

自律移動と最適制御

大域計画, 障害物回避, 機械学習, 潜在空間,
伝統的な制御の統一的な理解のために

千葉工業大学 上田 隆一

This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License.](#)



今日やること

- 本日の話の動機（これだけで終わらないようにします）
- 探索問題と最適制御
- 確率モデルを使った制御問題の記述と様々な分野の関係

この発表の動機: いろんな人がいろんな人と話がかみ合わない

- 制御ゴリゴリの人/学習の人/確率ロボティクスの人で互いに会話が合わない
- 制御ゴリゴリの人と制御ゴリゴリの人が話が合わない
 - なんかそんな座談会の記事を見たような気がします
- 制御をポントリヤーギンの最大（最小）原理から考えている人とベルマン方程式から考えている人で会話が合わない
話が盛り上がってほしい（なんなら「上田あいつアホだ」でもいいので）

話が盛り上がる鍵: 一般化・抽象化

- 一般化すると、もしかしたらほかの分野のことも分かるかもしれない
 - 抽象化の山を登り、具体的な課題に下山
 - 高く上るとどこにも降りていける
 - 遭難しそうとかそういうのはナシで
 - 話題の模倣学習やVLAとかも分かるかもしれない
 - ということで・・・
 - 今回は自律移動なので、そこから問題を抽象化してみましょう
- 分かること
 - 現在、自律移動研究の主流の方法はどうなのか？
 - 隣接分野はどうなっているのか？

今日の話がうまくできるかどうか分からぬ いので

- 考えはこの本の9章に書いてあります
 - 誰も9章まで辿り着いてないのではないか?
 - 青色の本でもオレンジ色の本でも同じ現象
 - お食事中の方すみません

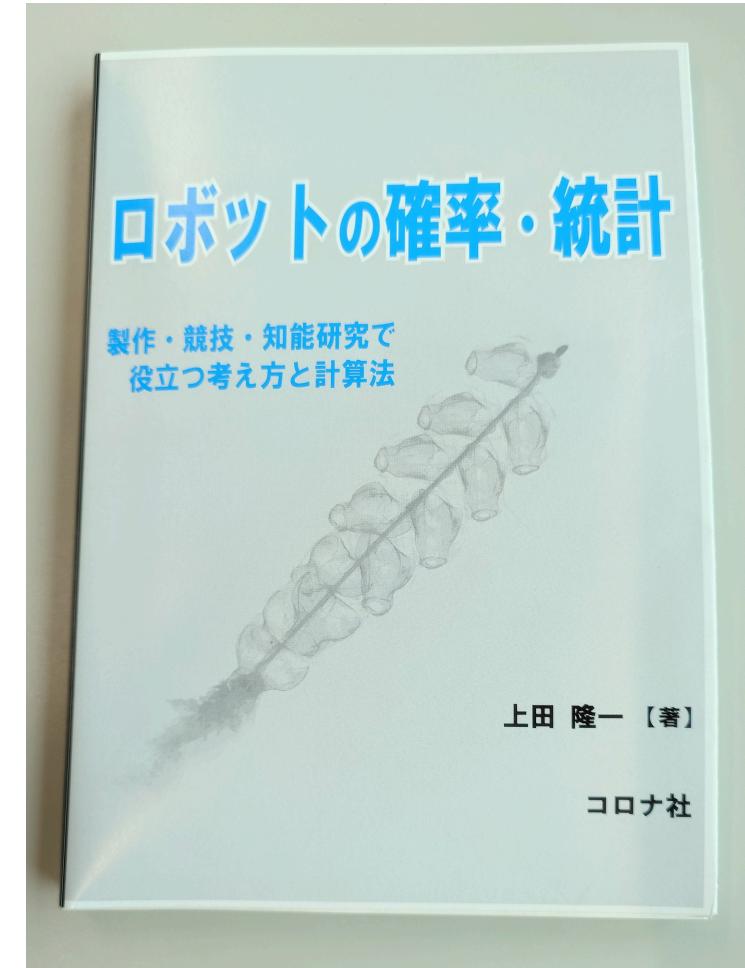


上田 隆一
@ryuichiueda

プロモーションする Ⓢ ...

「ロボットの確率・統計」にみんなが制御といってるものと強化学習って言ってるものと、今うんこにいくかどうか悩むことは同じ式から導出できる同じ問題なんだよ仲良くしようね」

って書いたのでみんな買って読んで仲良くして欲しい（せんでん）



探索から制御問題へ

話の出発点: ロボットの大域経路計画（探索）

- 実空間中にロボットの現在地から目的地まで線を引く
 - ダイクストラ、A*、RRT、...
 - いまだ現役
- 一方、移動ロボットや自動車を自律移動させることは難しい
 - 自己位置推定がずれる（今回は直接扱わず）
 - 障害物をうまく避けれない
 - • • •

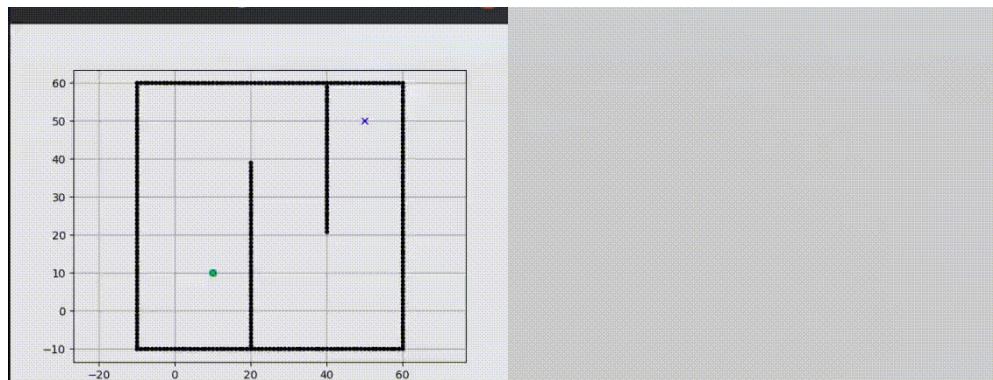
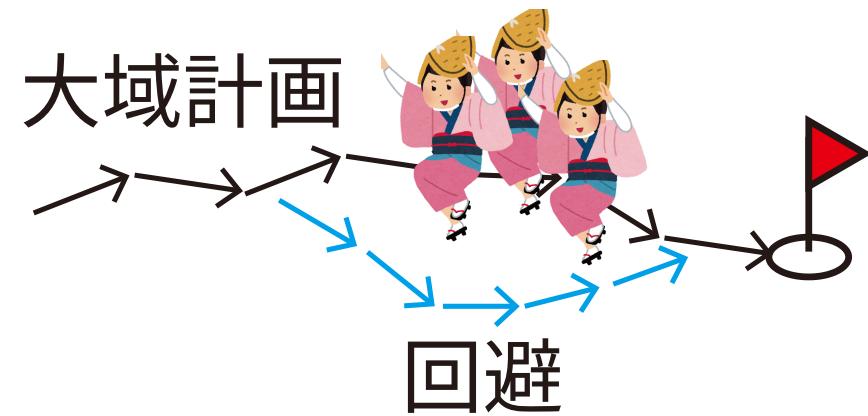


図: [AtsushiSakai/PythonRobotics](#)で

一般的なアプローチ: 問題の分割

- 大域計画+諸問題の解決
 - 大域計画+障害物回避
- だいたいの場合、これで問題ない
 - 現場であれば問題が出たらまた潰せば良い
- 本当にこれで問題ない?
 - 理論上解決できないのか?
 - 根本から解決する方法はないのか?

⇒問題を整理しましょう



ナビゲーション（大域+局所）の問題 = 制御

- 探索は手段であって、制御問題と考えることが妥当
- 制御: 好ましく無い状態 x を、好ましい状態の集合 \mathcal{X}_f に持っていく
 - なにか力 u を加えて

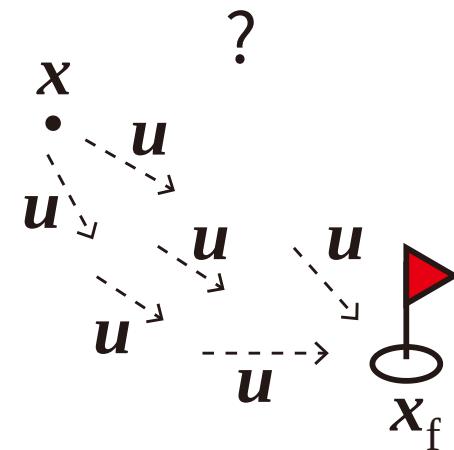
現象	制御
ロボットが目的地でない場所に	目的地にいる状態にしたい
機械が振動している	振動してない状態に戻したい
ライントレースのロボットがラインからはずれた	ライン中央に戻したい
洗濯物が洗濯機の中に	畳んで収納したい

大雑把に考えると全部同じ

大雑把な枠組みの制御問題: 最適制御問題

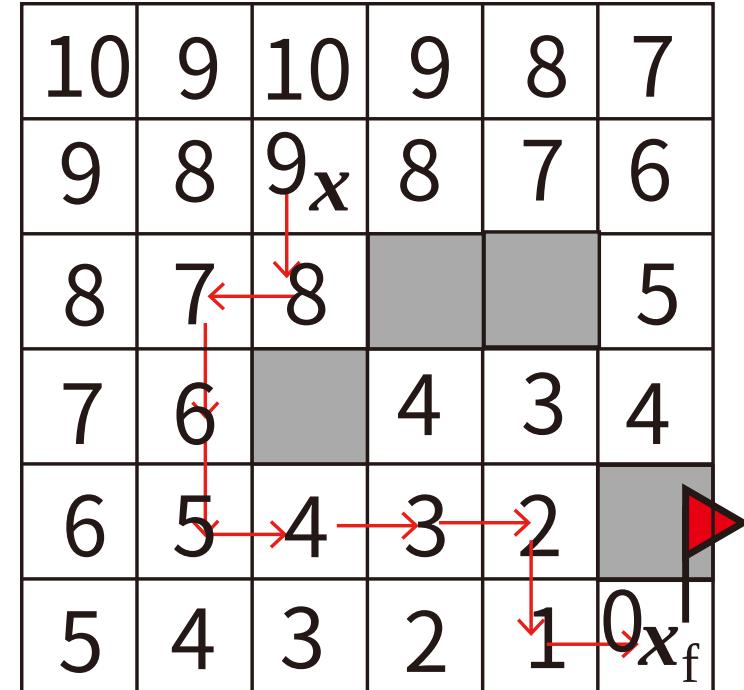
マルコフ決定過程 (MDP) でもあるけど最適制御と言います

- 状態 x を終端状態の集合 \mathcal{X}_f の任意の要素 x_f まで導きたい
 - x には速度や時間、制御不可能なものの変数も入れられる
- $u \in \mathcal{U}$ という力をかけると次の時刻に状態 x が x' に遷移
 - $x' \sim p(x'|x, u)$
 - ※とりあえず離散時関係で考えます
- 状態遷移にはコスト: $\ell(x, u, x')$
 - 「時間消費」、「エネルギー消費」、「危険性」などを点数化
- 終端状態にも点数: $V(x_f)$
- コストの総和 $J = \sum \ell + V(x_f)$ を最小化したい



単純な（将棋盤上の）移動の問題の場合

- 問題
 - x : ロボットの位置（2次元）
 - u : 前後左右（状態遷移: 決定論的）
 - $\ell(x, u, x') = 1\text{step}$
 - $V(x_f) = 0$
- 右図: 問題と解の例
- 重要
 - 状態ごとに終端状態までのコストが見積もれる
 - 最良の u を選ぶと 1 ステップのコストとコストの見積もりの減少が釣り合う
 - いまの状態 x と解は無関係



最適制御問題の解の性質（どう解くかという話とは別）

- 「釣り合い」の式: ベルマン方程式
 - $V^*(x) = \min_u \langle V^*(x') + \ell(x, u, x') \rangle_{p(x'|u,x)}$
 - $\langle \cdot \rangle_p$: 期待値
 - V^* : 最適状態価値関数（最適な値関数）
- V^* から各状態で最適な行動 $u^* = \pi^*(x)$ が得られる
 - π^* : 最適方策
- もう一つ重要: 最適でなくても方策 π に対して V^π が存在
 - $V^\pi(x) = \langle V^\pi(x') + \ell(x, u, x') \rangle_{p(x'|u,x)}$
 - ここで $u = \pi(x)$

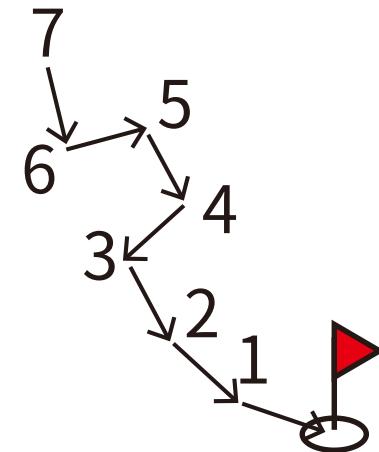
「あとどれくらいで仕事が終わるかな？ という見積もりが正確だと正しい行動ができる」

10	9	10	9	8	7
9	8	9 _x	8	7	6
8	7	8			5
7	6		4	3	4
6	5	4	3	2	
5	4	3	2	1	0 _{x_f}

探索で得られる大域計画の解の性質

- 1本道の方策 π ができている（右図の矢印）
 - 最適である保証はない
 - 経路からちょっと外れたところは基本無視
 - $p(x'|x, u)$ を決定論的に解釈
- V^π は方策を求めるついでに計算されている
 - 概算
 - 途中の計算で周辺の V も求められるが捨てられがち
- かなり雑に解いている

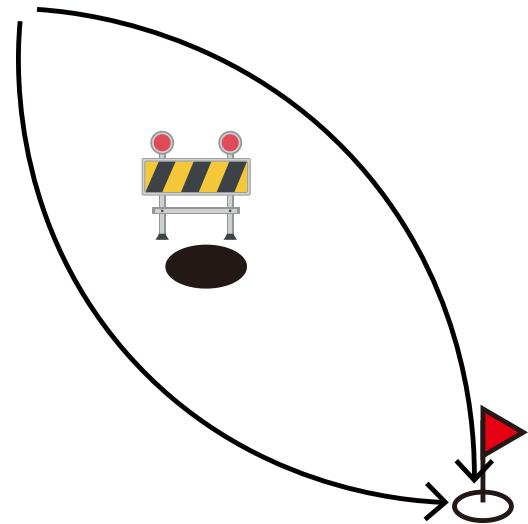
きっちり方策通りにロボットが動けば問題はない、が



よく問題になること: 経路のチャタリング

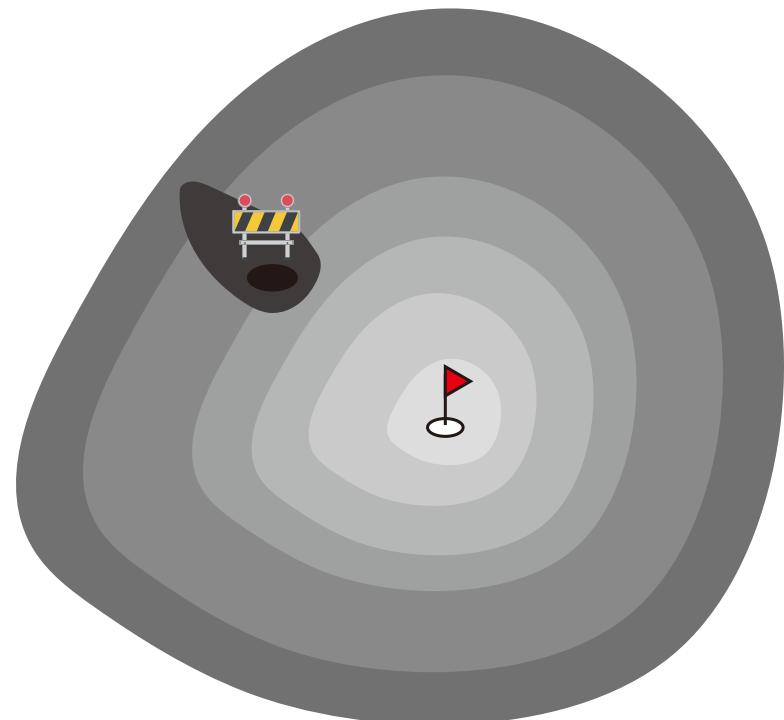
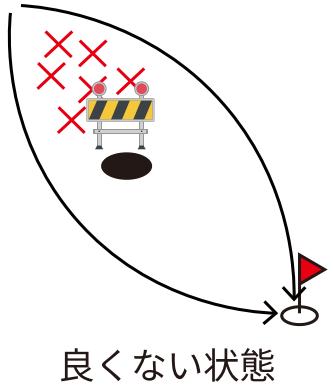
- 経路を再計算したら変わった/また計算したら戻った
 - 再計算が必要になる場合
 - 自己位置推定の結果がジャンプした
 - 突然の障害物の出現（再計算をやめられない）
 - もっと良いルートの探求
- チャタリングは危険
 - 下手にプログラムすると中途半端な動きに
 - 自動車: 中央分離帯で死ぬ（よくある事故）
 - なぜか自動車より遅い移動ロボットでも起こる
 - ROSのNavigation Stackのサンプル

(小手先で解決できるかもしれないが) 根本的な原因是？



原因

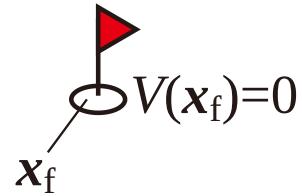
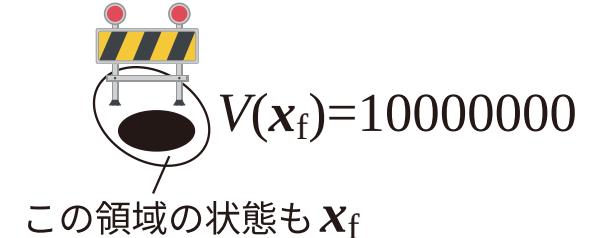
- 大きくチャタリングする=間に良くない状態が存在
 - 良くない: V の値が悪い=入ってはいけない
 - 探索のたびに↑を忘れる→「入っていいや」となる
- なんでそうなっちゃうのか: 線（経路）で考えているから
- どうするか: 面（正確には場）で考える
 - 事故を起こした状態もゴール（終端状態）だと考えて V^π や π を計算（次ページ）



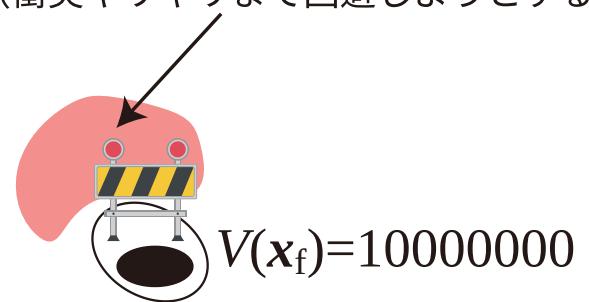
経路ではなく場で考える

「事故もゴール（終端状態）」

- 「事故を避ける方策」の計算方法
 - $V(x_f)$ を大きな値で固定
 - 状態遷移にペナルティーを与える方法もある（等価）
 - 線ではなく面（場）で計算
 - 危険な箇所近傍の V^π, π を全部解く
 - 状態遷移 $p(x'|u, x)$ はちゃんと確率的に扱う
 - ギリギリを攻めない
 - V^π がなめらかに
- 探索や多くの制御、最適化では障害物を「境界（制約）条件」として扱ってしまう（いいの？）
 - 「最適な経路」 ≠ 「事故を避ける方策」 \Rightarrow 脆い



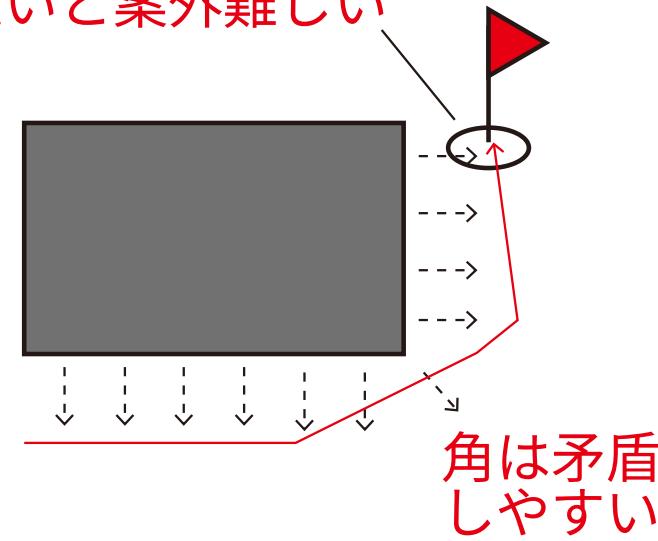
衝突全力回避方策ができる
(本来のゴールより近いので先にできる)
(衝突ギリギリまで回避しようとする)



探索結果+障害物回避の方策の組み合わせはダメ？

- OK。ただし互いに矛盾しないように（難しい）
 - 矛盾がない: 両方の方策から計算される V^π に停留点がない
 - 問題は1つなのに2つ別の方策→矛盾
- よくある矛盾
 - 角曲がりたい➡角の壁から遠ざかりたいで無限ループに
 - 同じ状態で別の方策が混在
 - 1994年の名古屋空港での中華航空機の事故
 - 着陸したいパイロットと地面から離れたいオートパイロット

ゴールと危険個所が
近いと案外難しい



角は矛盾
しやすい

凡例

- > 壁避け方策
- 計画された経路

方策の貼り合わせを避けて制御問題を解いてる例はあるのか？

- 実はある（一長一短という話は一旦棚上げして）
 - 強化学習と価値反復（動的計画法）
- 例: [白地図を持たせたロボットの価値反復でのナビゲーション](#)[上田RSJ2022]
 - 価値反復
 - 強化学習をモデルベースで解く方法（説明が逆ですが）
 - 確率的な状態遷移を扱える
 - 地図全体とロボットの周辺で全く同じ計算（方策が矛盾しない）
 - 地図全体: 大域計画用
 - ロボットの周辺: 障害物回避用
- [実機での障害物回避の例](#)[Ueda2023]

ここまでまとめ

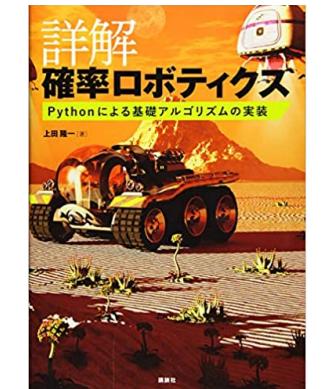
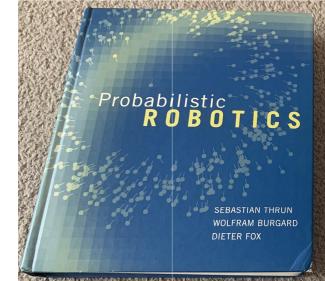
- 移動ロボットの経路計画を最適制御の観点から考えた
 - 探索や変分法を使う場合、次の問題への適切な対処が必要
 - チャタリングの問題
 - ゴールへの移動と危険回避行動が同時に扱えない問題
 - 確率的な状態遷移が考えられない問題
- さらに一言
 - 方策を貼り合わせる手法を論文に書くときは上記の問題にちゃんと触れてほしい（特に障害物回避の論文）
 - 「強化学習は安全性に疑問が・・・」という意見は本当か？
 - 学生のとき、MCLのときにもおんなじことを言われた

最適制御から様々な分野への話

- そもそもなんでこのような話をしているのか

確率ロボティクスの本質（個人的な解釈）

= 制御のパラダイムシフト



全部うしろは制御の話
(だってロボットなんだもん)

確率モデルで表現することで失うもの

状態方程式: $x_t \sim p(x|x_{t-1}, u_t)$

観測方程式: $z_t \sim p(z|x_t)$

- 一般化されすぎていて既存の制御理論から離れるか、再解釈が必要になる
 - リアプノフやポントリヤーギンの話、あらゆる微分方程式・変分法の話
 - 前提にこれのある人との人では話がすれ違う
 - 再解釈の例: グラフ上で微分を定義してなにかするなど
 - 学部のときの指導教官の故湯浅先生の研究
- 一定の割合の人の「制御とはこういうもの」から外れる
 - 交流がなくなりがち
 - 値値反復を使うと低次元の非線形制御問題をほぼ完璧に解いて遅いマイコンに方策を搭載可能なのでみんな使ったほうがいいと思って25年

確率モデルで表現することで得られるもの

状態方程式: $x_t \sim p(x|x_{t-1}, u_t)$

観測方程式: $z_t \sim p(z|x_t)$

- 状態 x や制御（行動） u の解釈が自由に
 - 確率モデルが決められればなんでもよい
 - ややこしい実世界を扱うには都合がよい
- 2000年頃にロボット周辺で取り入れていた分野
 - 機械学習（強化学習・ニューラルネットワーク）
 - 確率ロボティクス

確率モデル+ベルマン方程式の面白さ

- 「 $p(x|u, x')$ さえ決まつていれば状態 x はなんでもよい」
⇒ 状態空間に距離や内積の定義は不要（測度の定義が必要）
 - むしろ π^* 、 V^* が距離のようなものの定義に
 - なめらかで矛盾がない
 - いずれかをうまく近似表現するとロボットが動く
- この面白さが出ている典型的な例（議論の余地あり）
 - 強化学習で脚ロボットがよく歩くのは、特異点の問題から解放されているから？
- ニューラルネットの内部状態が微分（誤差逆伝播）で生成されるのがまた面白い

どこをモデル化するのか

- 内部状態の空間を滑らかにすることが正解かもしれない
 - 実世界を厳密に、滑らかにモデル化するのではなく
 - 学習の分野の人たちが前から言っていたことで特に新しい視点でもない
 - 方法論が確立して顕在化
 - あとは体を柔らかくして実世界のごつごつを吸收
- 確率ロボティクス側はこの考えが希薄だったかも
 - ロボットを動かそうとすると、結局、状態をベクトルで表して環境の地図を正確にしようとするに

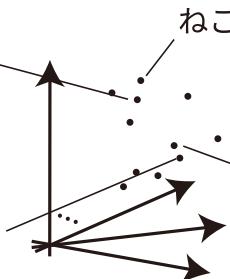
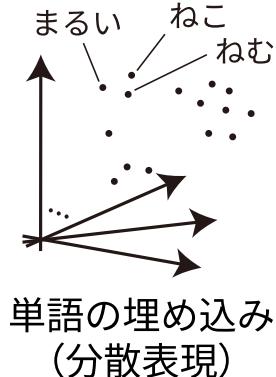


こんな雑然とした部屋(私の部屋)
はモデル化したくない

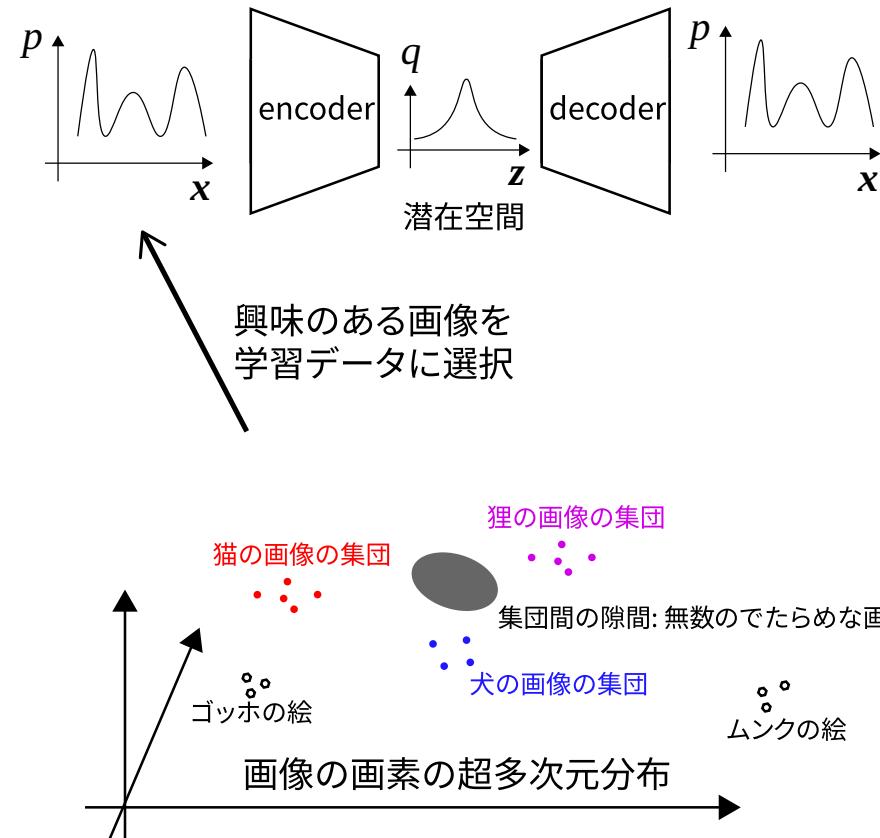
内部状態の構成（令和最新版）

超多次元空間への対応・次元の呪いの克服

- 変分オートエンコーダ（VAE）、拡散モデル、フローマッチング（FM）が作る潜在空間や分布（右図）
 - 複雑な分布を圧縮表現
 - Transformerが入出力する言語や映像のトークン（埋め込みのベクトル）の空間



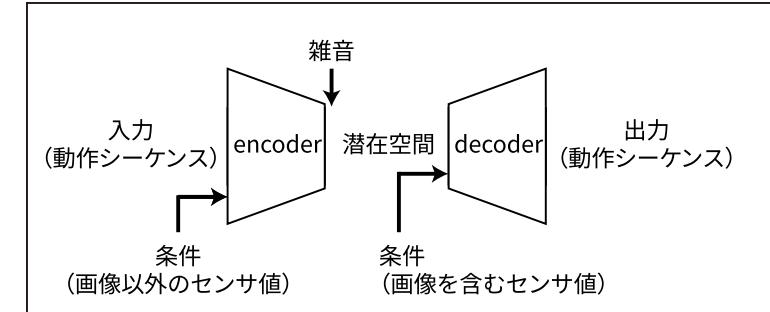
CLIP [Radford 2021]でできるvision-language空間



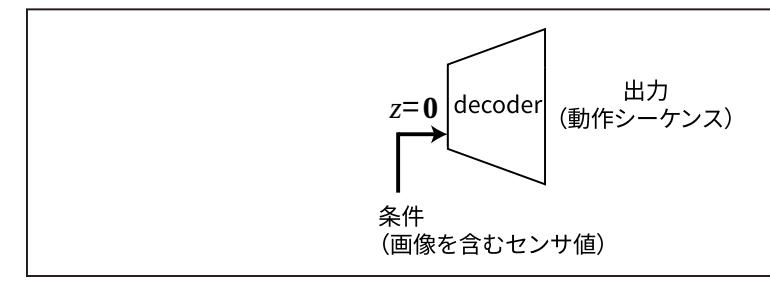
ロボット基盤モデルにおける空間表現

制御という観点から見ると面白い（ π 寄りだったり V 寄りだったり両方だったり）

- π 寄りの手法（ACTや π_0 ）
 - VAEや拡散モデル、FMでさまざまな動作シーケンスの分布を学習・生成
 - 私の博士論文のテーマも「方策の圧縮表現」で、自己位置推定でもSLAMでもなかつたりする
- V の推定寄りの手法（RT-1, 2など）
 - 指示と画像から V を良くする次の行動だけ出力
 - これが完璧だと（遅くとも）なんでもできる



ACT[Zhao2023]の学習



ACTの使用



どう動けば価値が上がる（タスク完了に近づく）だろうか？

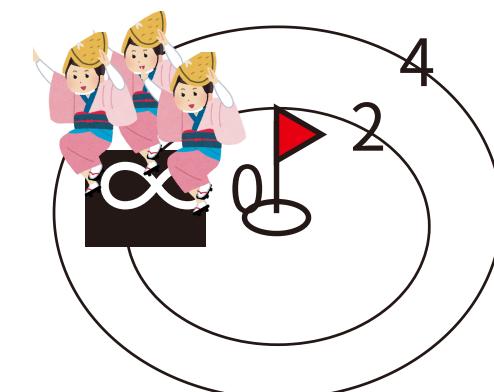
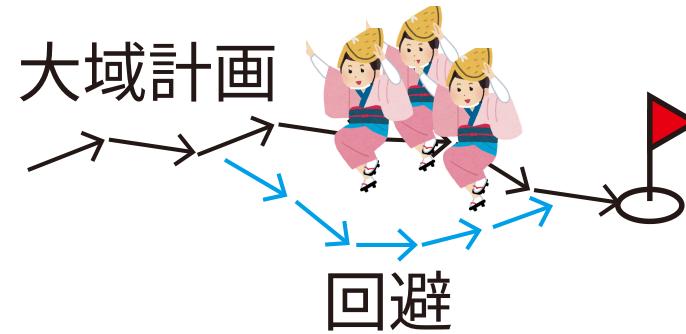
まとめ

- 制御という観点から見た経路計画については途中のまとめのとおり
- 制御を確率モデル（だけ）で考える方法
 - おそらく実世界の複雑性と格闘するには良い方法
 - VLAもそう解釈できる（ということに勉強して今更気づきました）
 - 解いている問題が同じなのだから当然ではある
 - 手法ではなく問題に着目する気持ちがあれば理解できる・話ができる
- その先（妄想）
 - もし、空間中のさまざまな場を $O(1)$ で正確に計算できる計算機ができたらどうなる？
 - V のような場が磁場のように一瞬で決まる計算機
 - 粘菌のコンピュータはこれに近い

以後ボツスライド

次の問題: 大域計画と局所計画との組み合わせ

- 問題: 突然の障害物の発見により π^* , V^* が変化
 - 今まで解いた π , V^π の見直しが必要
 - 完全に解き直してもよい
- よくやるアプローチ: 大域計画問題に局所計画問題を足す
 - 問題の2階建て
 - 何か問題でも?
 - だいたいの問題はこれまで話で説明できるが、もう1つ問題が増える
- 増える問題: 解くべき最適制御問題は1つなのにわざわざ別の問題を解き、そして解が矛盾



例: 大域計画器で経路生成、局所計画器で細かく制御する場合

(ROSのデファクトの方法)

1. 大域計画器の示すパスを局所計画器が履行しようとしたら無理だった（大域計画器が大雑把にしか環境を見ていない）
 2. 局所計画器が大域計画器に無理だと伝える
 3. 大域計画器は無理な理由が理解できないから再計画できない
- 上記は論外として、ナビゲーションと衝突回避のモード切替をするシステムでも、互いに矛盾させないことが重要

例: ウェイポイントナビゲーション

- ロボットの動作がぎこちなくなりがち
 - 自己位置推定がジャンプしてわざわざウェイポイントを踏みなおしに戻る
 - 実装が下手だと急旋回して自己位置推定が破綻
 - 経路がずれて途中飛ばしたほうがよさそうだが一応ウェイポイントを踏みに行く
 - これが実現できるくらいの制御器を持っていればウェイポイントはいらない

根本的な問題

- モード切り替えをするシステムでは、状態空間の次元が増える
 - $\mathbf{x}_{\text{aug}} = (\mathbf{x}, \text{mode})$
 - 「mode」の例
 - 今はナビゲーションしているのか衝突回避しているのか
 - どのウェイポイントを目指しているのか
- 分けて簡単にしたつもりが逆に問題が難しく
 - $p(\mathbf{x}|\mathbf{x}', \mathbf{u})$ が $p(\mathbf{x}|\mathbf{x}', \mathbf{u}, \text{mode})$ に

- 「いまどの問題をロボットが解いているか」という状態変数が1つ増える
 - 気づきにくいが
- 軽く考えてはいけない
 - 航空機事故もこれで結構起こる（オートパイロットとマニュアルの切り替え）

探索と制御（本発表の結論？）

- 各地点のコストの計算: 制御問題
- 最適制御問題
 - 状態空間と状態を考える

LLMでの制御

- Robotics Transformer 2
 - ひとつずつ動きを出力
- とりあえず少しでも価値が上がる行動をしていれば最終的にタスクは達成できる

その先の話

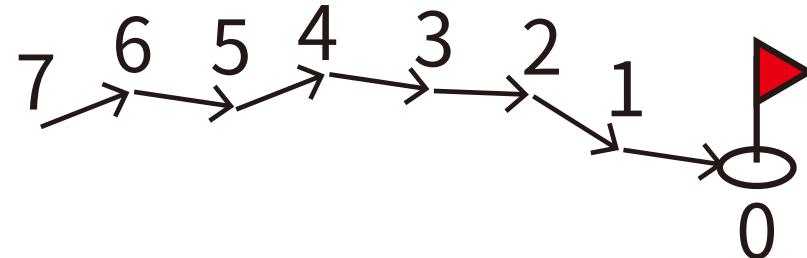
- 次元の呪いが克服されたら？
- 現在の計算機と性能のオーダーが違う計算機の登場

- コスト: 状態 (位置) x に対して $V(x)$

最適制御問題の解の性質（どう解くかという話とは別）

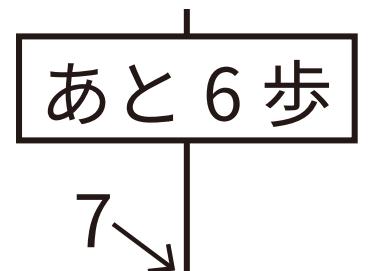
最大原理で解くような問題でもうまく x を設計すればこうなる

- ある制御則の解（方策） $u = \pi(x)$ に対し、ひとつひとつの状態から終端状態までのコストの期待値が計算できる
 - 実数を返す関数 $V^\pi(x)$: 状態価値関数（値関数）
- 大域計画の問題で考えると、別に難しい話ではない
 - ある場所 x にいるとき、行き方 $\pi(x)$ が決まっていれば、目的地までの時間の期待値 $V^\pi(x)$ が見積もれる



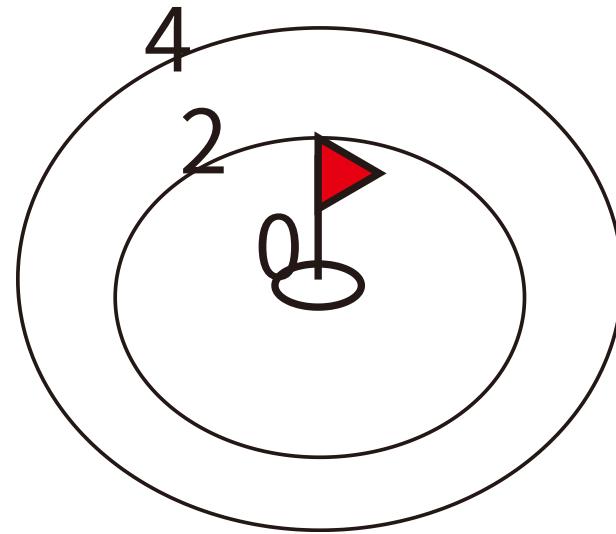
V の性質（状態遷移が決定論的な場合）

- x の価値は遷移先 x' の価値に遷移したときのコスト ℓ を足したもの
 - $V^\pi(x) = V^\pi(x') + \ell(x, u, x')$
 - ここで $u = \pi(x)$ 、 $x' = f(x, u)$
 - 例: ゴールまで7歩のところから1歩歩いたら、次の状態はゴールまで6歩に
- 補足: ℓ は一般化できる
 - 「ここは悪路だからコストを倍に」ということをしても破綻しない（ので以後、步数として話を単純化）



終端状態 $x_f \in X_f$ の扱い

- それ以上状態遷移しない状態
- $V^\pi(x_f) = 0$ など、価値を固定しておく
- 他の状態の V^π は $V^\pi(x_f)$ にしたがって決まる
 - $V^\pi(x_f)$ を底とするポテンシャル場に



方策の改善と最適性

- もっと良い行き方 $\pi'(\mathbf{x})$ があれば、時間の期待値が $V^{\pi'}(\mathbf{x})$ に短縮される
 - 方策を改善していくと収束
 - 収束した V : 最適状態価値関数 V^*
 - $V^*(\mathbf{x}) = \min_{\mathbf{u}} \{V^*(\mathbf{x}') + \ell(\mathbf{x}, \mathbf{u}, \mathbf{x}')\}$
 - まったく停留点のないポテンシャル関数に
 - V^* を与える方策: 最適方策 π^*
 - $\pi^*(\mathbf{x}) = \operatorname{argmin}_{\mathbf{u}} \{V^*(\mathbf{x}') + \ell(\mathbf{x}, \mathbf{u}, \mathbf{x}')\}$

状態遷移が確率的な場合への拡張

- $V^\pi(\mathbf{x}) = \langle V^\pi(\mathbf{x}') + \ell(\mathbf{x}, \mathbf{u}, \mathbf{x}') \rangle_{p(\mathbf{x}|\mathbf{u}, \mathbf{x}')}$
 - $\langle f(\mathbf{x}) \rangle_{p(\mathbf{x})}$: 分布 p のときの f の期待値
- 最適なとき (ベルマン方程式)
 - $V^*(\mathbf{x}) = \min_{\mathbf{u}} \langle V^*(\mathbf{x}') + \ell(\mathbf{x}, \mathbf{u}, \mathbf{x}') \rangle_{p(\mathbf{x}|\mathbf{u}, \mathbf{x}')}}$
 - $\pi^*(\mathbf{x}) = \operatorname{argmin}_{\mathbf{u}} \langle V^*(\mathbf{x}') + \ell(\mathbf{x}, \mathbf{u}, \mathbf{x}') \rangle_{p(\mathbf{x}|\mathbf{u}, \mathbf{x}')}}$
- この定式化のおもしろいところ
 - \mathbf{x} や \mathbf{u} はベクトルで表記しているけどその必要はない
 - $p(\mathbf{x}|\mathbf{u}, \mathbf{x}')$ さえ厳密に決まっていればよい
 - 距離の定義が空間になくてもよい
 - むしろ解の V^* が距離のようなものの定義になっている

危険回避は確率的な状態遷移から自然に求めたほうがよい

計算量が大きいことはさておき

- 確率的な状態遷移を使う
 - 自動車やロボットの動きには再現性がない
 - 路面の状況、風、内部のガタなど
 - 狹い通路の端を走るとぶつかる可能性を表現できる（右図上）
 - 通路中央に V^* の谷が計算される（右図下） → ロボットが中央を自然に走る
 - 谷が分岐していても対称性が破れる

