Simultaneous Localization and Mapping: A General Approach to Different Methods  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Veysel Erçağlar Yunus Atahan Uğur Can Kozan Melikcan Türkdemir

**I. INTRODUCTION**

Robots in millenium era were always popular. They were popular among both users and researchers. In mobile robots, self driving or observing from outside and processing inside were important. Under heavy research years, Simultaneous Localization and Mapping (SLAM) became extremely popular among researchers. SLAM is a method that on an unknown location, the agent is creating a map concurrently keeping the data of agent’s location. This technique allows a robot to behave like an intelligent being. SLAM is widely used in self-driving cars, and robots that built to make investigation on unknown places to people (Such as MARS). SLAM is preferred because with no prior knowledge robots are still making good progress. There are multiple SLAM algorithms on literature that are beneficial in particular case or not effective. Introduced algorithms for SLAM are as EKF SLAM, Fast SLAM, L-SLAM, GraphSLAM, LSD-SLAM, S-PTAM, ORB-SLAM, MonoSLAM, CoSLAM. There are other algorithms used for SLAM but in this paper, we will try to focus on three of them. At the end of this paper, the implementations will show their comparisons in terms of their efficiency, run time complexity etc.

**II.METHOD**

**2.1 Extended Kalman Filter SLAM**

One of the basic answers for the SLAM was offered by Cheeseman and Smith who processed the EKF to mutually represent the landmark position with the model.[1] It is a class of algorithms that uses Extended Kalman Filter for SLAM problem. EKF is used to estimate the pose of robot and position of landmarks in the map robot moves. Extended Kalman Filter steps is as follows:

* State Prediction:

Estimate new position of the robot

* Measurement Prediction:

Predicting the observation

* Measurement:

Getting real observation with sensors

* Data Association:

Check the difference between predicted observation and real observation that gathered with sensors

* Update:

Change current state (position) of the robot to next state according to the estimation made in data association step.

Samsuri et al. points that the runtime complexity of EKF is in worst case ***O(n3)***. [2]

Formally, the algorithm of Extended Kalman Filter as follows:

*Algorithm* **EKF (Problem , Initial Covariance)** *returns the corresponding updated data***{** *Start with Initial Covariance;  
 ObtainedData = Initial Covariance;  
 while true****{*** *Calculate the weights from Initial Covariance;  
 Consider the noise;  
 Get new measurements;  
 ⅋ = Update the state estimations;  
 Calculate the new covariance with obtained ⅋;  
 Guess the new state estimation and covariance for the* ***tt+1*** *step;  
 ObtainedData = Estimation and Covariance;  
 return ObtainedData;*  
 **}  
}**

**Code**

Code is written with help of the tutorials [4][5][6] and code below has two essential parts one is generic source function and the other part is the testing part.

import numpy as np

from kalman\_filter import predict

measurements = [0, 1.1, 1.9, 2.5, 3.7, 4.9, 6]

x = np.zeros((2,1)) # initial state (location and velocity)

P = np.eye(2,2)\*1000# initial variance

u = np.zeros((2,1)) # external motion

F = np.array([[1., 1.], [0, 1.]]) # next state function

H = np.array([[1., 0.]]) # measurement function

R = np.array([[1.]]) # measurement variance

for m in measurements:

print("{:6.4f}".format(\*predict(x,u,m,F,P,R,H)[0][0]),"\t", measurements.index(m))

import numpy as np

#x: initial state

#u: external input

#z: measurement

#F: next state matrix

#P: initial variance

#R: Measurement variance

#H: Measurement function matrix

#Q: Process variance

def predict(x, u, z, F, P, R, H=None, Q=None):

#INITIALIZATION

i\_p=np.eye(\*P.shape)

if H is None:

H=np.ones(x.shape)

if Q is None:

Q=np.zeros(P.shape)

#PREDICTION

x\_n=np.add(np.matmul(F, x), u)

P=np.matmul(np.matmul(F, P), F.transpose())+Q

#MEASUREMENTS

z\_n=np.matmul(H, x\_n)

err\_z\_z\_n=np.subtract(z,z\_n)

h\_t=H.transpose()

Knum=np.matmul(P, h\_t)

Kden=np.add(np.matmul(np.matmul(H, P), h\_t),R)

K=np.matmul(Knum,np.linalg.inv(Kden))

#UPDATE

x\_n=np.add(x\_n, np.matmul(K, err\_z\_z\_n))

p\_n=np.matmul(np.subtract(i\_p, np.matmul(K,H)),P)

return x\_n, p\_n

**Output**

0.0000 0

1.0995 1

1.8991 2

2.4988 3

3.6982 4

4.8976 5

5.9970 6

**REFERENCES**

[1] R. Smith and P. Cheeseman. *On the representation and estimation of spatial uncertainty.*

*Intl. J. of Robotics Research*, 5(4):56–68, 1987.

[2] Samsuri et al. *Computational Cost Analysis of Extended Kalman Filter in Simultaneous Localization and Mapping Problem for Autonomous Vehicle* 2015  
 Russel S. and Norvig P. *Artificial Intelligence: A Modern Approach Third Edition,* pg. 993 2016 Pearson

[3] Jaiswal, P., 2018, *Sensor Fusion — Part 2: Kalman Filter*, Retrieved from <https://towardsdatascience.com/sensor-fusion-part-2-kalman-filter>

[4] Kleeman, L., 2017, *Understanding and ApplyingKalman Filtering*, Retrieved from http://biorobotics.ri.cmu.edu/papers/sbp\_papers/integrated3/kleeman\_kalman\_basics.pdf

[5] LAARAIEDH, M., 2018, *Implementation of Kalman Filter with Python Language*, Retrieved from https://arxiv.org/ftp/arxiv/papers/1204/1204.0375.pdf