Pfinal

Efficiency of FastSLAM

- Proposal distribution

"FastSLAM 1.0" → "FastSLAM 2.0"
("Probabilistic Robotics book") modifies proposal distribution (The observations are taken into account)

- Map management
  - M particles
  - N map features (landmarks)

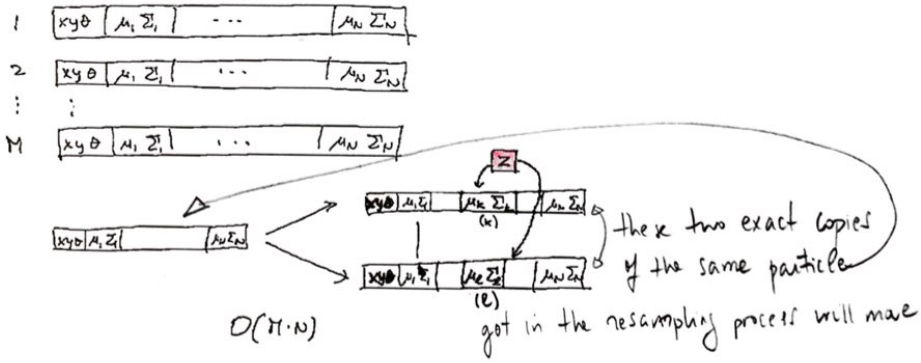
$O(MN)$

copy

The problem of the presented fast slam approach is that our proposal is only made based on the given control and this may lead to inefficiencies because if the control is very noisy this may lead to the midespread of the particles and only a few will survive in the end, because all the others do not fit very well into the observations of our robot.

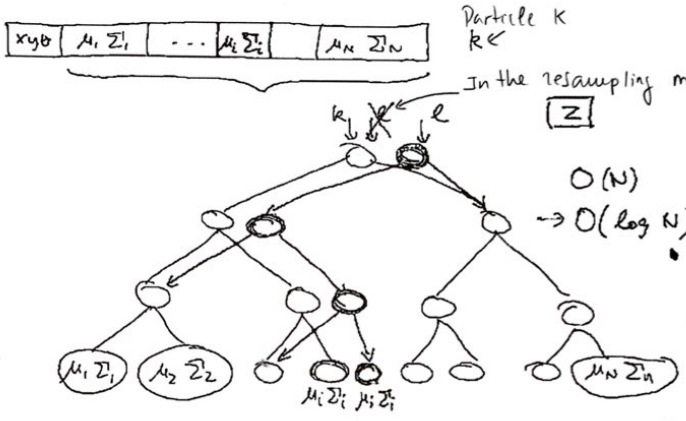
Each $z_t = \begin{bmatrix} r_t \\ \phi_t \end{bmatrix}$ got in the scan

Implementation: $O(MN)$



differently in the prediction step, which won't modify the map. When the observation occurs they will be updated differently. This measurement relates to an existing landmark in one particle, let's say landmark K, and another landmark, landmark L, in the other particle. If this particles are close, these indices, probably, are the same ⇒ $K = L$. So we modify the list of landmarks only in one place, where all the other entries stay the same

Let's think about putting all the landmarks into a balanced tree 5-7@7.2



Then, if there is at least one change, we need another ~~root~~, a new root, that the pointer e will point.

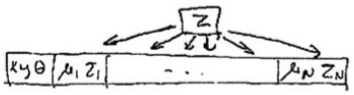
Whenever we need some element with a certain index we would now have a logarithmic time complexity for accessing it

---

Efficiency

$O(MN) \rightarrow O(M \log N)$

M: # particles N: # map features



Conclusions – FastSLAM

- Particle filters SLAM
- Each particle is <u>one path</u> plus <u>one map</u>
- Map features are independent (given the path)
  → one (independent) EKF per feature
  → In contrast to EKF SLAM: ⊕ no correlations
  → maintains dependencies only implicitly ⊖
- Each particle uses its own data associations
  → In contrast to EKF SLAM
- Solves both: offline and online SLAM.