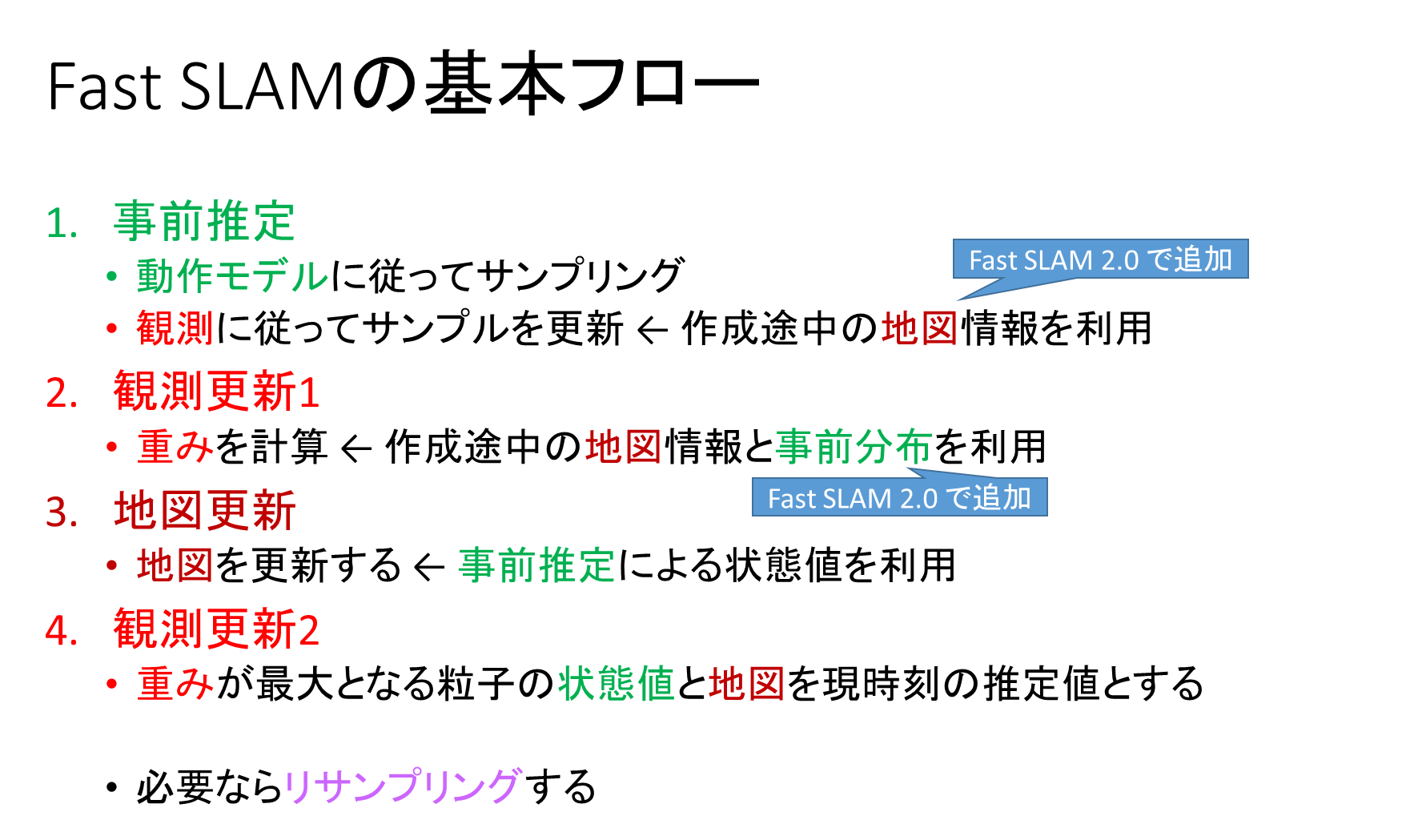
まずはFast SLAM の基本フローを押さえておきます．SLAMのアルゴリズムは長く，ごちゃごちゃとした数式を使うので，「何やら難しそうだ…」という印象を受けてしまいます．私はそう思っていました．ところが，基本的なステップだけを抽出すれば，以下の図のようなシンプルなものです．



自己位置推定で言うとことの「観測更新」の中ほどに「地図更新」が挿入されただけです．なるほど，当たり前っちゃ当たり前です．何もビビる必要はなさそうです．

さて，察しの良い方は各ステップに隙間があることに気が付かれたと思います．この隙間には，もう少しだけ詳しい説明が入ります．

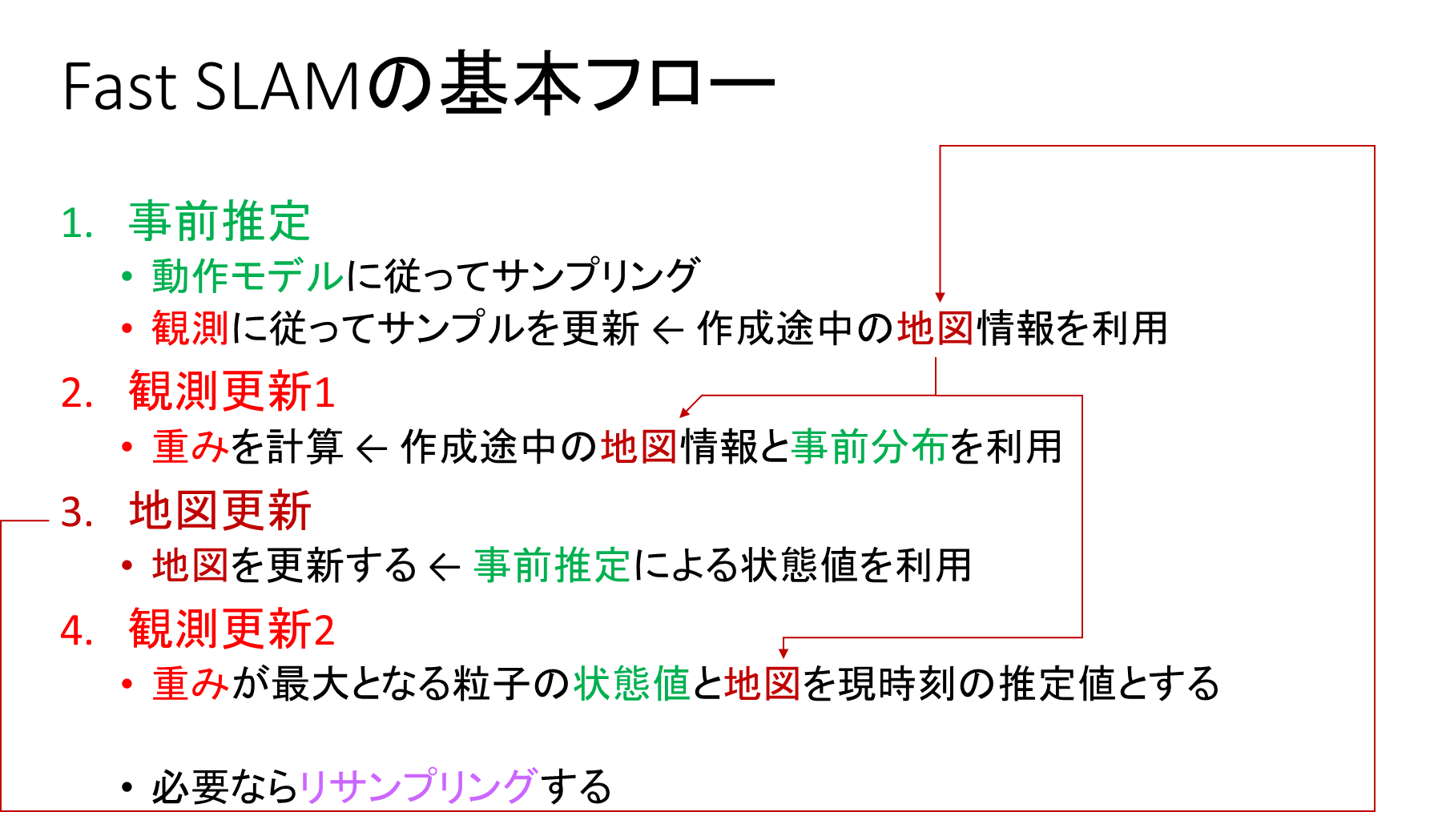


書いたとおりで，各説明のフォローは後述して，ここでは基本フローの掲載に留めておきます．ただ一点，大事な補足があります．「Fast SLAM 2.0で追加」というコメントが付与されている部分は，「Fast SLAM 1.0」では使わないものとなっています．ここが，Fast SLAM 1.0と2.0との間に潜む決定的な相違点です．詳細は後述することとして，ここでは先に進むこととします．

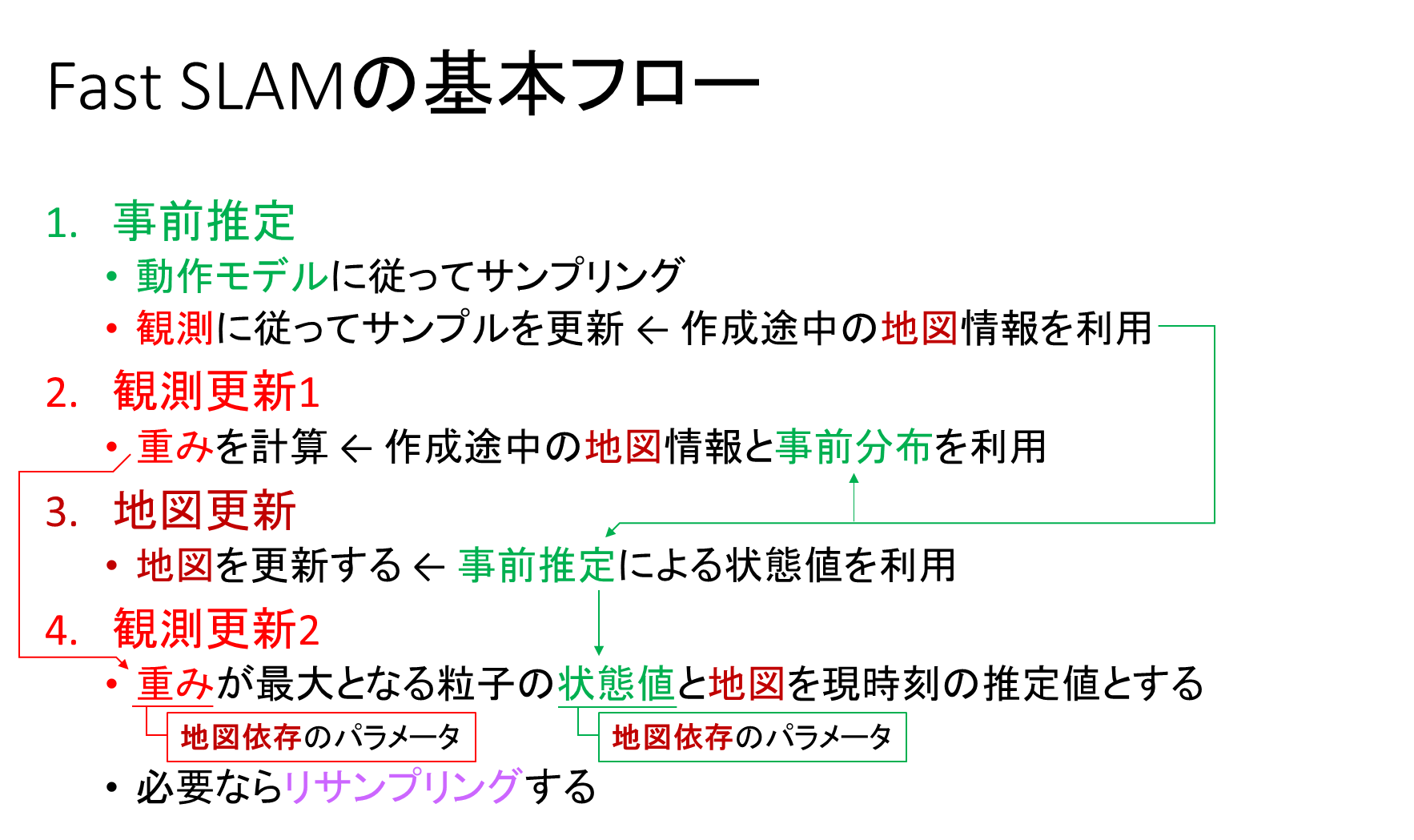
SLAM に潜在する問題

SLAM は便利で素晴しい技術ですが，過信は危険です．それを簡単に説明します．

下記Fast SLAM 2.0のフローを振り返ります．事前推定，重み計算の際に，観測の情報を使うことになります．この観測情報とのマッチングを行うのは，作成途中の地図です．



あれ，じゃあ地図はそもそもどこから来たんだっけ？となると，これは事前推定で得られた「まだ不確かな状態値」を基に計算されています．つまり，事前推定の誤差がそのまま「地図」の精度にダイレクトに影響するのです．そんな地図を使って次の時刻も事前推定や観測更新を行って，また地図を更新して，というお話を延々と続けるのです．

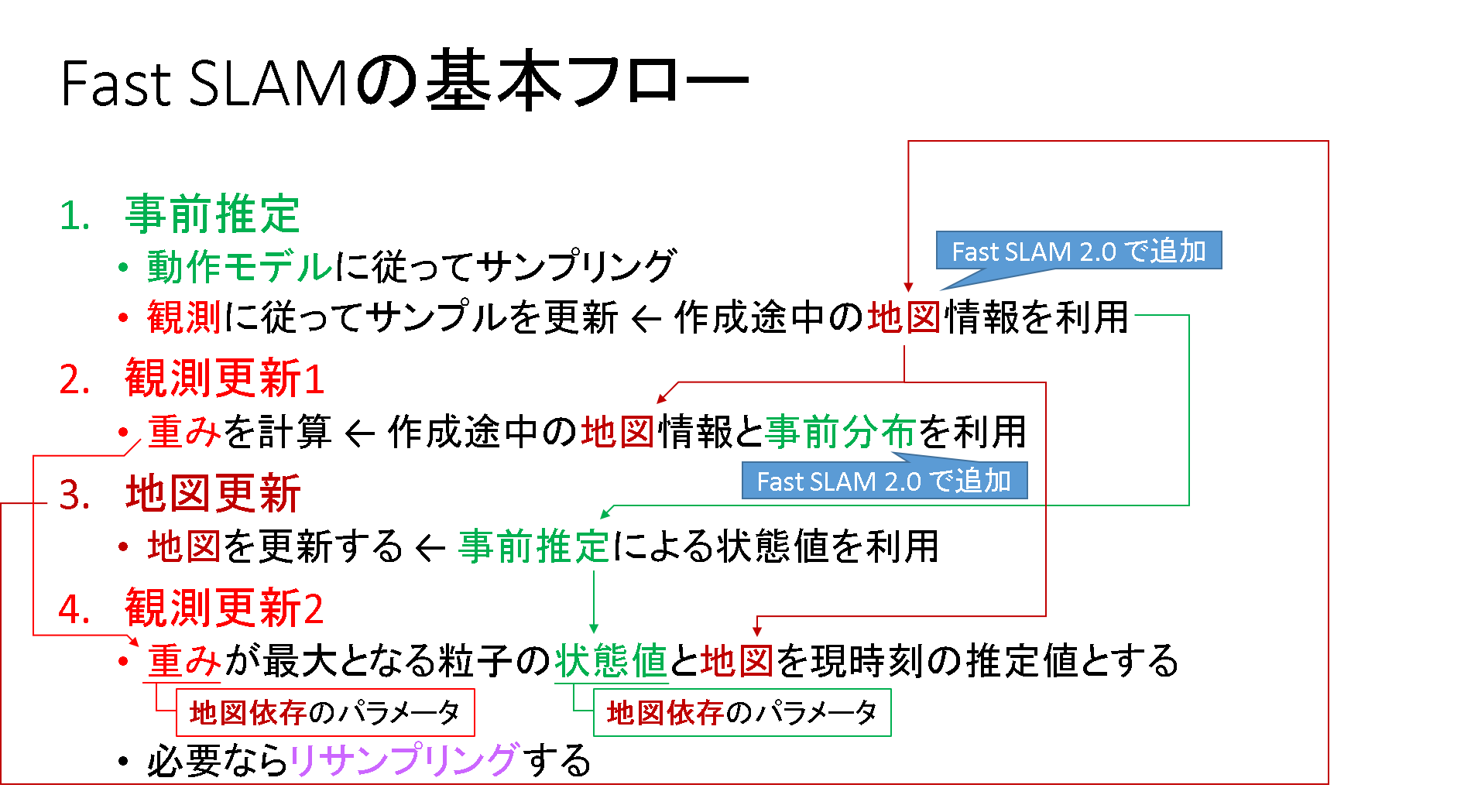


これはSLAM　が原理的に有する問題で，回避は難しいものです．レファレンスとなるものがない状態からスタートさせられて，結局最後まで答え合わせは出来ないのです．せいぜい出来てセルフチェックとカンニング程度です．

例えるならば，何らかの筆記テストを受ける場面を想定します．普通は自分の信念と知識にしたがって回答をします（事前推定）．でも，当方は相当な勉強不足なので，その回答にはかなりのばらつきがあって当たるかどうか良くわかりません．でもその途中で，少し戻ってちょいちょい答えの見直しくらいするのは何の問題もありません．そうすることで計算ミス等を見つけられるチャンスくらいはありそうです．ただし，解答はお預け，点数も分からない，自分の出した結論が合っているのか間違っているのかが，真の答えは最後の最後まで分からない，という結構理不尽なテストを受けている状態であると解釈できます．

そんな困った当方に唯一許された答えの確認方法が，カンニング（観測）といったところでしょうか．それが正答なのかどうかは分からないけど，観測された結果と照らし合わせて一致度が高ければ採用できそうだ，というものです（尤度）．周りが優秀な人ばかりならよいですが，適当に答えを書いてしまう人だっているかもしれない．この辺りが「センサの信頼度（誤差分散）」に繋がる話になります．ですから，Fast SLAMではLRFやRGBDセンサは優等生であると信じていることを大前提としており，この仮定が崩れると推定精度はガタ落ち必至です．適当に回答した人の答えをカンニングしたって，当てにならなそうですよね笑．

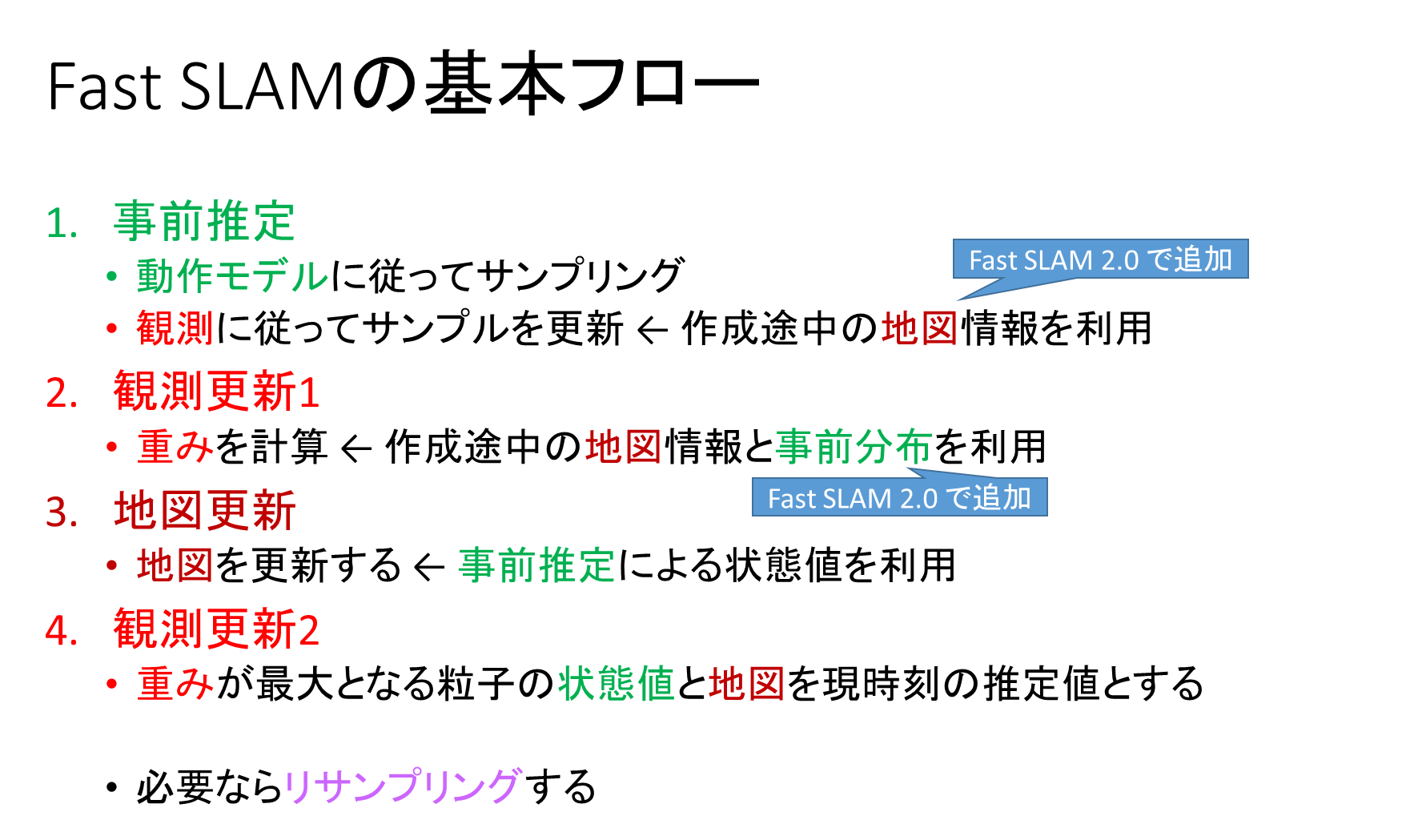
とにかく，Fast SLAMでは本質的に不確かなものだらけの世界で戦う必要があること，事前推定よりも観測情報の方に大きな信頼を寄せているということは，これを利用する上で理解しておく必要がありそうです．



そんな不確かなものを扱っているのだと考えると，パーティクルフィルタを使うことが如何に合理的であるかを余計に理解できます．何もかもが不確かなのだから，とりあえずパーティクルをばら撒きまくって，そこからたまたま尤度が高くなるものが見つかれば万々歳なのです．もちろん当たりの悪いパーティクルもわんさか出るでしょうが，その場合はパーティクルの多様性を失い過ぎない程度にリサンプリングするようチューニングしてしまえばいいんです．

ただし，あんまり適当にばら撒いて尤度の高いパーティクルが見つけられないというのも癪なので，ばら撒き方の指針として「動作モデル」を与えているのです．

更にFast SLAM 2.0 では，事前推定の段階で，より確実に観測尤度の高いパーティクルを得たいがために，わざわざ「事前分布」を近似するパーティクルの位置を「観測更新」によって移動させるような処理まで追加しています．



文献によれば，確かに数式上は問題なく展開されているようなのです．なんか，あの条件付確率の式展開を見ていると，もう何でもありなんじゃないかと思えてきてしまいます笑．

（もちろんそんなことはなく，満たすべき仮定の範疇で数式が展開される必要はあるわけで，それが守られているのは分かるのですが^^;　その結論として，「事前推定」の式を「事後推定」の形式に変換できちゃいました！なんていう破壊力抜群の事実を目の当たりにすると，これまで勉強したベイズフィルタの基本式ってなんだったんだろう？と言う感情が湧き出ずにはいられない…^^;）

もういいですね笑．結局，結論はこうです．

* SLAM の基本フローとしては，自己位置推定のフローに「地図更新」のステップが挿入されただけだよ．
* SLAM は便利だけど，地図には推定誤差が載っちゃうし，推定結果にも地図の誤差は載っちゃうで，これは結構大きな問題なんだよ．
* でもそんな未知の環境でも，パーティクルと観測をうまく使えば，そこそこ当てはまりのよさそうな状態推定はできそうなんだよ．センサの精度が良ければね．

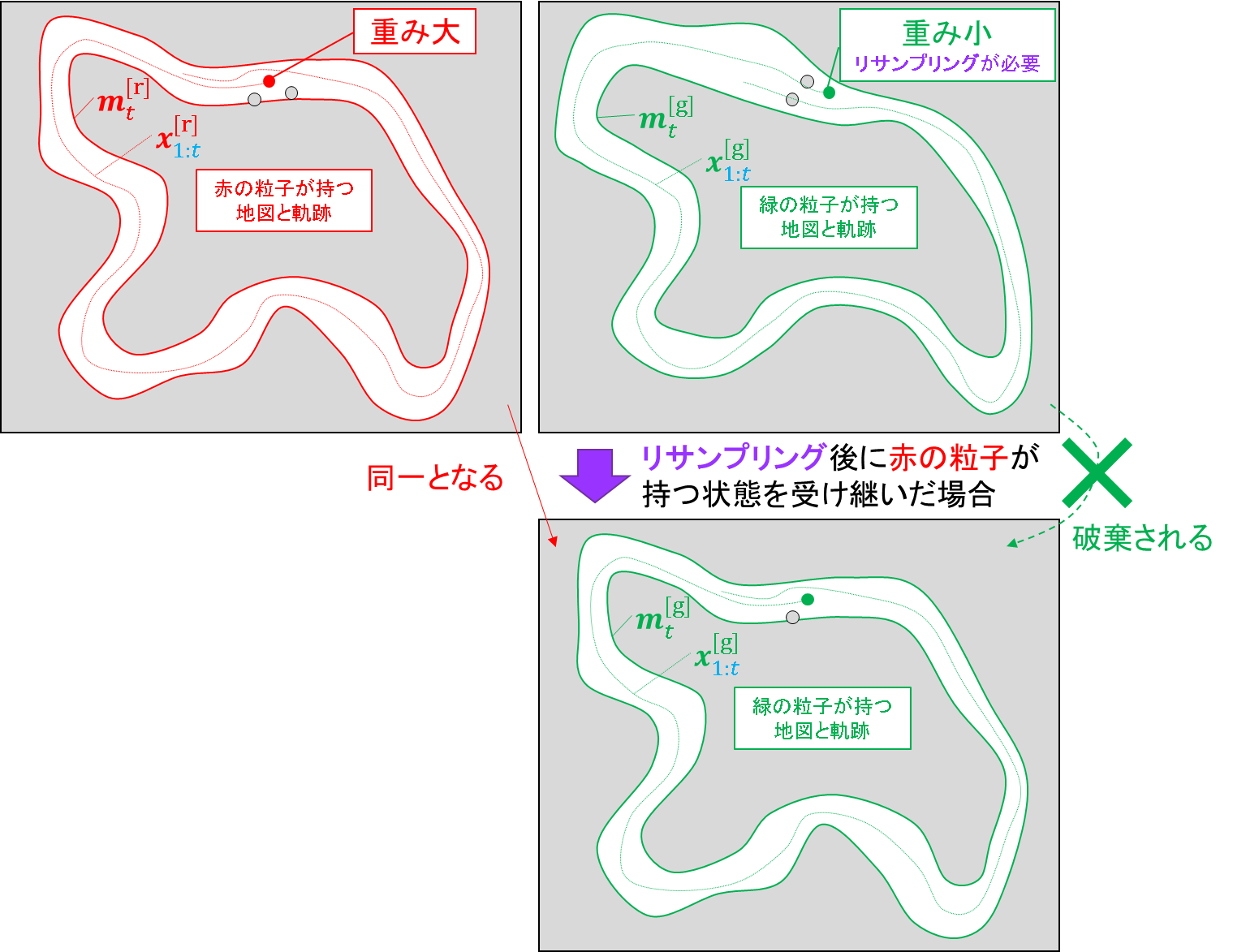
と言うことが分かれば，ここは良いかなと思います笑．「簡単に」とか言っておきながら，しょっぱなから色々止まらなくなってしまいました．長文失礼致しました．

## ケーススタディ

困ったときは思考実験です．本当はガチ実験ができればいいのですが，小生しがなきサラリーマンゆえ御容赦頂きたく…^^;

さて，少し前にいくつかの粒子（赤，青，緑）を使って地図を推定する問題を図示しました．このうち，地図的に見て当てはまりのよさそうな赤の粒子と，当てはまりの悪そうな緑の粒子についてピックアップしてケーススタディをしてみようと思います．

* 赤の粒子
  + ある時刻において赤の粒子の重みが大きかったとします．すると，次の時刻もこの粒子の持つ情報（軌跡，地図）は変わりません．
  + ここで言う「変えない」ということについては，「何もしない」ということではなく，「敢えて情報を保持するのだ」という重要な選択を行っていると考えるべきです．この根拠は本節で後述します．
* 緑の粒子
  + ある時刻において緑の粒子の重みが小さく，この粒子がリサンプリングの対象になったとします．すると，次の時刻以降はこの粒子の持つ情報が，他の粒子の情報にごそっと置き換わることになります．
  + 今，その置き換える対象の粒子が赤の粒子であったとしましょう．その場合，「緑の粒子の情報を赤の粒子の情報で上書きする」という選択を行ったと言うことになります．



こいつ，何を「当たり前なこと」を言っているんだ，と思われるでしょう．その通り，「当たり前も当たり前なこと」なのです．ですが，この「当たり前なこと」をしっかり頭に刷り込んでから，議論を進めないと，当方はすぐに迷子になってしまうのです ^^;

さて，この当たり前過ぎる現象に対して，ここで重要になりそうな論点は2つあると考えました．いずれもリサンプリングに関わるものです．

* 更新の処理実行の判断基準
  + 「重み」をリサンプリングの指標にすることが，「完全SLAM問題」を解く上で妥当なことなのか？
* 更新の処理そのもの
  + リサンプリングをする，しないという処理が，本当に「完全SLAM問題」を解くことに相当するのか？

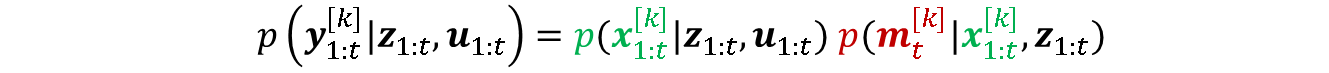
リサンプリングなんて今まで散々扱ってきたのに，何でわざわざこんなことを気にするのか？面倒くさい奴だな，って感じですよね^^;

こんな疑問が沸いた理由としては，Rao-Blackwell 化を施す前の，単純マルコフ性を仮定したパーティクルフィルタでは，リサンプリングに関わる処理はあくまで「オンラインSLAM 問題」を解くために行われていたこと，粒子が近似するのは「分布」であったこと，が気になったからです．

以前のパーティクルフィルタでは，過去の軌跡全てを考慮したりすることはなく，ただ毎時刻最新の状態を更新すればことが足りていました．その上，パーティクルが近似するのはあくまで「分布」です．リサンプリングによって「推定値」そのものを置き換えるようなことはなく，あくまでパーティクルの「分布」に修正を施す仕組みでしかなかったはずです．こういった理由から，Rao-Blackwell 化前に扱っていた，単純マルコフ性を仮定した自己位置推定のためのパーティクルフィルタの枠組みでは，「完全SLAM 問題」への対策にまで上り詰めさせることができないのです（少なくとも，私の貧弱な思考力では…）．

こういうことを考えると，元来「オンラインSLAM問題」を解くための，しかも「分布」を修正するにすぎなかったリサンプリングという手法を，「完全SLAM問題」を直接解くための手段としてしまうことは，リサンプリングにとって荷が重いのではないのか？という疑問が沸くのです．

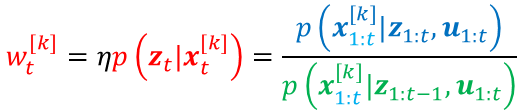
一方， Rao-Blackwell 化によってFast SLAM を定式化する上で，「完全SLAM問題」を解くことを前提としており，ちゃんと証明に成功しています．実際，推定後の状態についた時間に関する添え字は「1:t」と言う風に軌跡を保持したものとなっていて，式展開上も整合性が取れているのです．



うーん．なぜだ，なぜこんなことができるんだ？これを紐解くために，上述した「更新の処理実行の判断基準」と「更新の処理そのもの」の妥当性について順番に見ていこうと思います．

### 更新の処理実行の判断基準

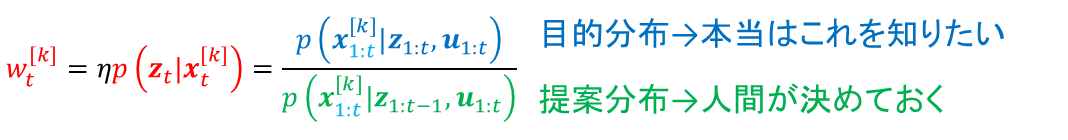
Fast SLAM においても，パーティクルのリサンプリングの判断基準として，「重み」を用いています．ただし，重みの時間添え字は「t」です．



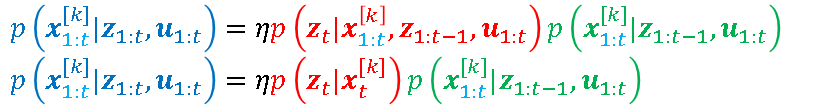
現時刻で求めた「重み」だけを使って，「過去に蓄積した全ての軌跡」を丸ごと置き換えるなんてことをして本当に良いのでしょうか？ちょっと荷が重すぎやしないのだろうか？という疑問が沸きます．これの妥当性を思考実験的に検証しようと思います．

ここでのゴールです．一見，時刻tについてしか考慮してなさそうに見える「重み」が，ちゃんと時間「1:t」を考慮したうえで算出されているんだという確信が得られれば良いと考えています．さすがに時間「1:t」の情報が集約されているとなれば，「完全SLAM問題」を解くための指標として差し支えないとは思うのです．

そんな思いを胸にProbabilistic Robotics における「重み」の式展開を眺めていると，時間インデックス「t」と「1:t」が交差するところが一点だけあるのです．



これだ．多分ここがミソだと思いました．これはまさにベイズフィルタの定義式から導出される部分です．なぜならば，下記の展開が成立するからです．



ロボットの位置xの時間インデックスが「1:t」から「t」に置き換わりました．現時刻の観測は現時刻の位置にしか依存しないからです．でも，提案分布も目的分布も，本当は過去の全ての軌跡の情報を元に算出しているんだよ，という正論的な仮定を残しておくのです．

あとで

2点しかないので，書いてしまいます．

* 事前推定
  + Fast SLAM 2.0 では，事前推定の時点で「観測」更新に相当する処理が追加されます．Fast SLAM 2.0においては，本質的に「観測」を強く信じる機構となっています．
    - 特徴ベースの場合，本更新は「拡張カルマンフィルタ」で行います．
    - 格子ベースの場合，本更新は「スキャンマッチング」で行います．
* 観測更新1
  + Fast SLAM 2.0 では，尤度関数に「事前推定」の不確かさを陽に組み込ませます．これにより，尤度関数の分散を広げることができ，より広範囲の粒子を生存させることが出来ます．
    - 特徴ベースの場合，尤度関数更新は「拡張カルマンフィルタ」で行います．
    - 格子ベースの場合，尤度関数更新は「スキャンマッチング」で行います．