian noise.

We provide a pseudo code of the algorithm below:

Text, letter

Description automatically generated

Figure 1 Kalman Filter Implementation

## Implement *SampleMotionModel*

Given a robot state and action we generate the next state as such:

Where:

To sample from the distribution, we used Julia’s *MersenneTwister* generator

## Implement a function *GenerateObservation*

We generate an observation such:

Where:

Same as with the motion model, we sample the gaussian distribution using Julia’s *MersenneTwister* generator.

## Simulation of our ‘GPS’ model

### (i + ii) generating actions and observations

In this simulation, we generated actions and observations at each time step as follows:

Table 1 GPS simulation

for \_ in 1:T-1

x\_gt = SampleMotionModel(𝒫, ak, x\_gt)

z = GenerateObservation(𝒫, x\_gt)

#generate beliefs

x\_deadreckoning = PropagateBelief(x\_deadreckoning, 𝒫, ak)

x\_kalman = PropagateUpdateBelief(x\_kalman, 𝒫, ak, z)

The parameters used:

### (iii+iv) We achieved the following results

We present below the trajectory made by the robot (ground truth) against both beliefs.  
As expected, the dead reckoning beliefs expand with time, and the Kalman filter generated beliefs are superior to them. In fact, they are so accurate that their covariance is too small to display.

Chart

Description automatically generated with medium confidence

Figure 2 Ground Truth trajectory and dead reckoning propagated beliefs

Chart, bubble chart

Description automatically generated

Figure 3 Ground Truth trajectory and Kalman filter beliefs

# Hands-on tasks 2 – Beacons

Our map now consists of 9 beacons where with **known locations**, spread in a rectangular form where the bottom left corner of that rectangle is placed in .

## a.+b. Implement the observation model

The observations model with parameter set for a given beacon located at :

Where:

Notice that a successful measurement is a tuple of both the relative location and the index of the beacon. In practice, we loop over all the beacons to check which, if any, provides a successful measurement.

In this simulation, we assume only one beacon is in range of the robot at any given moment. As such, the robot may obtain at most 1 observation at each time step.   
Given that the beacon’s locations are known, this allows us to write a simpler function which returns the global location

While this function is different from what we were asked to provide in section (b), it does serve the same exact purpose as:

## c. simulation - only state estimation

### (i) Repeat the simulation from Hands-on tasks 1 – GPS

In this section we only simulated one trajectory .

The simulation’s parameters are as follows:

As expected, the dead-reckoning belief’s covariance expands with time, while the Kalman based belief’s shrinks quickly when a beacon is within range.  
We present the figures below:

Chart, scatter chart

Description automatically generated

Figure 4 Beacons Scenario: Ground Truth trajectory and dead reckoning beliefs

Chart

Description automatically generated

Figure 5 Beacons Scenario: Ground Truth trajectory and Kalman beliefs for dynamic measurement covariance

### (ii) Repeat the simulation from the previous section with fixed

We repeat the simulation from the previous section with a static measurement’s noise covariance:

To the naked eye, the resulting belief look the same.

Chart, bubble chart

Description automatically generated

Figure 6 Beacons Scenario: Ground Truth trajectory and Kalman beliefs for dynamic measurement covariance

### (iii+iv) Present and compare the errors between the ground truth and the estimations for each of the methods (dynamic vs. static covariance).

We drew the error norms , and a measure of the estimation confidence, , for each of the methods, showing they are very similar to the point where estimation with dynamic covariance provides no real benefit.

Chart, line chart

Description automatically generated

Figure 7 Beacon Scenario: Frobenius norm of errors against time

Chart, line chart

Description automatically generated

Figure 8 Beacon Scenario: trace of belief's covariances against time

## .d Active estimation

In this section we test 10 different trajectories each containing a repeating action for 100 time steps.

For each trajectory we estimated the objective function

Where the cost functions are:

The objective function was estimated in two ways for each we present a recursive formulation provided, for convenience at the first-time step

1. Maximum likelihood:

Where

This method allows us to draw the trajectory’s estimated evolution in time as there is only one hypothesis belief estimated for each previous belief.

1. Sigma points:

Where:

The sigma points and weights are calculated in the same manner as they are for Unscented Kalman filter [Kalman and Bayesian Filters in Python, Roger R Labbe Jr, 2020] in a function we created called .

We chose and .

Chart, bubble chart

Description automatically generated

Figure 9 Sigma Points example, Roger R Labbe Jr, 2020

On both cases, the 5th trajectory, , was deemed best, as the ‘beacon density’ along its path is the highest.

Chart, scatter chart

Description automatically generated

Figure 10 Trajectories and belief estimations for ML method

|  |  |
| --- | --- |
| Figure 11 J per trajectory for ML method | Figure 12 J per trajectory for sigma points method |

The objective functions values are higher in the sigma points methods, as we allow for measurement error to affect the update section of the Kalman filter (innovation is not zero as it is in ML). In the sigma points method, we average over worst estimations.