

二次元マーカを用いるロボットマニピュレーションの 位置誤差に関する研究^{*}, 1), 2)

福本 靖彦, 神内 杜夫, 坂東 慎之介, 竹中 慎, 小林 宏明, 長谷見 健太郎

ロボットに取り付けたカメラで作業環境に貼り付けた二次元マーカの位置を測定し、その位置を基に作業を行うと、ロボットの据え付け位置が多少変わっても問題なく作業を行うことができる。本研究ではこのような方法で作業を行ったときのロボットの手先の位置誤差について考え、位置誤差を正規分布としてモデル化する方法を提案するとともに、位置誤差が最小となるようなマーカの貼り付け位置と姿勢を求める最適化問題を定式化し、実験によりその有効性を確認した。

1 緒言

本県では、県政の基本指針として『「人生 100 年時代のフロンティア県・香川」実現計画³⁾』を策定しているが、その中で人口が加速度的に減少していることが課題として挙げられている。本県の令和 6 年 4 月 1 日現在の人口は約 92 万人であるが⁴⁾、同計画では令和 42 年以降に人口を約 77 万人で安定させることを目標としている。つまり今後、同計画により人口減少にうまく歯止めがかかったとしても、人口は現在に比べ、約 16% の減少が見込まれている。このことに伴い、労働人口も同様に減少することから、これまで以上にロボット等を用いた自動化が求められる。

さて昨今、協働ロボットと呼ばれるアーム型ロボットの活用が急速に拡大している。協働ロボットは安全柵を設けなくても良く、人と同じ環境で作業ができるロボットである。また、ティーチングのしやすさや、ハンド等の周辺機器との連携のしやすさが飛躍的に向上していることも注目すべき特徴である⁵⁾。これらの特徴によりロボットをより手軽に使うことができるようになってきており、必要な時に必要な場所にロボットを設置する、あるいはロボットを台車等に乗せて移動させて使うという使い方が現実的となってきている。ロボットが移動するようになると、ロボットの据え付け位置の再現性が問題となる。一度据え付けたロボットを別の場所へ移設して使用したのち、また元の場所に戻した場合に、据え付け位置の再現性が悪いと一からティーチングをやり直すといった作業量の増大を招く恐れがある。

ロボットの据え付け位置の再現性が十分得られなくても元のプログラムをそのまま使えるようにするための方法として、ロボットの手先に取り付けたカメラで、作業環境に貼った二次元マーカの位置と姿勢を計測し、マーカの位置を基準にして作業を行う(図 1)ソリューション^{6~14)}が市販されている。このような方法で作業を行うことを本研究ではマーカベースマニピュレーションと呼ぶこととする。

ここで通常のティーチング・プレイバック動作におい

ては、ロボットが作業を行う際の手先の位置誤差はロボットの繰り返し精度の範囲に収まると考えられる。しかし、マーカベースマニピュレーションを行う際には、これに加えてマーカの位置の計測誤差や、マーカと実際に作業を行う位置との相対位置も影響を与えるため、作業を行う手先の位置誤差の範囲を示すことは容易ではない。実際、マーカベースマニピュレーションのソリューションを提供する製品においても、特定のマーカベースマニピュレーションを行った際の精度を参考値として示すにとどまったり⁸⁾、マーカ計測の精度しか示されていない¹³⁾。よって、ユーザーが作業位置の誤差の範囲を把握することは難しい。

そこで本研究では、マーカベースマニピュレーションを行うときのロボットの手先の位置の誤差を評価する方法について検討した。また、それを設計に応用し、推定誤差を最小化するマーカの貼り付け位置の最適化手法の検討を行った。

2 節では本稿で用いる文字等の定義を行う。3 節ではマーカベースマニピュレーションの仕組みを説明する。4 節では計測誤差を正規分布でモデル化して作業位置の推定誤差の確率分布を計算し、その確率分布により推定誤差を評価する方法を示す。5 節では推定誤差を最小化する最適化問題を定式化する。6 節では実験結果を示す。7 節でまとめを述べる。

2 定義

本研究では、位置と姿勢をそれぞれ $p \in \mathbb{R}^3$, $a \in \mathbb{R}^3$ で

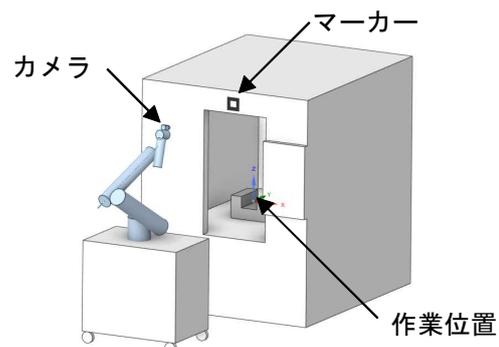


図 1 ロボットによる工作機械上の二次元マーカ位置計測およびワーク設置作業 (イメージ図)

* 本稿はロボティクス・メカトロニクス講演会 2024 にて発表した内容¹⁾と、第 25 回計測自動制御学会システムインテグレーション部門講演会にて発表した内容²⁾を統合して再編集したものである。

表す。なお、姿勢は回転ベクトル表記であり、 \mathbf{a} の方向が回転軸を表し、 \mathbf{a} の大きさが回転量を表す。回転量の単位は rad とする。また、本稿では、位置と姿勢を合わせた六次元ベクトル $\mathbf{r} \in \mathbb{R}^6$ を単純に位置と呼ぶ場合もある。また、 $\mathbf{R} \in \mathbb{R}^{3 \times 3}$ を \mathbf{a} と等価な姿勢を表す回転行列とし、 $\mathbf{T} \in \mathbb{R}^{4 \times 4}$ を \mathbf{r} と等価な位置を表す同次変換行列とする。

また本稿では、以下の五つの座標系を定義する。一つめはロボットの根本等に固定したロボット座標系 (r)、二つめはロボットの手先に固定したハンド座標系 (h)、三つめはカメラに固定したカメラ座標系 (c)、四つめはマーカーに固定したマーカー座標系 (m)、五つめは作業位置に固定した作業座標系 (t) である。

位置を示す際、本稿では、 $\mathbf{p}, \mathbf{a}, \mathbf{R}, \mathbf{r}, \mathbf{T}$ の左上と右下に上述の五つの座標系 (r, h, c, m, t) の添え字を付して示すこととする。例えば ${}^r\mathbf{T}_h \in \mathbb{R}^6$ はロボット座標系で表示したハンドの位置を意味する (図2)。また、m, t については、それぞれ計測値 (カメラで計測した値) と、推定値 (マーカー位置の計測値に基づく計算値) を意味する場合にはそれぞれ mm, te と記す。

3 マーカーベースマニピュレーション

マーカーベースマニピュレーションでは、ロボットに取り付けたカメラで作業環境に貼り付けた二次元マーカーの位置と姿勢が計測され、その値を基に様々な作業が行われる。カメラで認識がしやすい外観を有するマーカーを用いることにより、作業位置にある把持対象物等を直接カメラで認識するよりも画像認識が容易になることや、対象物に応じた形状モデル等が不要になるといったメリットがある。

マーカーベースマニピュレーションは既に様々なロボットで実用化されており、Techman/オムロン⁶⁾、カワダロボティクスの協働ロボットの標準的な機能となっている⁷⁾ ほか、ファナック、デンソーウェーブ、三菱電機はそれぞれのロボットのオプション機能として提供している^{8~10)}。なお、安川電機の協働ロボットはカメラではなく、レーザーセンサを用いて位置補正を行う機能を標準搭載している¹¹⁾。また、Robotiq は Universal Robots 製品向けにマーカーベースマニピュレーションを行うためのソリューションを提供している¹²⁾。OnRobot は様々な協働ロボット向けに同様のソリューションを提供している¹³⁾。

マーカーベースマニピュレーションでは、通常のティーチング・プレイバックのように単純に ${}^r\mathbf{T}_t$ を登録するのではなく、 ${}^r\mathbf{T}_h, {}^h\mathbf{T}_c, {}^c\mathbf{T}_m, {}^m\mathbf{T}_t$ とたどって作業位置を得る (図2)。ただ ${}^m\mathbf{T}_t$ の値を覚えていても、ロボットの据え付け位置が変わると意味をなさなくなってしまうが、 ${}^r\mathbf{T}_h, {}^h\mathbf{T}_c, {}^c\mathbf{T}_m, {}^m\mathbf{T}_t$ とたどる場合は、カメラでマーカーの位置 ${}^c\mathbf{T}_m$ を計測することで、据え付け位置が変わった時の影響に対応できる。具体的な処理の流れを以下で説明する。

まず準備として、マーカーとキャリブレーションボードを用意する。これらはチェスボードと呼ばれるパターンが印刷された板等が用いられる。チェスボードの格子模様の交点間の位置関係は既知であるとする。マーカーとキャリブレーションボードの例を図3に示す。なお、チェスボード以外にサークルグリッド⁸⁾、QR コード⁹⁾、

ArUco マーカー¹⁴⁾ 等を用いる場合がある。

次に ${}^c\mathbf{T}_m$ を求めるために必要なカメラキャリブレーションと、 ${}^h\mathbf{T}_c$ を求めるハンドアイキャリブレーションを行う。これらのキャリブレーションは、ロボットを様々な位置に動かして様々な位置からキャリブレーションボードを撮影した画像やそのときのロボットの先端座標値を取得し、必要な計算を行うことで実現する。この計算については、市販されているマーカーベースマニピュレーションのソリューションを利用する場合には計算プログラムが付属しているが、そのようなソリューションを用いない場合は画像処理ライブラリ OpenCV の `caribrateCamera` 関数や `caribrateHandEye` 関数を用いると容易に実装できる。

最後に ${}^m\mathbf{T}_t$ を登録する。 ${}^m\mathbf{T}_t$ の値は図面等から値が分かっている場合はそれを用いることができる。そうでない場合は、

$${}^m\mathbf{T}_t \cong {}^r\mathbf{T}_{mm}^{-1} {}^r\mathbf{T}_t \quad (1)$$

等で代用する。ここで ${}^r\mathbf{T}_t$ の値は、マニュアル動作でロボットの先端を作業位置に移動させることで得ることができる。 ${}^r\mathbf{T}_{mm}$ は一度マーカーの位置を計測すれば

$${}^r\mathbf{T}_{mm} = {}^r\mathbf{T}_h {}^h\mathbf{T}_c {}^c\mathbf{T}_m \quad (2)$$

として計算することができる。上式の ${}^r\mathbf{T}_h$ はマーカーを撮影するときのロボットの先端位置である。 ${}^c\mathbf{T}_m$ は PnP (Perspective-n-Point) 問題を解くことで得られる。なお、市販のソリューションを用いずにマーカーベースマニピュレーションを構築する場合は OpenCV の `solvePnP` 関数を用いると容易に PnP 問題の解を得ることができる。

マーカーベースマニピュレーションの運用時は、まずマーカーの位置と姿勢を計測して式(2)によりマーカーの位置 ${}^r\mathbf{T}_{mm}$ を得る。そして ${}^r\mathbf{T}_{mm}$ と、あらかじめ登録しておいたマーカーと作業位置の相対関係 ${}^m\mathbf{T}_t$ を用いて

$${}^r\mathbf{T}_{te} = {}^r\mathbf{T}_{mm} {}^m\mathbf{T}_t \quad (3)$$

により作業位置を計算すれば良い。

ここで、マーカーベースマニピュレーションの作業

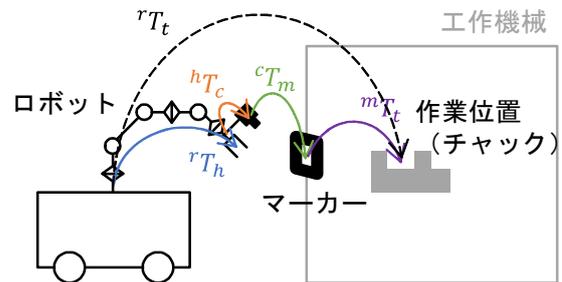


図2 位置関係

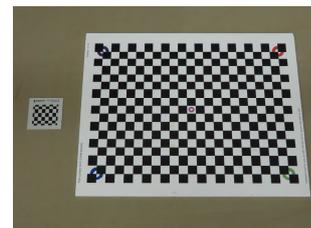


図3 マーカー (左) とキャリブレーションボード (右) の例 (WristCamera (Robotiq) 製)

位置の推定誤差はマーカークの計測誤差だけでは決まらず、マーカークと作業位置の相対位置にも依存する。このことは、例えば図4のようにマーカークの姿勢の計測値に誤差があると、マーカークから遠いほど作業位置の並進成分の推定誤差が大きくなることなどから確認することができる。

4 確率分布による作業位置の推定誤差の表現

マーカークの位置を計測すると、その計測値には計測誤差が含まれる。計測誤差は作業位置の推定値に影響を与える。ロボットシステムの設計を行うにあたり推定誤差の程度を知る必要が生じる場合があるが、そのためにはまず六次元の誤差の分布範囲をどのように示すかということから考える必要がある。これに対して本研究では、誤差を六次元の確率分布で考えることとした。以下では、六次元の計測誤差を六次元正規分布でモデル化して作業位置の推定誤差の確率分布を求めると、マーカークの姿勢の計測誤差が十分小さいときには推定誤差の確率分布をある正規分布で近似できることを説明する。また、六次元正規分布で表現することの有用性についても併せて説明する。

まずマーカークの計測誤差を、 ${}^mT_{mm}$ として表すと、作業位置の推定誤差は

$${}^t r_{te} \equiv {}^m T_t^{-1} {}^m T_{mm} {}^m T_t \quad (4)$$

として計算することができる。特に $\|{}^m a_{mm}\| \ll 1$ 、つまり姿勢の計測誤差が 1 rad ($=57.3^\circ$) より十分小さい場合に

$${}^t r_{te} \cong A {}^m r_{mm} \quad (5)$$

$$A \equiv \begin{bmatrix} {}^m R_t^T & -{}^m R_t^T [{}^m p_t \times] \\ 0 & {}^m R_t^T \end{bmatrix} \quad (6)$$

と書くことができ¹⁾、 ${}^t r_{te}$ と ${}^m r_{mm}$ が線形であると見なすことができる。なお、 $[{}^m p_t \times]$ は三次元ベクトルに ${}^m p_t$ を左から掛ける外積計算と等価となる歪対称行列である。

よって、マーカークの位置の計測誤差を六次元の正規分布 $N(0, \Sigma_m)$ でモデル化すると、マーカークの姿勢を十分精度良く計測できる場合には作業位置の推定誤差の確率分布を正規分布 $N(0, \Sigma_t)$ で近似することができる。なお、 $\Sigma_m \in \mathbb{R}^{6 \times 6}$ は ${}^m r_{mm}$ の分散共分散行列であり、

$$\Sigma_t = A \Sigma_m A^T, \quad (7)$$

である。

したがって、あらかじめ複数回計測を行って計測誤差の確率分布を得ておけば、作業位置の推定誤差の確率分布を近似した正規分布 $N(0, \Sigma_t)$ を求めることができる。

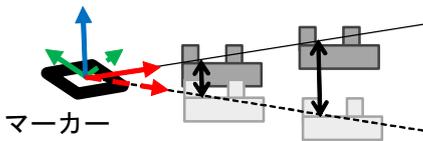


図4 マーカークの計測誤差と作業位置推定への影響

実線の赤、緑、青の矢印はマーカークの位置の真値を表し、点線の矢印は計測値を表す。マーカークの姿勢の計測誤差は、マーカークと作業位置の距離が遠いほど大きな作業位置推定誤差を生む。

$\Sigma_t \in \mathbb{R}^{6 \times 6}$ が分かれば、様々な誤差情報を容易に提示することができ、非常に有用である。例えば推定誤差の99.7%領域 (3σ に相当、 σ :標準偏差)の六次元楕円体は

$${}^t r_{te}^T \Sigma_t^{-1} {}^t r_{te} \leq c^2 \quad (8)$$

となる。なお、 c^2 は自由度6のカイ二乗分布の下側99.7%点の値であり、書物等に示されたカイ二乗分布表を参照するか、Pythonの科学演算ライブラリのSciPyの`scipy.stats`モジュールの`chi2.ppf`関数等を用いて、値を得ることができる。例えば、99.7%領域の並進の最大誤差は

$$\max \|{}^t p_{te}\| = \sqrt{c^2 \max_{i=1,2,3} (\lambda_i)} \quad (9)$$

で計算することができる。ただし、 λ_i は Σ_t の左上の 3×3 のブロック行列の固有値である。また、並進のXY成分の99.7%領域は $N(0, \Sigma_t)$ を周辺化して並進のXYの二次元の正規分布を計算すれば容易に得られ、図示することができる。

5 作業位置の推定誤差の最小化

前節では推定誤差の評価方法について述べた。本節ではこれを設計に応用する。具体的には、推定誤差が最小となるようなマーカークの貼り付け位置及び姿勢を計算する最適化問題を定式化する。

ここで、推定誤差は六次元正規分布に従う確率変数であるので、何を最小化するかを具体化する必要がある。これに対して本研究では推定誤差の並進成分の99.7% (任意の確率)領域の最大値、つまり99.7%誤差楕円体の最大半径を最小化することとした。これは、推定誤差の並進成分のいわゆる 3σ の値を最小化することに相当する。

以上を踏まえ、解くべき最適化問題を以下のように定式化する。

目的関数:

$$f({}^t r_m) \quad (10)$$

制約条件:

$$g({}^t r_m) \geq 0 \quad (11)$$

ここで、

$$f({}^t r_m) = \max_{i=1,2,3} (\lambda_i) \quad (12)$$

であり、 $\lambda_1, \dots, \lambda_3$ は ${}^t r_m$ に依存する値である。 $g({}^t r_m)$ はマーカークを ${}^t r_m$ に貼りつけることが可能な場合に非負の値を返す関数とする。物理的に貼り付け可能なスペースがあるかどうかに加え、ロボットと作業環境が干渉しないといった制約を考える場合は、それもここに含めるものとする。

なお実際には誤差楕円体の半径は $\sqrt{c^2 \lambda_i}$ であるが、 c^2 は定数、 Σ_t は正定値、平方根は単調な操作であるので、単純に λ_i の最大値を最小化すれば誤差楕円体の最大径を最小化することと等価である。以上の理由から、目的関数を式(12)のように定義した。

また、推定誤差の回転成分の大きさはマーカークと作業位置の相対位置に依存しないので²⁾、最適化の指標としては含めなかった。

6 実験

本節では図5に示す工作機械の前面にマーカーを貼り付け、工作機械内部のチャックの推定位置のばらつきについて考える例題を扱うこととした。この例題において、6. 1項では図6の位置にマーカーを貼り付けたときのチャックの位置の推定誤差を評価する。4節で示した方法では計測誤差を正規分布でモデル化し、更に計測誤差の確率分布から推定誤差の確率分布を近似計算していることから、まずは、これらのモデルと実際のデータの差異を確認することとした。そして6. 2項では、工作機械の前面に任意の姿勢でマーカーを貼り付け可能であると仮定して、5節で示したマーカーの貼り付け位置最適化を行い、推定誤差の低減効果について検証することとした。

6. 1 推定誤差の評価実験

本項では図6のように、マーカー中心の位置を工作機械前面の作業位置に最も近い位置(図5において「最近点」として示した位置)とし、マーカー表面の法線方向が工作機械の外部を向く(- Y_i 方向を向く)状態を想定した検討を行う。まず、計測誤差の測定実験を行い、続いてその値を基にして作業位置の推定誤差の評価を行う。

計測誤差の測定実験の実験系の外観を図7(a)に示す。ロボットはMOTOMAN-SDA10F(安川電機)の左腕を用いた。カメラはacA2500-14uc(Basler)、レンズはC125-0618-5M-P(Basler)を用いた。カメラの位置は、マーカーの正面からマーカー座標系のX軸周りに約 30° 傾けた方向でマーカー中心から約200mm離れた位置を基準位置とした。これらのカメラとレンズの選定と撮影位置についてはRobotiqの製品であるWrist Cameraの仕様を参考にした。二次元マーカーはRobotiqのカメラ用マーカーであるACC-TAG-1BAG5を用いた。カメラキャリブレーションとハンドアイキャリブレーションはあらかじめ実施した。

このセッティングでのマーカーの撮影例を図7(b)に示す。このとき、事前実験としてカメラの位置を上述の基準位置から動かさずに100回の撮影を行ってカメラ座標系で見たマーカー位置の計測値を調べると、その値のばらつきは十分小さいことが確認されたので、100回撮影したうちの初回の値である

$${}^c\mathbf{r}_m = [-4.2, 3.6, 207.5, 2.60, -0.04, -0.02]^T \quad (13)$$
をマーカー位置の真値として用いた。ここで、並進成分の単位はmmであり、姿勢成分の単位はradである。なおPnP問題はOpenCVのsolvePnP関数により解いた。

計測誤差データを取得するためには、台車等でロボット初期据え付け位置とは別の場所へ移動させ、再度元の位置に目視等で設置し直す操作を繰り返すと良いが、この方法は作業が煩雑となる。そこで本実験では、ロボット本体を動かさず代わりに撮影時のロボットの手先位置をランダムに変化させることとした。変位量はガウシアンノイズで与えることとし、並進成分には平均0mmで標準偏差が10mmのノイズを重畳し、回転成分には平均0radで標準偏差が $\pi/180$ rad($=1^\circ$)のノイズを重畳した。この方法で1000回マーカーの位置と姿勢を測定した。

実験により得られた1000回の計測データと式(13)のマーカー位置と姿勢の真値の値を利用して、マーカーの位置と姿勢の計測誤差データを計算し、更にその計測誤差データから Σ_m を計算した。計算された計測誤

差データと正規分布 $N(0, \Sigma_m)$ を周辺化して二次元化した正規分布における99.7%楕円を図8に示す。

続いて、式(4)を用いて得られた1000点の推定誤差データを計算するとともに、 Σ_m を用いて推定誤差の確率分布を近似計算した。計算で得られた1000点の推定誤差データと、近似分布により得られた99.7%楕円体を周辺化した楕円を図9に示す。

図8を見ると、計測誤差は完全には正規分布にしたがっていないと考えられるものの、描画した99.7%楕円は分布の特徴をよく捉えていることが分かる。また図9を見ると、近似分布の99.7%楕円もまた推定誤差の傾向をよく捉えていることが分かる。なお、どちらも誤差の値が大きい異常値のようなデータが想定より多く存在している。図8と図9では1000点のデータがプロットされているので99.7%楕円の外にはみ出るデータは3点程度であると期待されるが、いずれも3点を超えるデータが99.7%楕円外にはみ出ている。した

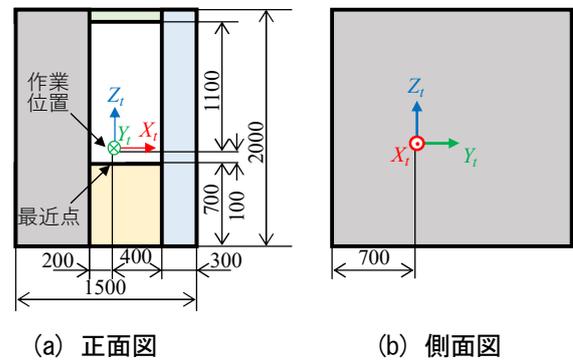


図5 例題で想定する工作機械の形状

正面図の中で色を塗っていない箇所は扉部であり、マーカーを貼れないものとする。単位: mm。

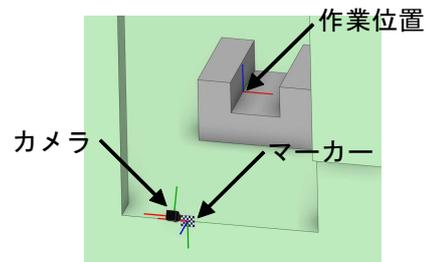


図6 マーカーの貼り付け位置

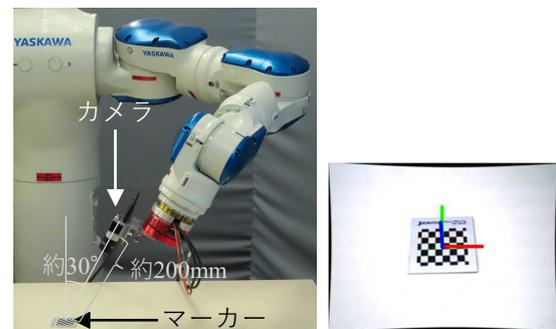


図7 測定誤差計測実験の外観

マーカー座標系の矢印はX軸(赤), Y軸(緑), Z軸(青)に対応。

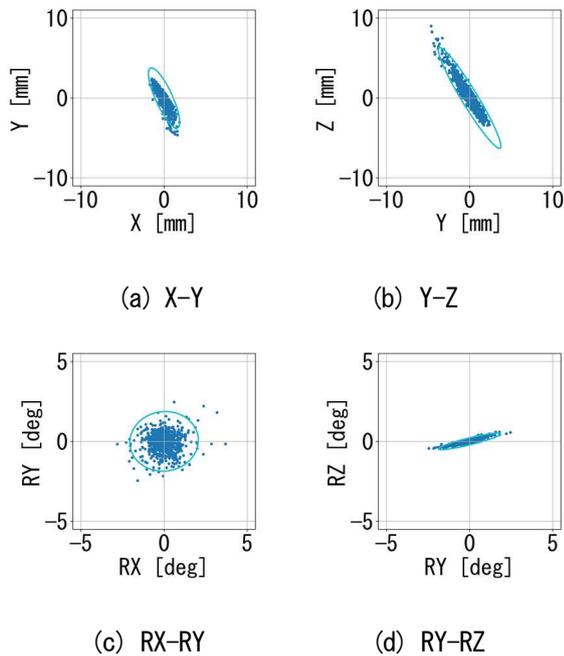


図8 計測誤差データ

ドットは計測されたデータ，楕円は計測データを正規分布にフィッティングしたときの99.7%誤差楕円。

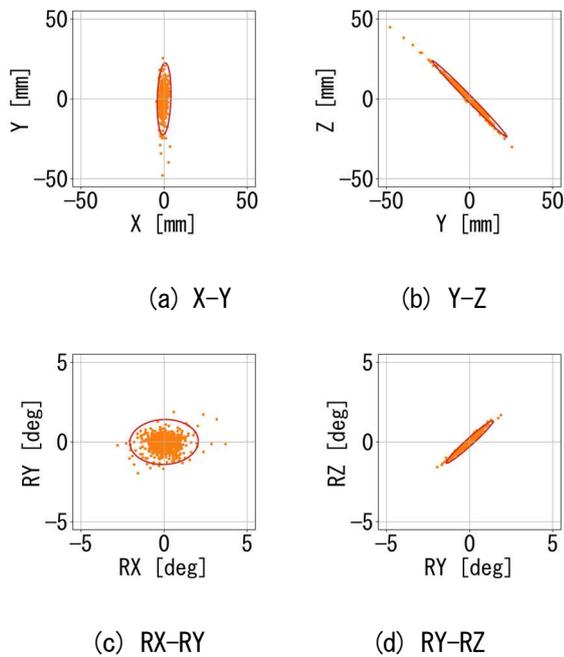


図9 推定誤差データ

ドットは計測誤差データから計算された推定誤差データ，楕円は推定誤差の近似分布から得られる99.7%誤差楕円。

がって，誤差の最大値を把握したい場合には計算処理方法について注意が必要である。

6. 2 マーカー位置の最適化実験

本項では工作機械内部のチャックの位置の推定誤差が最小となるようなマーカーの貼り付け位置と姿勢を決める設計問題を考え，最適化の効果を検証する。

マーカーを貼り付け可能な位置と姿勢について，位置は工作機械の前面の扉部以外の任意の位置とし，姿勢は特に条件を設けず任意の姿勢とした。ただし，カメラの位置 ${}^t\mathbf{r}_c$ が工作機械前面よりも内側になる位置は不適とした。並進位置に関する制約から ${}^t\mathbf{r}_m$ の並進のY成分は-700 mmに決まるので，これを除く5変数を最適化した。また，前面の扉部を除く領域を位置の許容解とする制約は，図5(a)のように工作機械前面を4分割し，位置をそれぞれの領域内に限定して最適化を行う四つの小問題を解き，それらの解の中から最適解の一つ選ぶという形で実装した。最適化計算にはSciPyライブラリのminimize関数を利用し，最適化アルゴリズムはSLSQPを用いた。計測誤差データは前項で得た1000データを4:1に分割し，800データを最適化計算に利用する訓練データとし，残り200データを最適化結果の評価に用いるテストデータとした。実作業の現場での利用が最も多いと思われる前項の貼り付け方法

(以下、「最近正面貼り付け」と呼ぶ)を最適化結果との比較対象とした。

図10に最適化により得られた貼り付け位置を示す。また図11に，最適化した場合と最近正面貼り付けの場合について，それぞれ計測誤差データのテストデータから推定誤差を計算した結果を箱ひげ図で示す。

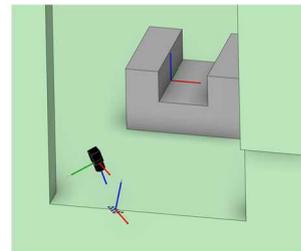


図10 最適な貼り付け位置

図中の直線は，カメラ座標系，マーカー座標系，作業座標系のX軸(赤)，Y軸(緑)，Z軸(青)に対応。

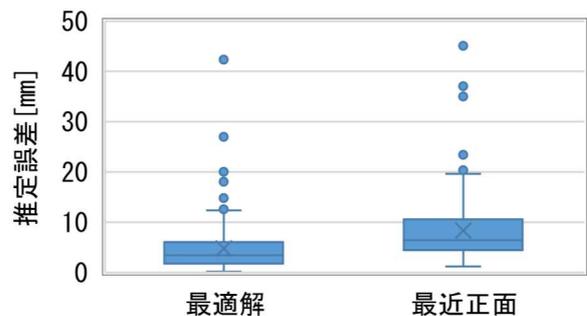


図11 最適化の効果

図10に示す通り、本実験で得られた最適解は、貼り付け位置は最近点であるが、貼り付け姿勢は上向きであった。つまり、最近正面貼り付けと位置は同じであるが、向きが大きく異なる結果となった。

図11を見ると、最適化を行うことで例えば最大値が6.1%、75パーセンタイル値が42.8%低減されている。75パーセンタイル値の低減幅は顕著であるが、最大値における低減効果は比較的小さい。前項では、計測誤差を正規分布に当てはめて考えると、稀に生じる大きな誤差を与えるデータが表現しきれないことが分かったが、これにより75パーセンタイル値では最適化効果が大きい、最大値では最適化効果が比較的小さくなってしまったものと考えられる。

本実験では、最適なマーカの貼り付け位置は、作業位置から最も近い位置で、姿勢は正面にそのまま貼り付けるのとは異なる姿勢であった。姿勢を変えることで、推定誤差の最大値は6.1%低減され、75パーセンタイル値は42.8%低減された。

7 結言

本研究では、環境に貼り付けた二次元マーカの位置と姿勢を、ロボットに取り付けたカメラで計測し、その値を基に作業を行う場合のロボットの手先の位置と姿勢の精度について検討した。まず、繰り返しマーカの位置と姿勢の計測を行って計測誤差データを取得し、その値を用いて作業位置の推定誤差の確率分布の近似値を求める方法を示した。これにより、作業位置の六次元の推定誤差の傾向を容易に把握することができることを確認した。また、この評価方法を設計に応用し、マーカを貼り付ける位置と姿勢を最適化する最適化問題を定式化した。推定誤差を最小化するものであるが、推定誤差の回転成分はマーカの貼り付け位置に依存しないので、推定誤差の並進成分の大きさを最小化することとした。実験を行い、提案した推定誤差評価方法が推定誤差の傾向をよく表していることが分かった。また、マーカの貼り付け位置と姿勢を最適化することにより推定誤差を低減することができることが分かった。なお、推定誤差の最大値については比較的低減効果が小さかったが、75パーセンタイル値では42.8%という大きな低減効果が観測された。

本研究では、作業位置の値を計算した値を評価したのみである。ロボットの手先を実際に作業位置に動かしてロボットの手先の位置を評価することが今後の課題である。

参考文献

- 1) 福本靖彦, 神内杜夫, 坂東慎之介, 竹中慎, 小林宏明, 長谷見健太郎, マーカの位置計測を伴うマニピュレーションにおける計測誤差の作業位置への伝播の近似計算, ロボティクス・メカトロニクス講演会講演概要集, 1P1-B01 (2024) .
- 2) 福本靖彦, 神内杜夫, 坂東慎之介, 竹中慎, 小林宏明, 長谷見健太郎: マーカベースマニピュレーションにおけるマーカ位置最適化, 計測自動制御学会システムインテグレーション部門講演会講演概要集, 3C2-05 (2024) .
- 3) 香川県 政策部政策課: 「人生100年時代のフロンティア県・香川」実現計画 (2024) .
- 4) 香川県 政策部統計調査課: 香川県人口移動調査報告 令和6年4月月報 (2024) .
- 5) 野田哲男: ものづくりにおける産業用ロボットの展開と今後の課題, システム/制御/情報, Vol. 65, No. 3, pp. 78-84 (2021) .
- 6) オムロン: 協調ロボットの特徴, <https://www.fa.omron.co.jp/product/robotics/lineup/collaborative/feature/> (検索日: 2025. 1. 14)
- 7) 宮森剛, 藤井洋之, 長嶋功一: 「NEXTAGE」の歩みと社会実装への取組み〜ヒト型ロボット「NEXTAGE」による挑戦〜, 川田技報, Vol. 43, pp. 14-20 (2024) .
- 8) 滝澤象太, 並木勇太, 安藤俊之, 丹野善博, 小田有記: ファナックロボットの最新ビジョン機能について, FANUC Technical Review, Vol. 30, No. 1, pp. 12-20 (2022) .
- 9) デンソーウェーブ: 簡単セットアップ, <https://www.denso-wave.com/fsys/ja/robot/event/RTJ2024/panel/03.pdf> (検索日: 2025. 1. 14)
- 10) 村田健二: 産業用ロボットFRシリーズ“MELFA Smart Plus”, 三菱電機技報, Vol. 92, No. 4, p. 35(249)-38(252) (2018) .
- 11) 鹿又智行: 安川電機の人協働ロボットと適用事例, ロボット, No. 271, pp. 21-25 (2023) .
- 12) ROBOTIQ: リストカメラ, <https://robotiq.com/ja/products/wrist-camera> (検索日: 2025. 1. 14)
- 13) OnRobot: Eyes Datasheet v3.3, https://onrobot.com/sites/default/files/documents/Datasheet_Eyes_EN_0.pdf (検索日: 2025. 2. 10) .
- 14) DMG 森精機: WH-AMR 10, <https://www.dmgmori.co.jp/products/machine/id=5205> (検索日: 2025. 2. 14)

産業用ロボットに対する大規模言語モデルの適用検討 —ChatGPT を用いたピック & プレイスプログラムの生成—

福本 靖彦

本研究では、ChatGPT を用いて産業用ロボットの動作プログラムを生成させた。対象タスクはピック & プレイス作業とした。生成されたプログラムには誤りが含まれる場合もあったが、修正指示を行うことで最終的に適切な動作を実現した。本稿では実装例と有効性の検討結果を示す。

1 緒言

ChatGPT の登場以降、大規模言語モデル (LLM: Large Language Model) が注目を集めている。ロボット分野でも有望な技術と目され、様々な研究が行われている^{1~5)}。そのような研究の一つに LLM を用いてロボットの動作プログラムを生成するものがある^{1, 2)}。同様の取り組みは産業界でも行われており、展示会にて LLM を活用したロボットの動作プログラム生成のデモンストレーションが行われている⁶⁾。

そこで本研究では、Vemprala らの研究²⁾を参考にして、LLM によりピック & プレイスを行うロボットの動作プログラムの生成実験を行った。実験では不適切なプログラムが生成される場合もあったが、修正指示を行うことで最終的に適切な動作が実現された。本稿ではその実装方法を記すとともに、この手法の有効性について検討した結果を示す。

2 Vemprala らの研究²⁾

Vemprala は ChatGPT for Robotics として、ChatGPT を用いてロボットの動作プログラムを作成する以下のようなフレームワークを提示している。

STEP 1: 必要なロボット機能に関する関数を準備する。

STEP 2: ChatGPT に入力するプロンプトを用意する。プロンプトには作業内容や、用意した関数について記述する。

STEP 3: ChatGPT が生成したプログラムを評価し、ChatGPT にフィードバックを返すループを繰り返す。

STEP 4: 完成したプログラムの運用を開始する。

関数は OpenCV など既存の技術であらかじめ用意した機能をラップする等により作成するもので、特にその振る舞いが分かるような関数名をつけることが重要としている。また必ずしも一発完動を想定しておらず、LLM の柔軟性を生かして STEP 3 でプログラムの修正を行う。特に、生成されるプログラムは必ずしも安全ではないため、生成したプログラムをリアルタイムで動作させることは注意が必要としている。プログラム言語は主に Python を用いる例が示されているが、XML 形式でコマンドを出力させる場合や、プログラムまで出力させずにコ

ードの流れを箇条書きで出力させるケースも考えられるとしている。

Vemprala らの研究ではいくつかの適用事例が示され、そのプロンプトも公開されているが、本研究ではその中の一つである 6 軸のアーム型ロボットで様々な色の直方体ブロックを吸着搬送した例を参考に示す。この事例では、Python による動作プログラムを生成させており、

- `grab()`
- `release()`
- `get_position(object)`
- `move_to(position)`

という 4 つの API が用意されている。それぞれ、グリッパの吸着を on にする関数、off にする関数、文字列 (“green block” など) で指定された対象物の三次元の位置と姿勢を返す関数、三次元の位置と姿勢を指定するとロボットがその位置に移動する関数である。このような設定で、以下のような順番でプログラムの作成を指示している。

- A) 対象物の名前を指定するとそれをピックアップするスキルを学習するよう指示する。
- B) 把持している物体を指定の位置に置くスキルを学習するよう指示する。
- C) 緑のブロックを所定のパッドの上におくプログラムを作成させる。
- D) 二つの茶色のブロックを緑のブロックの上に置くプログラムを作成させる。
- E) マイクロソフトのロゴのようにブロックを並べるプログラムを作成させる。

以上の A) や B) の処理は関数として生成され、C) 以降ではそれらが活用されている。D) では二つ目の茶色のブロックは緑のブロックの上の上に乗ることになるが、そのようなロジックについて ChatGPT の内部で処理してくれている。E) では ChatGPT が備えているマイクロソフトのロゴの色の並びに関する知識が活用され、それに応じた動作プログラムが生成されている。

なお、Vemprala らの研究では例えば B) を行うプロンプトは

```
Now I want you to learn how to place and object in a position. Considering you are holding a generic object, you first need to first move to a safe position above the target location, bring
```

the object down and then release it.

であった。手先を対象位置の手前に一度移動し、そこから直線的に手を伸ばすことはピック&プレースの常識的な動作であるが、このことを明示したプロンプトを用いている。つまり ChatGPT はマイクロソフトのロゴに関する知識は備えていても、ロボットのプログラミングについて十分な常識は持っていない可能性があることが示唆されている。しかしながら、適切なプロンプトを与えることで最終的にロボットの動作プログラムを生成できることを示している。

最後に、同研究では `grab()` などの四つの API を用意しているが、その実装は示されていない。したがって、Vemprala らのフレームワークを用いてロボットの動作プログラムを作成するには、API の実装方法から検討する必要がある。

3 実験

前章で示した Vemprala らの研究²⁾を参考にして、ChatGPT を用いてロボットの動作プログラムの生成を行った。対象タスクはピック&プレース動作とし、事前準備が必要な API の実装を行った。プログラムの生成を行うと、誤りのあるプログラムが出力された場合もあったが、どのような問題が生じたかを示して修正を指示することで最終的に正しいプログラムに修正された。

3.1 対象タスク

本研究では、タスクとして、アルミの直方体ワークをエアグリッパで把持して所定の場所に置き、その後、その直方体を元の位置に戻すという動作を扱った。これは、検査装置や加工装置にワークを供給・排出する動作をイメージしたものである。ワークは 5 つ用意し、これを順に処理する動作を行わせた。

3.2 実験装置

LLM として ChatGPT の無料版を用いた(おそらく GPT のバージョンは 4o with canvas が用いられた)。

ロボットは安川電機の MOTOMAN-SDA10F の右腕を用いた。グリッパは近藤製作所のエアグリッパ HA-4MS を使い、ロボットの汎用出力の 131 番により開閉できるようにした。ワークはアルミの直方体を用い、アルミフレーム等により組んだワーク置き場と、検査装置や加工装置等の機械のダミーを用意した。これらのハードウェアは図 1 のように配置した。

3.3 プログラム言語

本研究ではロボットプログラムを直接生成するのではなく、ロボットの動作を記述する Python プログラムを生成させるものとした。

安川電機のロボットでは INFORM という専用のロボット言語で動作プログラムが作成されるため、直接 INFORM のプログラムを生成するのが自然であるが、以下の二つの理由から Python での生成を行うこととした。一つは Vemprala らが Python でプログラムを生成していたので、それを踏襲するためである。もう一つは事前検討の段階で INFORM のプログラムを ChatGPT で生成しようとしたところ、そもそも文法的に問題のないプログラムを生成することが難しかったためである。INFORM によるプログラムの内容は JBI ファイルというテキストファイルに格納

する必要があるが、JBI ファイルに文法的な誤りがあった場合にはコントローラにファイルを読み込むことすらできず、デバグの手間が大きい。

なお、Python で動作プログラムを生成する場合、ロボットが Python に対応している場合⁷⁾や、ORiN などのミドルウェアを用いている場合には比較的容易に API を用意できると思われる。しかし、一般的には何らかの方法で Python と連携する仕組みを用意する必要が出てくる。このことについては次の項で説明する。

3.4 事前準備する API

2 節で示した Vemprala の指針の STEP 1 について説明する。具体的な API として、Vemprala らによるマネジュレーションの例で示されている四つの関数 `grab()`、`release()`、`get_position(object)`、`move_to(position)` を本研究でも用意した。別添の付録 1 にこれらの実装を示すとともに、以下でその概要について説明する。

本研究で用いるロボットは Python で制御する機能がないので、まず API ごとに必要なロボット動作を行うロボットジョブを INFORM で作成した。そして PC 側から、実行するジョブ番号と実行に必要な情報 (`move_to()` の場合、移動先の座標) をロボットの内部の変数に書き込むことで、ジョブを実行させる仕組みとした。

なお本来、`get_position(object)` は適切なセンサ等を用意して値を得る必要があるが、本研究ではダミーとしてあらかじめロボットをジョグ動作して調べた座標値を固定値で返す関数として実装した(これは、通常のティーチングを用いる場合と等価な処理である)。)
`move_to(position)` については Vemprala は障害物回避機能を含む経路生成機能を持たせることを示唆しているが、本研究では単なる直線補間の移動命令として実装した。移動速度は 5 mm/s の固定値とした。

3.5 ChatGPT に入力するプロンプトの作成方針

Vemprala が用いたプロンプト²⁾を日本語訳したものをベースにして、3.1 項のタスクを実行できるように以下のような流れで動作プログラムを生成させる。

- ワークを指定するとそれをピックアップするスキルを学習するよう指示する。
- 把持しているワークを指定の位置に置くスキルを学習するよう指示する。
- ワークをピックアップして機械に置き、3 秒待つてからそのワークを元の位置に戻すスキルを学習するよう指示する。

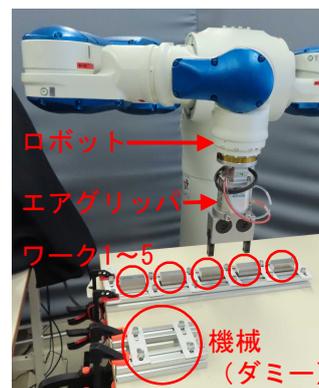


図 1 実験装置

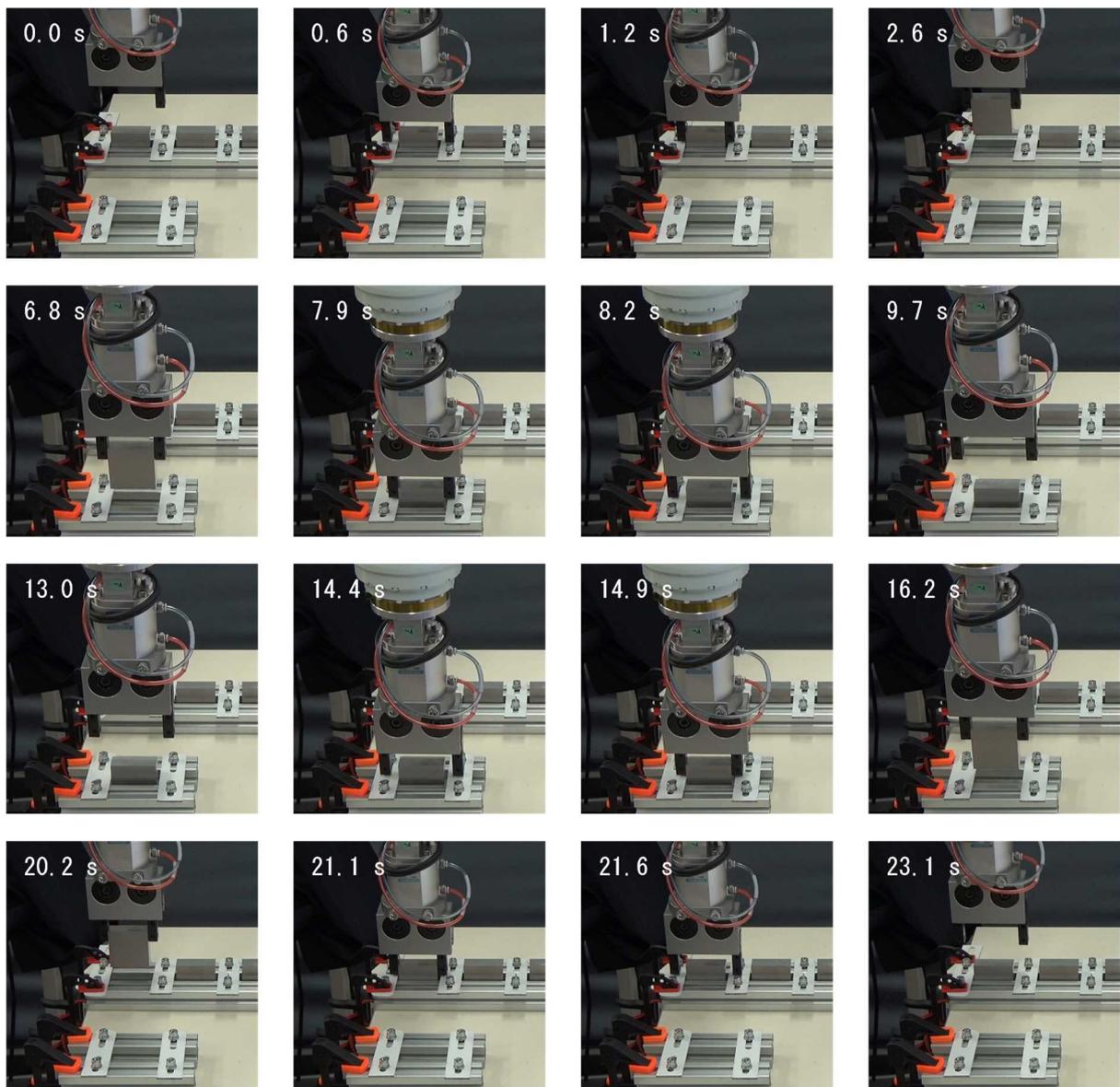


図2 最終的なロボットの動作の様子（一つめのワークの処理のみ）

D) ワーク 1 からワーク 5 を順に機械にセットし、3 秒待ってから元の位置に戻す動作プログラムを作成するよう指示する。

生成されたプログラムはロボットで動作させ、問題が生じた際には修正の指示を行った。修正の指示は、ただ起こった問題を記述するのみか、修正方法をやや曖昧に指示するという方法で行った。

3. 6 実験結果

プログラムを生成させると不具合のあるプログラムが出力されたため修正の指示を必要としたが、10 往復のやり取りを通して最終的に所望の動作を実現するプログラムが作成できた。本実験で行った ChatGPT とのやり取りを別添の付録 2 に示す。ロボットの動作の様子を図 2 に示す。

プログラムに不具合があった場合の修正指示においては明確な修正方法を指示しなかったが、うまく修正が行われた。これは ChatGPT の柔軟な判断能力に依るものと思われる。したがって、ロボットや対象作業に関する知識や経験があまりない作業者であっても動作プログラム

を作成できる可能性がある（ただし、産業用ロボットの教示等の業務に係る特別教育を受講する必要がある。）。

ただし、一度の指摘で適切に修正ができなかった場合があったほか、動作が不適切なだけでなく、そもそも文法エラーが生じた場合もあった。また、最終的なプログラムの品質があまり良くないと思われたケースもあった。したがって、複雑な作業を行わせる場合には修正に労力を要する可能性もあると思われた。ただし、これはあらかじめ用意した API が比較的低位の API であったことも原因であると思われる。例えば、`move_to()` が単なる移動命令ではなく、障害物回避機能を含む移動命令であれば生じないような問題も観測された。

なお、指示に不明点がある場合には聞き返すよう指示していたが、聞き返されることはなかった。したがって勝手な判断で修正が行われた箇所もあったが、本実験においてはむしろ効率的に修正された結果となった。場合によってはあらかじめ聞き返してくれることで誤った試行を減らすこともできると思われるが、すべてを逐一確認されることも作業者にとって手間であり、柔軟性との

トレードオフになるところと思われる。

以下では手直しを要した三つの箇所について説明する。

3. 6. 1 修正1 (グリッパの初期化)

ワークをピックアップするスキルの学習では、グリッパの初期化処理が欠けており、グリッパの初期状態が閉じた状態であると、ワークに手先が衝突するものだった。これに対して、

このコードを実行した際、グリッパの初期状態が閉じた状態であったため、把持位置に移動する際にワークに衝突してしまいました。グリッパの初期状態が閉じた状態であっても問題ないようにコードを修正してもらえますか？

という指示を行うと、ワークに近づく前にグリッパを開く処理を加える修正が行われた。グリッパを開く初期化処理を加えるという具体的な指示を行った訳ではなく、ただ起こった問題を説明し、対処するよう指示しただけであったが、適切なプログラムに修正することができた。

3. 6. 2 修正2 (ワークを元の位置に置くデバグ)

ワークをピックアップする関数 `pick_up_workpiece()` とワークを置く関数 `place_workpiece()` を作成した後に、

素晴らしい！では、これらのスキルを使っていずれかのワークを拾い、機械の所定の場所に置き3秒待ってからワークを元の位置に戻す関数を追加してください。

と指示した際、以下のような関数 `process_workpiece()` が生成された。

```
def process_workpiece(workpiece):
    # ワークを把持
    pick_up_workpiece(workpiece)

    # 機械に配置
    place_workpiece("machine")

    # 3秒待機
    time.sleep(3)

    # ワークを元の位置に戻す
    pick_up_workpiece("machine")
    place_workpiece(workpiece)
```

ここでは、最終行ではワークを元の位置に戻すという作業を行うために `place_workpiece(workpiece)` と書かれている。 `process_workpiece()` 関数の最初の行では `pick_up_workpiece(workpiece)` を行っており、そこに戻すという処理なので一見正しいように見える。しかし実際には、この行を実行する時点ではワークはロボットのグリッパの中にあるため、適切な動作とならなかった。そこで、

これを実行すると、ワークを再度把持して持ち上げた後に、更にもう一度50mm上方に上がってから50mm下がるという動作をして空中でワークを解放しました。元のワーク置き場に戻すようにプログラムを修正してください。

として生じた不具合を示して修正指示を行うと、

なるほど、ワークを再把持した後の移動処理に問題がありました。

`pick_up_workpiece("machine")` を実行すると、ワークを掴んで持ち上げた後に、再び `place_workpiece(w`

`orkpiece)` を実行する際、安全高さに一度移動してしまうのが原因です。

修正内容

- `pick_up_workpiece("machine")` の最後の持ち上げ動作を削除
- そのまま `place_workpiece(workpiece)` を呼び出してワークを元の場所に戻せるようにしました。

(略)

としてプログラムの修正が行われた。これは問題を正しく認識していない修正であり、不具合は解消されなかった。

そこで、

そもそも、ワーク置き場に戻っていません。コードを修正してもらえますか？

として再度の修正を指示すると、以下のように最初の行で `pick_up_workpiece()` を行ったときの位置を覚えておく処理を追加し、`place_workpiece()` にその位置情報を与える修正を行った。

```
def pick_up_workpiece(workpiece):
    (略)
    return position # **元の位置を返す**

def place_workpiece(target_location):
    # 目標地点の配置位置を取得
    position = get_position(target_location)
    (略)

def process_workpiece(workpiece):
    # **元のワーク位置を取得**
    original_position = ¥
    get_position(workpiece)
    (略)
    # **元のワーク位置に戻す**
    place_workpiece(original_position)

# ワーク1を処理
process_workpiece("work1")
```

まず、こちらから与えた指示中の「修正してもらえますか？」には誤植があり、正しくは「修正してもらえますか？」と入力すべきであったが、ChatGPTの柔軟性により問題なく処理されているようであった。次に、ChatGPTが示した修正に関する考え方は的確であるが、その実装を見ると `place_workpiece()` はオブジェクト名の文字列を引数とする関数であるのに、座標値を与えているというバグがあった。

したがって、

ワークを元の位置に戻す際に、ロボットが動作できませんでした。適切な座標値が得られていないようです。プログラムを修正してもらえますか？

として再度修正を指示したところ、

```
def process_workpiece(workpiece,
workplace):
    # **ワークを把持**
    pick_up_workpiece(workpiece)
```

(略)

```
# **元のワーク置き場に戻す**  
place_workpiece(workplace)  
  
# **ワーク1をワーク置き場1に戻す**  
process_workpiece("work1", "workplace1")
```

として、先のプログラムから考え方を変更し、最終行の `place_workpiece()` に与える引数をワーク置き場に変えることで所望の動作を実現した。ただしこの方法では、ワークの元の位置がワーク置き場であることが仮定されるので十分な汎用性を持たず、また `process_workpiece()` もワーク名とワーク置き場名の両方を引数として与える冗長な実装となったため、プログラムの品質はあまり高くない。先の、最初にワーク置き場からピックアップした際の位置を覚える考え方の方が良い方針と思われるが、これは採用されなかった。

3. 6. 3 修正3 (軌道の修正)

ワーク1~5を順番に機械にセットし、元の位置に戻す動作プログラムを作成させたところ、ロボットの移動中に手先がワークに干渉しそうなケースが生じた。そこで、

これを実行すると、ワークを元の位置に戻した後に次のワークを取る際、手先がワークに衝突しそうでした。ワークを置いた後に安全な位置に退避する動作を加えてもらえますか？

として修正を指示すると、指示通りにプログラムを修正した。その回答の中で

なるほど！
ワークを元の位置に戻した後、そのまま次のワークを取りに行くと、グリッパの位置が低いため衝突のリスクがある ということです。

として、問題の状況を理解しているような文言が含まれていた。なお、上記の修正指示内にある「安全な位置」については明確には示していないが、これについて聞き返してくることはなく、ワークを指定の位置に置いた後に手先を50mm上方に移動する処理が加えられた。ピックアップを行うサブタスクのプログラムを行う指示の中で、

なお、その際、対象物から安全な距離(50mm)上に移動してから対象物に手を伸ばし、掴むようにしてください。

という指示を行っていたので、ここから類推して勝手に判断したものと考えられる。

4 結言

本研究ではChatGPT for Roboticsのプロンプトを参考にして、五つのワークをピック&プレイスする動作プログラムをChatGPTにより生成した。ChatGPTへの指示は日本語の自然言語で行った。なおロボットの動作や認識に関するAPIをあらかじめ用意しておく必要があったので、(1)Pythonに対応していないロボットをPythonで動かすためのプログラムを自作し、その実装を示した。プログラムの生成過程において三つの不具合が生じたが、修正指示を行うことで最終的に適切なプログラムに修正された。なお修正指示においては、生じた不具合について説明はしたものの、どのように修正するかは指示しな

かったが、ChatGPT自身が解決策を提示した。ChatGPTとのやり取りの回数は10往復であった。

本実験を通して、(2)あらかじめ適切なAPIが準備されていれば、ロボットや対象作業について知識や経験があまりない作業者でもロボットの動作プログラミングが可能となると思われた。また、(3)どのレベルのAPIを用意できるかがプログラムの作成効率を大きく左右すると思われた。本実験では、単に手先を指定の座標まで移動させる低位のAPIを用いたが、生成されたプログラムの不具合の中には、障害物回避を行う経路生成まで行うような高位のAPIとして用意されていれば生じなかったようなものもあった。

なお、上で示したような高位のAPIの機能をモデルに内包したEnd-to-Endの手法も提案されており^{3~5)}、一部はモデルが公開されている^{4, 5)}。このようなEnd-to-Endのモデルについては使用するロボットや実タスクに合わせてファインチューニングが必要であるので今回は検討を行わなかったが、今後、このようなモデルの評価も行いたい。

また先行研究²⁾に示されたプロンプトを確認すると、ChatGPTにはロボットや製造業の常識が十分備わっているとは限らないことが示唆されていた。これに関して、産業技術総合研究所では製造業も想定したロボット基盤モデルの開発を発表しており⁸⁾、今後の研究の進展が期待される。

参考文献

- 1) J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence and A. Zeng: Code as Policies: Language Model Programs for Embodied Control, arXiv:2209.07753 (2022).
- 2) S. H. Vemprala, R. Bonatti, A. Buckner and A. Kapoor: ChatGPT for Robotics: Design Principles and Model Abilities, IEEE Access, Vol. 12, pp. 55682-55696 (2024).
- 3) A. Brohan, et al.: RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control, arXiv: 2307.15818 (2023).
- 4) Octo Model Team et al.: Octo: An Open-Source Generalist Robot Policy, Proc. of Robotics: Science and Systems (2024).
- 5) M. J. Kim et al.: OpenVLA: An Open-Source Vision-Language-Action Model, arXiv:2406.09246 (2024).
- 6) デンソーウェーブ: ChatGPTを活用した自然言語によるロボットプログラムの生成, <https://www.denso-wave.com/fsys/ja/robot/event/irex2023/panel/c02.pdf> (検索日: 2025. 2. 13)
- 7) 森岡昌宏: 2023 国際ロボット展 ファナック出展概要, ロボット, No. 277, pp. 35-37 (2024).
- 8) 産業技術総合研究所: 実世界の困難作業自動化を目指したロボット基盤モデルの研究開発を本格始動, https://www.aist.go.jp/aist_j/news/pr20250123_2.html (検索日: 2025. 02. 14)

付録1 APIの実装

ChatGPT for Robotics の方法では、あらかじめPythonプログラムでAPIを用意しておく必要がある。本研究でのAPIの実装について説明する。なお、APIはPython内で完結せず、ロボットのプログラミングペンダントで構築したINFORMによるジョブと、ロボット内で通信を用いてロボット内の変数を制御するMotoPlus（安川電機のロボットのC言語による開発環境）によるC言語プログラムも用いている。これらについて付録1.1～付録1.3で説明する。

付録1.1 PC側プログラム

```
import socket
import time

# 通信の準備
DstIP = "192.168.x.x" # ロボットの IP アドレス
DstPort = x
DstAddr = (DstIP,DstPort)
sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
sock.connect( DstAddr )

#=====#
#=====#

#=====#
# グリッパを閉じる API
def grab():

    # グリッパを閉じる指令(6個の数値はダミー)
    data = "grab,0,0,0,0,0,0"
    print("send: ", data)
    data = data.encode('utf-8')
    sock.send(data)

    # ジョブ側が完了するのを待つ
    #(B000 が 0 になったら返事が来る)
    data = sock.recv(1024)
    data = data.decode()
    print(data)

    # 閉じるまで待つ
    time.sleep(0.5)

    return

#=====#
# グリッパを開く API
def release():

    # 送信(6個の数値はダミー)
    data = "release,0,0,0,0,0,0"
    print("send: ", data)
    data = data.encode('utf-8')
    sock.send(data)
    # 受信
    data = sock.recv(1024)
    data = data.decode()
    print(data)
    # 開くまで待つ
    time.sleep(0.5)

    return
```

```
#=====#
# 文字列で与えられる object について、把持するための
# 手先位置を返す API
def get_position(object):

    positions = {¥
        "work1": [449.950, -323.726, ¥
                  -161.126, 178.7508, ¥
                  0.1462, -88.2750], ¥
        "work2": [447.815, -238.439, ¥
                  -160.626, 178.7508, ¥
                  0.1462, -88.2750], ¥
        "work3": [442.521, -152.887, ¥
                  -160.126, 178.7508, ¥
                  0.1462, -88.2750], ¥
        "work4": [438.623, -66.655, ¥
                  -159.626, 178.7508, ¥
                  0.1462, -88.2750], ¥
        "work5": [434.235, 16.754, ¥
                  -159.126, 178.7508, ¥
                  0.1462, -88.2750], ¥
        "workplace1": [449.950, -323.726, ¥
                       -161.126, 178.7508, ¥
                       0.1462, -88.2750], ¥
        "workplace2": [447.815, -238.439, ¥
                       -160.626, 178.7508, ¥
                       0.1462, -88.2750], ¥
        "workplace3": [442.521, -152.887, ¥
                       -160.126, 178.7508, ¥
                       0.1462, -88.2750], ¥
        "workplace4": [438.623, -66.655, ¥
                       -159.626, 178.7508, ¥
                       0.1462, -88.2750], ¥
        "workplace5": [434.235, 16.754, ¥
                       -159.126, 178.7508, ¥
                       0.1462, -88.2750], ¥
        "machine" : [637.378, -289.386, ¥
                    -156.379, 178.7508, ¥
                    0.1462, -88.2750]}

    return positions[object]

#=====#
# position の位置へ移動する API
# なお、position は[mm]/[deg]単位なので、
# [um]/[0.0001deg]単位に変換して送る
def move_to(position):

    position_um = []
    for i in range(0,3):
        position_um.append(¥
            (int)(position[i]*1000) )
    for i in range(3,6):
        position_um.append(¥
            (int)(position[i]*10000) )

    # 送信
    data = "move_to,{},{},{},{},{,} ".¥
        format(position_um[0],position_um[1],¥
            position_um[2],position_um[3],¥
            position_um[4],position_um[5])
    print("send: ", data)
    data = data.encode('utf-8')
    sock.send(data)
    # 受信
```

```

data = sock.recv(1024)
data = data.decode()
print(data)

#-----
# ここに ChatGPT が生成したプログラムを入れる
#-----

# 通信を終わらせる
sock.shutdown(socket.SHUT_RDWR)
sock.close()

```

付録 1. 2 INFORM によるロボットジョブ

API に対応するジョブ（下位ジョブ）と、それを実行する上位ジョブを作成した。上位ジョブは無限ループの中で変数B000の値に応じていずれかの下位ジョブを実行してからB000の値を初期化する。なお、どの下位ジョブも実行しない場合もある。作成したロボットジョブのコードを以下に示す。なお、`get_position(object)`に対応するジョブはロボットではなくセンサの処理であると考えられるので、対応するジョブは作成していない。

2024LLM-PYTHON-MOTOPLUS. JBI

```

/JOB
//NAME 2024LLM-PYTHON-MOTOPLUS
//POS
///NPOS 0,0,0,0,0,0
//INST
///DATE 2025/02/05 16:44
///ATTR SC, RW
///GROUP1 RB2
NOP
*LABEL
CALL JOB:2024LLM-GRASP IF B000=1
CALL JOB:2024LLM-RELEASE IF B000=2
CALL JOB:2024LLM-MOVE_TO IF B000=3
SET B000 0
JUMP *LABEL
END

```

2024LLM-GRASP. JBI

```

/JOB
//NAME 2024LLM-GRASP
//POS
///NPOS 0,0,0,0,0,0
//INST
///DATE 2025/02/05 16:28
///ATTR SC, RW
///GROUP1 RB2
NOP
DOUT OT#(131) ON
END

```

2024LLM-RELEASE. JBI

```

/JOB
//NAME 2024LLM-RELEASE
//POS
///NPOS 0,0,0,0,0,0

```

```

//INST
///DATE 2025/02/05 16:29
///ATTR SC, RW
///GROUP1 RB2
NOP
DOUT OT#(131) OFF
END

```

2024LLM-MOVE_TO. JBI

```

/JOB
//NAME 2024LLM-MOVE_TO
//POS
///NPOS 0,0,0,1,0,0
///TOOL 1
///POSTYPE PULSE
///PULSE
P00000=-9221, 39644, -
122551, 107639, 95958, 30418, 14053
//INST
///DATE 2025/02/05 16:29
///ATTR SC, RW
///GROUP1 RB2
NOP
MOVJ P000 VJ=5.00
END

```

付録 1. 3 ロボット側 C 言語プログラム

TCP/IP によるソケット通信で PC と通信し、ロボット内の変数を制御するプログラムを作成した。このプログラムは INFORM で作製したジョブと同時実行される。通信では、API 名と移動先の座標値をテキスト形式でやり取りした。C 言語プログラムでの処理の概要を以下に示す。

```

STEP 1: PC からの指令を受信する。
STEP 2: コマンドを解析し、どの API が実行されたかを読み取る。
STEP 3: ロボット内の変数 (B000) に API に応じた値を書き込む。また move_to() の場合には位置変数 P000 にも移動先の座標を書き込む。
STEP 4: INFORM によるジョブが B000 を 0 にするのを待つ。
STEP 5: API の実行完了を PC 側に送信する。
STEP 6: STEP 1 に戻る。

```

なお、一般的にはフィールドバス等を用いて直接必要な変数の値を書き込む方がより実装しやすいと考えられる。その場合、Python の API の定義の中で B000 が 0 に戻ることを定期的に確認する処理等を要する。

付録 2 プロンプト

ChatGPT とやり取りの履歴を示す。なお、灰色の文字は著者の入力したプロンプトであり、それ以外は ChatGPT の回答である。読みやすくするために改行やインデントを改変している場合がある。

```

6自由度の産業用ロボットがあります。エンドエフェクタとして平行グリッパが取り付けられています。シーンとタスクが与えられたら、このロボットの動作プログラムを Python で作成してください。

```

以下の関数を使うことができます：

grab() : 平行グリッパを閉じて物体をつかむ。
release() : 平行グリッパを開いて物体を開放する。
get_position(target) : 対象物の文字列が与えられると、その対象物が把持可能なものである場合には、把持するためのロボットの手先の座標と向き[X, Y, Z, Roll, Pitch, Yaw]を返す。あるいは対象物が把持可能なものではなく、別の把持物体をその対象物上に置くことができる場合には、把持対象物をその対象物上に置くためのロボットの手先の座標と向き[X, Y, Z, Roll, Pitch, Yaw]を返す。
move_to(position) : 位置[X, Y, Z, Roll, Pitch, Yaw]が与えられると、ロボットの手先をその位置に移動させる。

これらの関数を使用して新しい関数を作成して使うこともできますし、このほかのPythonの標準的な関数を使うこともできます。ただし、他の関数を定義せずに使用することはできません。

解答は単純明快にしてください。位置はmm単位で、角度は度単位で与えます。また、“Question - ”というタグを使って、明確な質問をすることもできます。以下は、明確な質問をする方法を説明するシナリオの例です。

あるシーンに2つの球体があるとします。

私：球体を取ってください。
あなた：Question - 球が2つあります。どちらを拾えばいいですか？
私：球体1をお願いします。

準備はいいですか？

準備完了です！シーンとタスクを教えてください。

シーン：
ロボットの前のテーブルには以下の対象物があります。
・ワーク1~5
・ワーク置き場1~5
・機械
ワークはアルミの直方体で、その大きさは幅50mm、奥行き15mm、高さ50mmです。ワーク置き場と機械にはワークを置くことができます。アルミの直方体を把持する際には上方から近づいて取ってください。また、アルミの直方体を置く場合にも、上方から近づいて置いてください。

タスク：

アルミの直方体の一つ手に取り、それを把持するスキルを学んでください。なお、その際、対象物から安全な距離（50mm）上に移動してから対象物に手を伸ばし、掴むようにしてください。

把持スキルを学習するためのシンプルなPythonプログラムを作成しました。ワーク1を把持する例として記述しています。

```
def pick_up_workpiece(workpiece):  
    # ワークの把持位置を取得  
    position = get_position(workpiece)  
  
    # 安全な高さへ移動  
    safe_position = [  
        position[0], position[1],  
        position[2] + 50, position[3],  
        position[4], position[5]  
    ]  
    move_to(safe_position)  
  
    # 把持位置へ移動  
    move_to(position)  
  
    # ワークを把持  
    grab()  
  
    # 安全な高さへ戻る  
    move_to(safe_position)  
  
# ワーク1を掴む  
pick_up_workpiece("work1")
```

このコードでは、以下の手順でワークを把持します。

1. get_position("work1") でワーク1の把持位置を取得。
2. 安全な高さ（50mm上）に移動。
3. ワークの上方から近づいて把持位置に移動。
4. grab() でワークをつかむ。
5. 再び安全な高さに移動。

これでロボットが安全にワークを掴む動作を学習できます。他のワークを把持したい場合は“work1”を“work2”などに変更してください。

このコードを実行した際、グリッパの初期状態が閉じた状態であったため、把持位置に移動する際にワークに衝突してしまいました。グリッパの初期状態が閉じた状態であっても問題ないようにコードを修正してもらえますか？

グリッパが閉じた状態でも問題なく動作するように、ワークに近づく前に release() を追加して開くように修正しました。

修正後のコード：

```
def pick_up_workpiece(workpiece):  
    # ワークの把持位置を取得  
    position = get_position(workpiece)  
  
    # グリッパを開く（衝突を防ぐため）  
    release()  
  
    # 安全な高さへ移動
```

```

safe_position = [¥
    position[0], position[1], ¥
    position[2] + 50, position[3], ¥
    position[4], position[5]]
move_to(safe_position)

# 把持位置に移動
move_to(position)

# 把持
grab()

# 持ち上げる
move_to(safe_position)

# ワーク1を把持
pick_up_workpiece("work1")

```

修正内容:

1. `release()` を追加
 - ワークに近づく前にグリッパを開くことで、閉じた状態でも衝突を防ぎます。
- これで、グリッパの初期状態が閉じていても問題なくワークを掴めます！

素晴らしい！次は、既に把持している対象物を所定の位置に配置するスキルを学んでください。一般的な物体操作を行うものとして、まず、目標地点の上方の安全な位置まで移動し、物体を降ろし、それから解放するようにしてください。

把持している対象物を所定の位置に配置するためのPythonプログラムを作成しました。

配置スキルの学習コード:

```

def place_workpiece(target_location):
    # 目標地点の配置位置を取得
    position = ¥
        get_position(target_location)

    # 安全な高さに移動
    safe_position = [¥
        position[0], position[1], ¥
        position[2] + 50, position[3], ¥
        position[4], position[5]]
    move_to(safe_position)

    # 配置位置に移動
    move_to(position)

    # 解放
    release()

    # 持ち上げる
    move_to(safe_position)

# ワーク1をワーク置き場1に配置
place_workpiece("workplace1")

```

動作手順:

1. `get_position(target_location)` で目標地点の配置位置を取得。
2. まず **安全な高さ (50mm 上)** に移動。

3. **配置位置に移動** して対象物を降ろす。
4. `release()` でグリッパを開き、対象物を解放。
5. **再び安全な高さへ移動** して終了。

これで、把持した対象物を安全に配置できます！

素晴らしい！では、これらのスキルを使っていずれかのワークを拾い、機械の所定の場所に置き3秒待ってからワークを元の位置に戻す関数を追加してください。

ありがとうございます！

以下の関数 `process_workpiece(workpiece)` を追加しました。この関数は、指定したワークを拾い、機械に配置し、3秒待機してから元の位置に戻します。

コード

```

import time

def pick_up_workpiece(workpiece):
    # ワークの把持位置を取得
    position = get_position(workpiece)

    # グリッパを開く (衝突を防ぐため)
    release()

    # 安全な高さに移動
    safe_position = [¥
        position[0], position[1], ¥
        position[2] + 50, position[3], ¥
        position[4], position[5]]
    move_to(safe_position)

    # 把持位置に移動
    move_to(position)

    # 把持
    grab()

    # 持ち上げる
    move_to(safe_position)

def place_workpiece(target_location):
    # 目標地点の配置位置を取得
    position = ¥
        get_position(target_location)

    # 安全な高さに移動
    safe_position = [¥
        position[0], position[1], ¥
        position[2] + 50, position[3], ¥
        position[4], position[5]]
    move_to(safe_position)

    # 配置位置に移動
    move_to(position)

    # 解放
    release()

    # 持ち上げる
    move_to(safe_position)

def process_workpiece(workpiece):

```

```

# ワークを把持
pick_up_workpiece(workpiece)

# 機械に配置
place_workpiece("machine")

# 3秒待機
time.sleep(3)

# ワークを元の位置に戻す
pick_up_workpiece("machine")
place_workpiece(workpiece)

# ワーク1を処理
process_workpiece("work1")

```

動作手順

1. ワークを把持
(pick_up_workpiece(workpiece))
2. 機械に配置 (place_workpiece("machine"))
3. 3秒待機 (time.sleep(3))
4. ワークを再度把持
(pick_up_workpiece("machine"))
5. 元の位置に戻す
(place_workpiece(workpiece))

これで、ワークを拾って機械に置き、3秒待機した後、元の位置に戻せます！ 🔄

これを実行すると、ワークを再度把持して持ち上げた後に、更にもう一度50mm上方に上がってから50mm下がるという動作をして空中でワークを解放しました。元のワーク置き場に戻すようにプログラムを修正してください。

なるほど、ワークを再把持した後の移動処理に問題がありました。

pick_up_workpiece("machine") を実行すると、ワークを掴んで持ち上げた後に、再び

place_workpiece(workpiece) を実行する際、安全高さに一度移動してしまうのが原因です。

修正内容

- pick_up_workpiece("machine") の最後の持ち上げ動作を削除
 - そのまま place_workpiece(workpiece) を呼び出してワークを元の場所に戻せるようにしました。

修正後のコード

```

import time

def pick_up_workpiece(workpiece):
    # ワークの把持位置を取得
    position = get_position(workpiece)

```

```

# グリッパを開く (衝突を防ぐため)
release()

```

```

# 安全な高さに移動
safe_position = [¥
    position[0], position[1], ¥
    position[2] + 50, position[3], ¥
    position[4], position[5]]
move_to(safe_position)

```

```

# 把持位置に移動
move_to(position)

```

```

# 把持
grab()

```

```

# 持ち上げる (元の高さに戻す)
move_to(safe_position)

```

```

def place_workpiece(target_location):
    # 目標地点の配置位置を取得
    position = ¥
    get_position(target_location)

```

```

# 安全な高さに移動
safe_position = [¥
    position[0], position[1], ¥
    position[2] + 50, position[3], ¥
    position[4], position[5]]
move_to(safe_position)

```

```

# 配置位置に移動
move_to(position)

```

```

# 解放
release()

```

```

# **ここで持ち上げないように修正**

```

```

def process_workpiece(workpiece):
    # ワークを把持
    pick_up_workpiece(workpiece)

```

```

# 機械に配置
place_workpiece("machine")

```

```

# 3秒待機
time.sleep(3)

```

```

# **ワークを元の位置に戻す**
pick_up_workpiece("machine")
place_workpiece(workpiece)

```

```

# ワーク1を処理
process_workpiece("work1")

```

修正ポイント

1. place_workpiece() の最後の持ち上げ動作を削除

- これにより、ワークを空中で解放する問題を防ぎます。
2. 元の位置に戻す際の処理を修正
- `pick_up_workpiece("machine")` で持ち上げた後、そのまま `place_workpiece(workpiece)` を実行して、ワークを正しく戻します。

期待される動作

- ☑ ワークを掴んで持ち上げる
- ☑ 機械に置く
- ☑ 3秒待機
- ☑ 再びワークを掴む
- ☑ 元のワーク置き場に正しく戻す 🔄

これで意図した動作になるはずです！ ✍️

そもそも、ワーク置き場に戻っていません。コードを修正してもらえますか？

確かに、`place_workpiece(workpiece)` の呼び出しで **ワーク置き場の位置を取得し直していなかった** ため、元のワーク置き場に戻っていませんでした。修正します！

修正内容

1. ワークの元の位置を保持
 - `process_workpiece(workpiece)` で **最初に取得したワークの位置を記録** し、それを `place_workpiece()` に渡す。
2. 正しくワーク置き場に戻す
 - `place_workpiece(original_position)` で、ワークの元の位置に戻せるようにする。

修正後のコード

```
import time

def pick_up_workpiece(workpiece):
    # ワークの把持位置を取得
    position = get_position(workpiece)

    # グリッパを開く (衝突を防ぐため)
    release()

    # 安全な高さに移動
    safe_position = [position[0], position[1],
                    position[2] + 50, position[3], position[4],
                    position[5]]
    move_to(safe_position)

    # 把持位置に移動
    move_to(position)
```

```
# 把持
grab()

# 持ち上げる
move_to(safe_position)

return position # **元の位置を返す**

def place_workpiece(target_location):
    # 目標地点の配置位置を取得
    position = get_position(target_location)

    # 安全な高さに移動
    safe_position = [position[0], position[1],
                    position[2] + 50, position[3], position[4],
                    position[5]]
    move_to(safe_position)

    # 配置位置に移動
    move_to(position)

    # 解放
    release()

def process_workpiece(workpiece):
    # **元のワーク位置を取得**
    original_position =
    get_position(workpiece)

    # ワークを把持
    pick_up_workpiece(workpiece)

    # 機械に配置
    place_workpiece("machine")

    # 3秒待機
    time.sleep(3)

    # ワークを再把持
    pick_up_workpiece("machine")

    # **元のワーク位置に戻す**
    place_workpiece(original_position)

# ワーク1を処理
process_workpiece("work1")
```

修正ポイント

- ☑ `pick_up_workpiece()` で **ワークの元の位置を取得し、記録** する
 - ☑ **記録した位置に戻す** (`place_workpiece(original_position)`)
- これでワークが正しく元のワーク置き場に戻るはずです！ ✍️

ワークを元の位置に戻す際に、ロボットが動作できませんでした。適切な座標値が得られていないようです。プログラムを修正してもらえますか？

なるほど、`original_position = get_position(workpiece)` の取得方法に問題がありそうですね。

`get_position(workpiece)` は 把持用の座標を返すので、配置のための座標とは異なる可能性があります。

修正内容

1. 元の位置を保存する際に `get_position(workpiece)` を使わない
 - 代わりに、ワークを配置するための位置を `get_position(workplace)` で取得する。
2. `process_workpiece()` で元のワーク置き場を指定
 - `place_workpiece(workplace)` を使い、元のワーク置き場に戻せるようにする。

修正後のコード

```
import time

def pick_up_workpiece(workpiece):
    # ワークの把持位置を取得
    position = get_position(workpiece)

    # グリッパを開く (衝突を防ぐため)
    release()

    # 安全な高さに移動
    safe_position = [¥
                    position[0], position[1], ¥
                    position[2] + 50, position[3], ¥
                    position[4], position[5]]
    move_to(safe_position)

    # 把持位置に移動
    move_to(position)

    # 把持
    grab()

    # 持ち上げる
    move_to(safe_position)

def place_workpiece(target_location):
    # 目標地点の配置位置を取得
    position = ¥
    get_position(target_location)

    # 安全な高さに移動
    safe_position = [¥
                    position[0], position[1], ¥
                    position[2] + 50, position[3], ¥
                    position[4], position[5]]
    move_to(safe_position)

    # 配置位置に移動
    move_to(position)

    # 解放
    release()

def process_workpiece(workpiece, workplace):
```

```
# **ワークを把持**
pick_up_workpiece(workpiece)

# **機械に配置**
place_workpiece("machine")

# **3 秒待機**
time.sleep(3)

# **ワークを再把持**
pick_up_workpiece("machine")

# **元のワーク置き場に戻す**
place_workpiece(workplace)

# **ワーク 1 をワーク置き場 1 に戻す**
process_workpiece("work1", "workplace1")
```

修正ポイント

- ✓ `get_position(workpiece)` ではなく、`get_position(workplace)` を使用
- ✓ `process_workpiece(workpiece, workplace)` で元のワーク置き場を指定 する
- ✓ `place_workpiece(workplace)` を使って 確実に元の位置に戻す

期待される動作

1. ワーク 1 を把持
2. ワークを機械に配置
3. 3 秒待機
4. ワークを再把持 → ワーク置き場 1 に正しく戻す 🔄

これで、ワークを確実に元のワーク置き場に戻せるはずですよ! 📌

素晴らしい! では、このスキルを使って、ワーク 1 からワーク 5 に対して順にワークを機械へ移動し、それぞれ 3 秒まってから元の位置へ戻すというタスクを行うプログラムを作成してください。

ありがとうございます!

ワーク 1 からワーク 5 まで順番に処理するプログラムを作成しました。

プログラムの動作

1. ワーク 1~ワーク 5 を順番に把持
2. それぞれ機械に配置し、3 秒待機
3. 元のワーク置き場に戻す

コード

```
import time

def pick_up_workpiece(workpiece):
```

```

# ワークの把持位置を取得
position = get_position(workpiece)

# グリッパを開く (衝突を防ぐため)
release()

# 安全な高さに移動
safe_position = [¥
    position[0], position[1], ¥
    position[2] + 50, position[3], ¥
    position[4], position[5]]
move_to(safe_position)

# 把持位置に移動
move_to(position)

# 把持
grab()

# 持ち上げる
move_to(safe_position)

def place_workpiece(target_location):
# 目標地点の配置位置を取得
position = ¥
    get_position(target_location)

# 安全な高さに移動
safe_position = [¥
    position[0], position[1], ¥
    position[2] + 50, position[3], ¥
    position[4], position[5]]
move_to(safe_position)

# 配置位置に移動
move_to(position)

# 解放
release()

def process_workpiece(workpiece, workplace):
# **ワークを把持**
pick_up_workpiece(workpiece)

# **機械に配置**
place_workpiece("machine")

# **3秒待機**
time.sleep(3)

# **ワークを再把持**
pick_up_workpiece("machine")

# **元のワーク置き場に戻す**
place_workpiece(workplace)

# **ワーク1~ワーク5を順番に処理**
workpieces = [("work1", "workplace1"),
    ("work2", "workplace2"),
    ("work3", "workplace3"),
    ("work4", "workplace4"),
    ("work5", "workplace5")]

for workpiece, workplace in workpieces:
    process_workpiece(workpiece, ¥
        workplace)

```

ポイント

- ✓ `process_workpiece()` を使い回し、ワークの移動を簡潔に実装
- ✓ ワーク1~ワーク5のリストを作成し、`for` ループで順番に処理
- ✓ ワークを機械へ移動 → 3秒待機 → 元の位置へ戻す を繰り返す

期待される動作

1. ワーク1を機械へ → 3秒待機 → 元の位置へ戻す
2. ワーク2を機械へ → 3秒待機 → 元の位置へ戻す
3. ワーク3を機械へ → 3秒待機 → 元の位置へ戻す
4. ワーク4を機械へ → 3秒待機 → 元の位置へ戻す
5. ワーク5を機械へ → 3秒待機 → 元の位置へ戻す

🌀 これで、ワーク1からワーク5まで順番に機械に運び、3秒待機後に元の位置へ戻す動作ができます! 🌀

これを実行すると、ワークを元の位置に戻した後に次のワークを取る際、手先がワークに衝突しそうでした。ワークを置いた後に安全な位置に退避する動作を加えてもらえませんか?

なるほど!

ワークを元の位置に戻した後、そのまま次のワークを取りに行くと、グリッパの位置が低いため衝突のリスクがある ということです。

修正内容

1. ワークを置いた後に、安全な退避位置へ移動する処理を追加
 - `place_workpiece()` の最後に 50mm 上に移動する処理 を追加
 - これで、次のワークを取りに行く際に安全な高さから開始できる

修正後のコード

```

import time

def pick_up_workpiece(workpiece):
# ワークの把持位置を取得
position = get_position(workpiece)

# グリッパを開く (衝突を防ぐため)
release()

```

```

# 安全な高さに移動
safe_position = [¥
    position[0], position[1],¥
    position[2] + 50, position[3],¥
    position[4], position[5]]
move_to(safe_position)

# 把持位置に移動
move_to(position)

# 把持
grab()

# 持ち上げる
move_to(safe_position)

def place_workpiece(target_location):
# 目標地点の配置位置を取得
position = ¥
    get_position(target_location)

# 安全な高さに移動
safe_position = [¥
    position[0], position[1],¥
    position[2] + 50, position[3],¥
    position[4], position[5]]
move_to(safe_position)

# 配置位置に移動
move_to(position)

# 解放
release()

# **安全な退避位置に移動**
move_to(safe_position)

def process_workpiece(workpiece, workplace):
# **ワークを把持**
pick_up_workpiece(workpiece)

# **機械に配置**
place_workpiece("machine")

# **3秒待機**
time.sleep(3)

# **ワークを再把持**

```

```

pick_up_workpiece("machine")

# **元のワーク置き場に戻す**
place_workpiece(workplace)

# **ワーク 1~ワーク 5 を順番に処理**
workpieces = [("work1", "workplace1"),
              ("work2", "workplace2"),
              ("work3", "workplace3"),
              ("work4", "workplace4"),
              ("work5", "workplace5")]

for workpiece, workplace in workpieces:
    process_workpiece(workpiece, ¥
                      workplace)

```

修正ポイント

- `place_workpiece()` の最後に退避動作を追加
 - ワークを置いた後、50mm 上方に移動して安全な高さを確保
- ワーク 1~ワーク 5 の処理順はそのまま

期待される動作

1. ワーク 1 を機械へ移動 → 3 秒待機 → 元の位置へ戻す → 退避
2. ワーク 2 を機械へ移動 → 3 秒待機 → 元の位置へ戻す → 退避
3. ワーク 3 を機械へ移動 → 3 秒待機 → 元の位置へ戻す → 退避
4. ワーク 4 を機械へ移動 → 3 秒待機 → 元の位置へ戻す → 退避
5. ワーク 5 を機械へ移動 → 3 秒待機 → 元の位置へ戻す → 退避

🚀 これで、ワークを置いた後に必ず安全な高さへ移動し、次のワークを取りに行く際の衝突リスクを防げます! 🌀

食品把持におけるロボットハンドの性能比較

神内 杜夫, 福本 靖彦, 竹中 慎, 小林 宏明

近年、食品工場や小売業、物流倉庫などにおいて、製品のハンドリングを自動化するために様々な種類のロボットハンドが開発されている。特に、食品のような柔らかく壊れやすい製品や不定形な製品を把持するためには、柔軟な材料で構成されたソフトグリッパと呼ばれるロボットハンドを用いることが多い。一方で、ロボットハンドごとに性能や特性が異なるため、対象製品に対して最適なロボットハンドを選定するためには費用や時間を要する。このため、本研究では、ロボットハンドの選定段階における判断材料を提案することを目的として、香川県産業技術センターが所有する市販ソフトグリッパ等のロボットハンドを使用し、様々な食品製品を対象に把持実験を行った。ワークの特性に応じたロボットハンドの適性について考察した内容を報告する。

1 緒言

近年、少子高齢化に伴い、労働力の確保が深刻な問題となっている^{1, 2)}。特に食品製造業では、他の製造業と比べて労働生産性や労働装備率が低く³⁾、ロボット技術の導入による自動化や省人化が期待されている。

食品工場や小売業、物流倉庫などの現場では、柔らかく壊れやすい物体や不定形な物体を扱うことが多く、食品の箱詰めやライン間のピッキング作業は、人手に頼るケースが多い。これらの作業をロボットで自動化する場合、高剛性なロボットハンドでは、対象物ごとに専用設計や制御条件の細かな調整をしなければ、把持対象とする食品（以下、食品ワークと呼ぶ）を適切に把持することが難しい。特に、少量多品種の食品ワークを扱う場合には、調整作業の負担が大きい。

このため、近年ではソフトグリッパと呼ばれる柔軟な材料で構成されたロボットハンドが注目されている。ソフトグリッパは、空気圧等を利用し、把持対象物の形状に沿って変形しながら把持するため、柔軟物や不定形物を安定的に把持しやすいという特長がある。しかし、ソフトグリッパの性能も把持対象物の形状や材質に依存するため、すべての食品ワークの把持ができるとは限らない。現在、多様なソフトグリッパが開発されているが⁴⁾、それぞれ異なる特性を持つため、食品ワークに応じたソフトグリッパを選定する必要がある。さらに、食品ワークによっては従来の高剛性なロボットハンドや吸着パッドなどの把持技術も有効な選択肢であり、最適なロボットハンドを選定するために

は費用や時間をかけた試行錯誤が必要である。

本研究では、ロボットハンドの選定段階における判断材料を提案することを目的として、香川県産業技術センターが所有する市販ソフトグリッパおよびソフトグリッパ以外の一般的なロボットハンドを使用し、様々な食品ワークの把持実験を行い、食品ワークの特性に応じた適切なロボットハンドの選定について考察した。

2 実験方法

2.1 ロボットハンド

把持実験に用いたロボットハンドを表1および図1に示す。なお、図1中の矢印はハンドが閉じる方向と吸着パッドの負圧の方向を示している。弾性フィンガによる把持手法は後述する。本研究では、香川県産業技術センターで所有する3種類のソフトグリッパに加え、食品業界に限らず工業製品でも汎用的が高いロボットハン

表1 使用したロボットハンド

製品名	メーカー名	駆動方式	接触面の材質
mGrip P2	SoftRobotics	空圧式（加圧）	シリコン樹脂
SOFTmatics (GR20QA-DKA)	ニッタ	空圧式（減圧）	ウレタン樹脂
弾性フィンガ (MH-X7654)	SMC	なし（手動）	シリコン樹脂
吸着パッド (SPB4-30)	シュマルツ	空圧式（減圧）	シリコン樹脂
電動2指グリッパ (2f-85)	ROBOTIQ	電動モーター	硬質ゴム

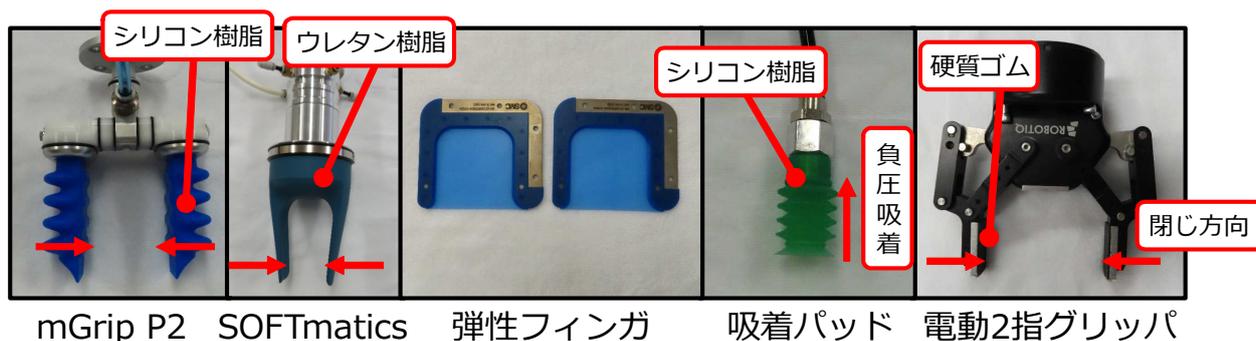


図1 使用したロボットハンド

ドとして、吸着パッドと電動2指グリッパの2種を加えた計5種類のロボットハンドを用いた。

本研究で用いたソフトグリッパの選定理由は次の通りである。まず、Soft Robotics社のソフトグリッパは世界的に高いシェアを誇り⁵⁾、ロボットメーカーにおいても取り扱いが増加している。また、ニッタ社の製品は国内メーカー製品の中では知名度が高く、多様なバリエーションを有するとともに、食品分野への応用実績も豊富である⁶⁾。他に、3指以上のグリッパや複数方向から包み込むようなグリッパも存在するが、本研究では最も基本的な2指による側面把持を対象としてグリッパを選定した。また、弾性フィンガは、ゴムシートの張力によって把持力を生み出す独自の機構⁷⁾を持ち、他の市販製品とは違った特性を有する可能性があるため採用した。

続いて、把持実験の方法については、給電が必要な電動2指グリッパを除き、ロボットに接続せず人がハ

ンド根元の固定部を手で持った状態で把持実験を行った。理由として、食品ワークの場合、柔軟物かつ不定形であり、把持に適するハンドの位置・姿勢やアプローチ方法等がワーク毎に異なることから、都度調整する手間を省くためである。なお、この方法は Chengらの先行研究⁸⁾を参考にしたものである。

また、弾性フィンガはシリコンゴムシートからなる2枚の板であり、エアチャックなどの平行開閉型の動作部に取り付けなければ使用できないが、本研究では動作部を用意できなかった。そこで、図2に示す治具を用いて、人が左右から挟み込む動作を行うこととした。この場合、フィンガの移動速度や平行度、把持力などが一定でないことから、実際のロボットハンドの動作とは一致しないが、弾性フィンガの特性評価という点では本手法でも十分対応できると考えた。残る電動2指グリッパについては協働ロボット (UR10, Universal Robot)に取り付けて把持することとした。

SOFTmatics および吸着パッドの動作には、真空エジェクタ (SCPMb10S04NCAAL, シュマルツ)を用いて0.30 MPaの圧縮空気を負圧に変換して動作を行った。発生する真空度はメーカー公称値でおよそ0.05 MPaである。mGripP2は専用コントローラから供給される圧縮空気を用いて動作を行った。

電動2指グリッパはハンドの閉じ量を徐々に狭めながら調整し、食品ワークを空中に持ち上げ可能になった位置で停止させた。本グリッパは力制御も可能であるが、適切な把持力を決定するには食品ワークごとに事前調査が必要となる。その過程でサンプルを損傷す

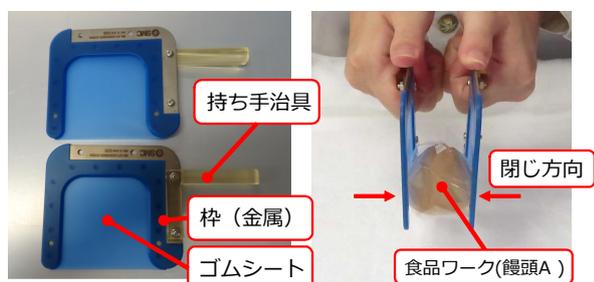


図2 弾性フィンガによる把持の様子

表2 実験に用いた食品ワーク

把持対象物	包装	寸法・形状	重量
ドーナツ	なし	直径約75 mmのドーナツ形	約60 g
どらやき	なし	直径約70 mmの円形	約60 g
バウムクーヘン	なし	直径約60 mmのドーナツ形	約35 g
饅頭A	ビニール個包装	直径約40 mmの円錐形	約25 g
饅頭B	ビニール個包装	幅40×50 mmのかまくら形	約22 g
砂糖菓子	ビニール個包装	80×35×15 mmの直方体	約60 g
ようかん	なし	幅20×40 mmの四角形	約3 g
マシュマロ	なし	直径25 mm~30 mmの球形	3 g~4 g
冷凍ハンバーグ	なし	直径約80 mmの円形	70 g~80 g
いりこ	なし	体長30 mm~35 mm	約0.2 g
ゼリー	プラスチック容器	直径55 mmのカップ	約100 g
パック食品A	真空パック	幅145×165 mmの平らなパック	約55 g
パック食品B	真空パック	幅130×170 mmの凹凸のあるパック	約180 g
ボトル飲料	PETボトル	直径約65 mmのボトル形	約575 g
紙パック飲料	紙パック	50×55×120 mmの直方体	約60 g



図3 実験に用いた食品ワーク



図4 mGripP2による把持痕の例

る可能性があるため、本実験では力制御を行わなかった。

2. 2 食品ワーク

把持実験に用いた食品ワークの概形寸法と重量を表2に、概観を図3に示す。把持実験に用いる食品ワークは、ロボットハンドの汎用性を評価するために、硬さや形状、表面特性などが異なるものを選定した。具体的には、冷凍食品などの硬さがある食品、焼菓子やマシュマロなどの柔らかく傷付きやすい食品、真空パックされた薄くて凹凸のある食品パッケージなど、幅広い食品ワークを選定した。また、香川県内の食品メーカーの製品を中心に選定した。これは、地域特産品の把持性能を評価することで、今後の県内食品メーカーの自動化支援に役立てるためである。

実験は室温で行い、各食品ワークの最終の製品状態におけるハンドリングを想定した。食品ワークの把持位置や持ち方は、側面把持もしくは吸着を基本として、持ち上げ可能な把持条件を試行錯誤により調整した。側面把持は図3の左右方向から把持を行った。また、食品ワークはテーブル上に平置きされているものとし、食品ワーク単体で自立しない初期姿勢は考慮しなかった。いりこのみ複数個をバラ積み状態で配置し、複数個をまとめて把持できるか検証した。

3 評価方法

食品ワークを把持した状態で空中に持ち上げ、グリップを揺らしたり傾けたりして落下の有無を確認した。電動2指グリップは空中に持ち上げた後、落下と滑りの有無を確認した。その後、ハンドとの接触面を観察し、傷や痕の有無を評価した。

本研究では、把持実験の結果を「把持可能」、「制約付き把持可能」、「把持不可」の3段階で評価した。食品ワークを落とすことなく把持でき、さらに把持後に図4に示すような、人が目視で確認できる程度の傷や痕が付いていなければ「把持可能」とした。例えば、図4はmGripP2を用いて饅頭Aを把持した際に生じた把持痕を示している。持ち上げることができない場合やすぐに落としてしまう場合、ロボットハンドとの接触により図4のような傷や痕が付いた場合は「把持不可」とした。持ち上げられるものの、滑りが見られる場合や、吸着パッドが密着しにくく吸着が不安定な場合は、「制約付き把持可能」とした。

上記の3つの区分以外の評価として、SOFTmaticsはハンドの仕様により60 mm以上開くことができないため、寸法が60 mm以上の食品ワークは把持対象外として評価した。弾性フィンガは、ハンド中央のゴムシート部分で把持することによりその特性が発揮されるため、金属の杵部分との接触が避けられないパック食品A、Bは把持対象外として評価した。また、2指グリップも最大開き幅の制約により、パック食品A、Bを把持対象外とした。

4 結果と考察

本章では、本研究で行った把持実験の結果を示し、各食品ワークに適したロボットハンドについて考察する。4.1節ではロボットハンドに着目した実験結果を、4.2節ではロボットハンドの違いによって、特に結果が異なった食品ワークに着目して、その結果を考察する。

表3に把持実験の結果を示す。丸記号は「把持可能」

表3 5種類のロボットハンドによる把持実験の結果

把持対象物	mGrip P2	SOFTmatics (GR20QA-DKA)	弾性フィンガ (MH-X7654)	吸着パッド	電動2指グリップ (2f-85)
ドーナツ	○	—	△ 滑りやすい	× 吸着不可	○
どらやき	△ 滑りやすい	—	△ 滑りやすい	× 破壊	× ハンドの痕が残る
パウムクーヘン	○	—	○	× 吸着痕が残る	× ハンドの痕が残る
饅頭A	× ハンドの痕が残る	○	○	× 吸着不可	× ハンドの痕が残る
饅頭B	○	△ 滑りやすい	○	× 吸着痕が残る	○
砂糖菓子	○	○	○	△ 吸着が不安定	○
ようかん	○	○	○	○	○
マシュマロ	× ハンドの痕が残る	× ハンドの痕が残る	○	× 吸着痕が残る	× ハンドの痕が残る
冷凍ハンバーグ	○	○	○	○	○
いりこ	○	○	× 持ち上げ不可	× 吸着不可	× 破壊
ゼリー	○	○	○	○	○
パック食品A	× 持ち上げ不可	—	—	○	—
パック食品B	× 持ち上げ不可	—	—	△ 吸着が不安定	—
ボトル飲料	○	—	○	○	○
紙パック飲料	○	—	○	○	○

○：把持可能
△：制約付き把持可能
×：把持不可
—：仕様上把持対象外

を示し、三角記号は「制約付き把持可能」を、バツ記号は「把持不可」を示す。「制約付き把持可能」および「把持不可」の場合、その理由を下段に示す。ハイフン記号は食品ワークのサイズとハンドサイズの不適合により把持対象外であることを示す。表3から、今回使用したロボットハンドの中に、すべての食品ワークを把持可能なものは存在しないことが分かる。

4. 1 ロボットハンドに着目した実験結果

mGripP2は3分の2以上の食品ワークを安定して把持できており、汎用性に優れることが分かる。SOFTmaticsはサイズの制約により把持対象外となった食品ワークを除くとmGripP2と類似した結果を示した。ハンド先端が曲面になっているため、饅頭Aなど一部の円形ワークの把持においては、SOFTmaticsの方が優位であると考えられる。

弾性フィンガは今回の実験において唯一、傷やハンドの痕を付けることなく把持が可能であり、マシュマロのような極めて柔らかい食品ワークでも安定して把持できた。一方で、いりこのように厚みがなく小さい食品ワークは把持が難しく、さらに、いりこは硬く尖った部分を有するため、ハンドが損傷する可能性があることから、把持には適していないと考えられる。

吸着パッドは食品に直接触れて把持する場合、傷や痕が付きやすく適切ではない。一方で、パック食品A、Bのような、薄く扁平な形状で側面把持に適さない食品ワークに対しては、吸着パッドでの把持が有効であ

ることが分かった。ただし、パック食品Bは食品形状に由来するフィルムのうねりがあるため、吸着位置によっては負圧が漏れることで持ち上げた後に落下するなど、吸着が不安定であった。このような吸着が不安定と評価された食品ワークに対しては、パックやパウチなどの把持に特化したより密着性の高い専用パッドへの変更や負圧の増加によって改善される可能性がある。

電動2指グリップは高剛性なロボットハンドであるにもかかわらず、多くの食品ワークを把持することが出来た。これは、人が目視により閉じ量を制御していたためであると考えられる。しかし、実際に個体差の大きい食品ワークや不定形な食品ワークを把持する場合には、ワークごとに閉じ量を個別に調整する必要があり、作業の手間が増える。力制御を導入すれば、対象の食品ワークに適した把持力を一度設定するだけで、個体差による微調整を省力化できる。ただし、異なる種類の食品ワークに対応するには、それぞれに適した把持力を決定する必要がある。また、ソフトグリップのようにハンドが変形しないため、より細やかな把持力の設定が求められると考えられる。

砂糖菓子や冷凍ハンバーグなど、硬い食品ワークは、直接触れて把持する場合でもほとんどのハンドで問題なく把持できた。同様に、ゼリー、ボトル飲料、紙パック飲料などの比較的硬い包装容器に入った食品ワークも、どのハンドでも把持可能であった。

4. 2 ハンドによって結果が異なった食品ワーク

ドーナツやどらやき、バウムクーヘンといった菓子類を直接把持した場合、滑りや把持痕が発生しやすく、把持可能なハンドが限られた。特にどらやきは柔らかい生地が2枚重なった構造であるため、側面把持が難しく、すべてのハンドで安定した把持ができなかった。

ビニールの個包装が施された饅頭Aを把持した際には、ビニールが押し付けられたことで痕が残ることがあった。一方、饅頭Bは表面性状の違いにより、ビニールの痕が付くことは無かった。ようかんも個包装されているが、食品とフィルムが隙間なく密着して包装されているため、把持痕が付きにくかった。

いりこは硬さがあるものの、小さくて脆いため把持できるハンドが限られた。特に、複数個をまとめて把持するのはどのハンドでも難しく、このような場合には下からすくい上げるような把持手法が適していると考えられる。

4. 3 まとめと考察

4. 1節、4. 2節の内容をまとめると、食品の把持に最適なロボットハンドを選定する場合、まずは汎用性の高いmGripP2やSOFTmaticsといったソフトグリップを試してみると把持に成功する可能性が高いと考えられる。食品ワークが非常に柔らかい場合には弾性フィンガが適しており、どらやきのような側面把持が難しい食品ワークには、今回の実験では用いなかった、下からすくい上げるような方式や3指以上のハンドで包み込むような方式が適している可能性が高い。真空パックされた食品ワークのような側面把持が難しい食品ワークには吸着パッドが適している。プラスチック容器などの硬さのある容器に包装されている場合や、冷凍食品など硬さのある食品ワークの場合には従来の高剛性な2指グリップも選択肢となり得る。

なお、本来であれば、不定形物への適正を評価するうえで、サンプルを多数用意して実験を行い、成功率などを加味して特性を評価するべきであるが、本研究では、都合上サンプル数を確保出来なかった。そのため、不定形物への適正については、追加の検証が必要と考えている。

5 結言

本研究では香川県産業技術センターが所有するソフトグリップおよびソフトグリップ以外の一般的なロボットハンドを用いて、様々な食品ワークを対象に把持

実験を行った。実験の結果、全ての食品ワークを把持可能なハンドは無かったものの、特に柔らかい食品ワークや薄く側面把持の難しい食品ワークを除けば、ソフトな2指グリップが比較的汎用性に優れていた。また、弾性フィンガは柔らかい食品ワークの把持に適しており、包装された側面把持が難しい食品ワークには吸着パッドが有効であることが分かった。さらに、冷凍食品や硬い包装容器に入った食品ワークについては、従来の高剛性なハンドでも十分な把持が可能であることが確認された。

今後の課題として、側面把持以外の方式を採用したロボットハンドや3指以上のロボットハンドの把持性能の検証、高剛性なロボットハンドにおける力制御時の把持性能の評価について検討していきたい。また、本研究で得られた成果をもとに、県内企業の生産現場のニーズにあわせたロボットハンド選定の技術支援などに活用していきたい。

参考文献

- 1) 香川県 政策部 政策課：「人生100年時代のフロンティア県・香川」実現計画(2023)。
- 2) 経済産業省：2024年版ものづくり白書(2024)。
- 3) 農林水産省 新事業・食品産業部食品製造課：食品製造業をめぐる情勢(2025)。
- 4) 平井慎一, 王忠奎：ソフトロボットによる食品ハンドリング, 日本ロボット学会誌, Vol. 37, No. 6, pp. 489~494 (2019)。
- 5) QY Research: グローバルソフトロボットグリップー市場の産業全体規模とシェア分析 2024-2030, <https://www.qyresearch.co.jp/news/6655/about-us> (検索日: 2025. 3. 20)。
- 6) ニッタ: ソフトハンドリング用ロボットハンド SOFTmaticsTM ソフマティックスTM, <https://www.nitta.co.jp/product/robohand> (検索日: 2025. 3. 20)。
- 7) SMC 株式会社: ワーク保持具及びチャック装置, 実用新案登録第3240286号, 令和4年12月13日。
- 8) Cheng Chi, Zhenjia Xu, Chuer Pan, Eric Cousineau, Benjamin Burchfiel, Siyuan Feng, Russ Tedrake, Shuran Song: Universal Manipulation Interface: In-The-Wild Robot Teaching Without In-The-Wild Robots, Robotics: Science and Systems (RSS) 2024.

タレットパンチプレス加工品の仕上げ加工自動化に関する検討

福本 靖彦, 神内 杜夫, 竹中 慎

タレットパンチプレス加工において製品と母材がワイヤージョイントにより繋がっている場合、ジョイント部を切断して製品を取り外したうえで、更に製品側に残ったジョイントの突起を除去する仕上げ加工が必要となる。本研究ではこの仕上げ加工を自動化するための検討を行い、試作システムを構築した。またジョイント部の仕上げ加工実験により、十分な加工品質が得られることを確認した。

1 緒言

各種製品の板金筐体の製造工程では、タレットパンチプレスを用いて板材の打ち抜きや成型を行う加工方法がある。その際、母材の板材と製品を繋いでいるワイヤージョイントを切断して製品（以降、ワークと呼ぶ）を取り外し、更にそのワーク側に残ったジョイント部（以降、「ワーク側に残った」を省略し、単にジョイント部と呼ぶ）の突起を手作業により除去する仕上げ加工を行うことがある。

本研究ではこの仕上げ加工をロボットにより自動化するための検討を行い、試作システムを構築した（図1）。仕上げ加工実験を行った結果、想定する製品として問題のない範囲でジョイント部の突起を除去できることを確認した。

2 仕上げ加工システム

2.1 試作システムの開発

本研究で対象とする仕上げ加工に似た作業としてバリ取り作業がある。バリ取り作業の自動化においては、ロボットの手先に回転工具等を取り付けて環境に固定したワークに押し当てる場合^{1~3)}と、逆に、回転工具等を環境に固定してロボットがワークを把持して回転工具に押し当てる場合^{3~5)}がある。後者の場合は、未加工ワークをストックから取り出す作業や、加工済みワークをストックに戻す作業も一つのロボットで行うことができるため、一連の作業を自動化するのに効率的である。したがって本研究ではロボットはワークを把持し、回転工具は作業環境に設置することとした。

ロボットで板状のワークを把持する場合のエンドエフェクタには、吸着ハンドが用いられることが多く、本研究においても吸着ハンドを用いた。ただし、吸着パッド部は、柔軟なゴム材料で構成されるため、ワークの把持剛性が低い。このため、ワークを回転工具に押し当てると、ジョイントの除去加工中の外力によりワークの回転運動が生じてしまい、適切な加工ができない（図2）。特に加工の送り速度を上げると問題が顕著となり、サイクルタイムを上げられないことも課題となる。

そこで本研究では、把持したワークを外部環境に押し当てて外部環境による物理的拘束を加えることで、ワークの回転運動に関する問題に対処した。具体的には、まず図3のような電動スライダと回転工具等からなる試作加工機を用意した。そしてロボットがワークを二つの位置決めピンに押し当てた状態で回転工具を装着した

電動スライダを直線運動させることでジョイント部の突起を除去する。以上の方法を用いると、ロボットがワークを設置する際に若干の位置誤差があっても、位置決めピンにワークを押し付けることで、ワークの位置が意図した位置に定まることが期待できる。これは吸着ハンドの柔軟性に起因するものであり、柔軟性を積極的に活用することで逆にメリットを生みだしている。

以上の検討により、図1に示す試作システムを構築した。また、主要な構成部品を表1に示す。

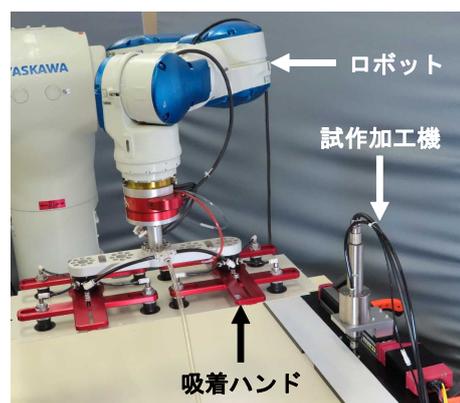


図1 試作した仕上げ加工システム

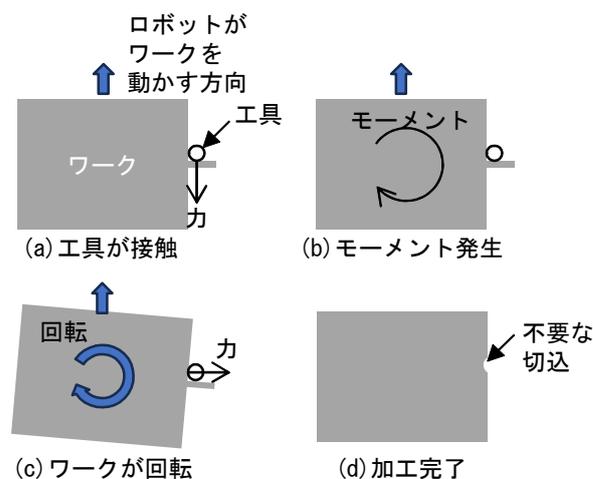


図2 ワーク把持部の剛性が不十分であると、工具がジョイント部を押す(a)際にワークに発生するモーメント(b)によりワークが無視できない程度に回転し(c)、ジョイント部の手前を切り込んでしまう(c, d)。

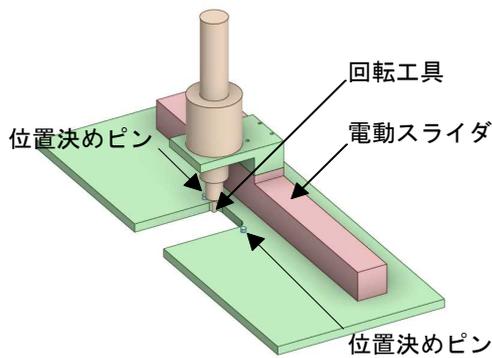


図3 試作加工機

表1 試作システムの主要な構成部品

構成部品	型番 (メーカー)
ロボット	MOTOMAN-SDA10F (安川電機)
ハンド	ロボットハンドキット (コンバム)
吸着パッド	PKMG-30-W (コンバム)
スピンドル	NR3060-AQC, EM-3060 (ナカニシ)
工具	BC2013 (モニター)
電動スライダ	EC-DS3M-200-0-MOR (IAI)

2. 2 仕上げ加工実験

本試作システムにより仕上げ加工実験を行った。加工サンプルは板厚 0.5 mm のスチール製の板材で、ジョイントの幅は 1 mm のものを用いた。回転工具の回転速度は 30000 rpm とし、電動スライダの移動速度は 20 mm/s に設定した。

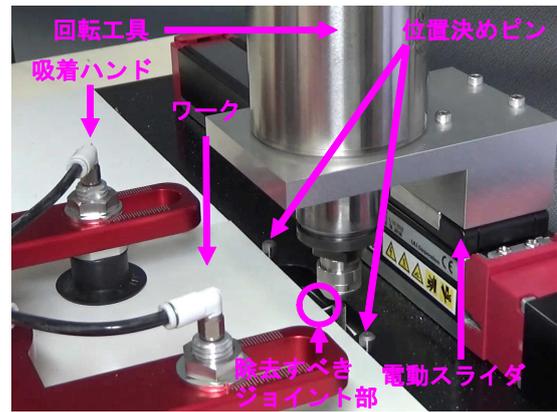
試作システムによる加工の様子を図4に示す。加工の結果を図5に示す。ジョイントの痕跡は残るものの、想定する製品として問題のない範囲でジョイント部の除去を行えたことが確認できた。

2. 3 試作システムの現場実装に向けての検討

本試作システムでは垂直多関節型のロボットを利用したが、ロボットの動作は所定の位置のワークを持ち上げ、水平搬送し、回転工具を装着した加工機に置く（押し当てる）のみである。一つのワークに複数の加工箇所があることが想定されるので、ワークの鉛直軸周りの姿勢を変えることは想定されるが、それ以外の軸周りの姿勢の変化は必要ないと考えられる。したがってより安価なスカラ型のロボットを用いることでコストを抑えることができる。

また図3の試作加工機を用いる場合、吸着ハンドとスピンドルの干渉が起らないように注意して設計を行う必要がある。この課題については、加工機のスピンドルの取り付け方向を上向きに変更することで解消されると考えられる。

最後に本加工システムは、ワークを位置決めピンにより適切に位置決めできることが前提条件である。例えばジョイント周辺のワーク形状が自由曲線である等、複雑な形状である場合には、位置決めピンの位置を工夫する必要がある。



(a) ワークを位置決めピンに押し当て



(b) 工具を回転し電動スライダを移動



(c) 加工完了

図4 加工実験の様子

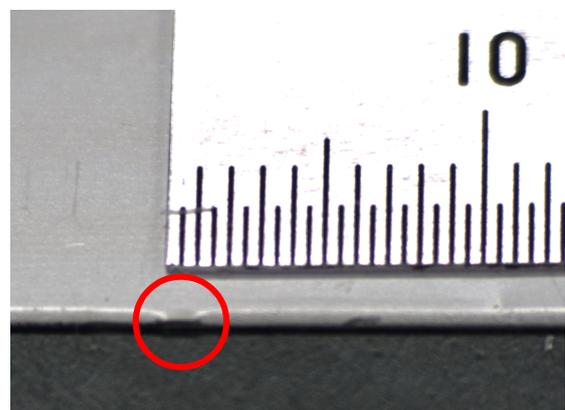


図5 加工後のワーク。赤丸で囲った箇所がジョイント部を除去した箇所である。

3 結言

本研究ではタレットパンチプレス加工品のワイヤージョイント部の除去を行う仕上げ加工を自動化するための検討を行った。検討の結果、吸着ハンドを備えたロボットでワークを搬送し、回転工具とスライダを組み合わせた加工機により仕上げ加工を行うシステムを試作した。また試作システムによる加工実験を行い、ジョイント部の突起を想定する製品として問題のない範囲で除去できることを確認した。

今後の課題として、スピンドルを上向きに変更することで多品種対応をやすくすることや、母材から製品を切断する工程を含めた自動化の検討が挙げられる。

参考文献

- 1) スギノマシン：ロボットバリ取りセル RDM-S, <https://www.sugino.com/products/deburring-tool/rdm-s/> (検索日：2025. 1. 17)
- 2) ファインシステム：小型バリ取りロボットセル FDM-002 型 熟練工のバリ取りレベルを自動化にて実現！, <https://www.fine-techno.co.jp/products/finesystem/pl339/> (検索日：2025. 1. 17)
- 3) 炭崎竜平, 海津雅洋, 堀健一, 寺田大祐, 増田芳樹：FA アプリケーションパッケージ “iQ Monozukuri,” 三菱電機技報, Vol. 91, No. 4, pp. 7(209)-10(212) (2017).
- 4) ヤマハファインテック：小型金属部品用バリ取り・面取りシステム, <https://www.yamahafinotech.co.jp/products/robots/ws056/> (検索日：2025. 1. 17)
- 5) MiSUMi：MiBOT[ミボット]バリ取りロボットパッケージ, https://jp.misumi-ec.com/special/robot/inquiry_semi_order/semi031/ (検索日：2025. 1. 17)

移動ロボット等によるけん引軌道の計算 —軌道のシミュレーション方法—

福本 靖彦

自律移動ロボットや無人搬送車などがカート等をけん引して走行するとき、カート等の軌道は自律移動ロボット等の軌道と異なるので、軌道のずれを考慮した運用を行う必要がある。そこで本稿では、けん引されたカートがどのような軌道を描くかをシミュレーションする方法を検討した。数値解のほか、直線や円弧の繋ぎ合わせの場合には解析解も計算できたので計算例を示す。

1 緒言

自律移動ロボットや無人搬送車 (AGV) 等 (以降、単にロボットと記す) を用いて工場内の物品の搬送自動化を行う際、その搬送方法はいくつかの方法があるが、本研究では車輪のついたカート等 (以降、単にカートと記す) をけん引して搬送する場合を扱う (図1)。この方法は、場合によってはロボット自身がカートの脱着まで可能である点、既存のカートを流用できる点、ロボットに搬送物の重量が直接作用しないのでロボットの可搬重量以上の重さの物品を運べる点、カートの下にロボットの潜り込むスペースが必要ない点、搬送物がコンベア等に載っていないかでもよい点がメリットである。一方、通路の幅が狭い場合にはロボットは通れたとしてもカートが壁に衝突してしまうなどの問題が生じる場合がある。

本研究では、ロボットがカートをけん引するときのカートの軌道を計算する方法を検討した。検討の結果、カートの軌道を数値的に計算する方法が得られたので説明する。また、ロボットの軌道が直線や円弧の繋ぎ合わせで得られている場合には、カートの軌道の解析解が得られたので、その計算例を示す。

本稿では、2節でロボットによるけん引の問題設定を行うとともに移動ロボットやカートの運動の基礎的事項を説明する。3節でカートの軌道の計算を行うための計算式を示す。4節でシミュレーションを行う。5節でまとめを述べる。

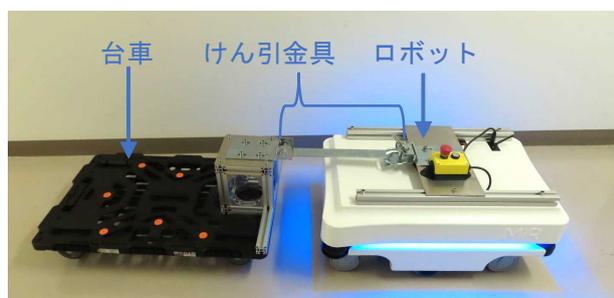


図1 ロボットによる平台車けん引システムの試作例。ロボットはMiR100 (Mobile Industrial Robots), けん引金具はJB-702B および JB702C (スペース), 台車はBT900J3-E100 (トラスコ中山) を利用。

2 けん引による搬送

本節では、ロボットによるけん引の問題設定と基礎的事項について説明する。

2.1 問題設定

ロボットは全方位に移動可能なものも、前後進と旋回が可能なものも扱う。前者はメカナムホイールなどを用いたものが該当し、後者は二輪速度差方式や前輪ステアリング方式のものが該当する。後者は前者の特殊ケースとして扱う。カートは二つの固定車を有するものとする。ロボットとカートは回転対偶で連結されているものとする。

ロボットやカートの運動は図2のように二次元で考える。ロボットの基準点は二つの駆動輪の中心や後輪の中心等とし、カートの基準点は二つの固定輪の中心とする。対偶の位置は、ロボットの基準点からロボットの後方にながった位置とする。また、カート側から見ると、カートの基準点から前方に進んだ位置とする。本稿で用いる文字を以下の通り定義する。

$p_1, p_p, p_2 \in R^2$: ロボット, 回転対偶, カートの位置
 $p_{x1}, p_{y1}, p_{x2}, p_{y2} \in R$: p_1, p_c, p_2 の X 成分と Y 成分
 $v_1, v_p, v_2 \in R^2$: ロボット, 回転対偶, カートの速度
 $V_1, V_2 \in R$: ロボットとカートの速度の直進方向成分
 $\theta_1, \theta_2 \in R$: ロボットとカートの姿勢
 $\omega_1, \omega_2 \in R$: ロボットとカートの姿勢の時間微分
 $r_1, r_2 \in R$: ロボットとカートの位置軌道の回転半径
 $l_1 \geq 0$: ロボットの基準点から回転対偶までの距離
 $l_2 > 0$: カートの基準点から回転対偶までの距離
 $h > 0$: シミュレーションの時刻み幅

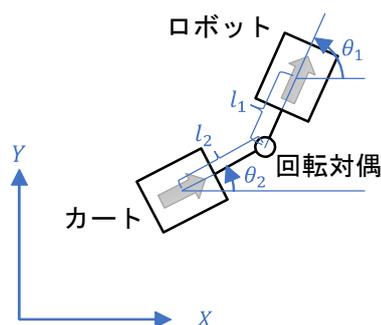


図2 けん引システムの構成

なお、距離の単位は[mm]を用いる。姿勢の角速度は左回りの時に正で、右回りの時に負とする。姿勢の角速度を単に角速度と呼ぶ。また、本研究ではある時刻からある時刻までの位置、姿勢、速度、角速度のデータの集合をそれぞれ位置の軌道、姿勢の軌道、速度の軌道、角速度の軌道と呼ぶ。

最後に、注意点を記す。ロボットでけん引を行うとき、ロボットがけん引されるカートや連結部を障害物として検知してしまう可能性があるため、センサーの位置や検出距離に注意してけん引部を設計する必要がある。また、走行のたびにロボットは自動で経路生成するため、経路の再現性が低くなるので、一度テストした際にカートがうまく通れても、次回以降も同様にうまく通れるとは限らない。これらについては、経路生成時のロボット自身の大きさの設定値（フットプリント）を大きく登録することで経路生成時に余裕を持たせたり、地図の走行禁止領域をうまく設定したり、経路途中に通るべきウェイポイントをうまく設定したり、後退動作は障害物のない広い場所で行うような運用にしたり、ロボットが見えないところは先にロボットが見てマップ上に障害物を認識してから通行したりするなど対処できる場合がある。

2. 2 ロボットの運動の基礎的事項

移動ロボットは、その駆動輪の回転速度やステアリング角度を設定すると、それに応じてロボット全体の速度 v_1 や（姿勢の）角速度 ω_1 が発生する。なお、二輪速度差方式や前輪ステアリング方式のロボットは前後進の方向にしか速度を発生させることができないので、その運動は $V_1 (\in \mathbb{R})$ と ω_1 で規定することができる。

$$v_1 = V_1 \begin{bmatrix} \cos \theta_1 \\ \sin \theta_1 \end{bmatrix} \quad (1)$$

と書くことができる。

二輪速度差方式や前輪ステアリング方式のロボットは基本的に円軌道を描く。ロボットの両輪に速度差があり、どちらの車輪の回転速度も一定値であり変化しない場合や、ステアリングを一定の角度に固定して走行する場合を考えると、ロボットが円軌道を描くことは容易に想像できると思われる。車輪の速度差やステアリング角が一定値ではなく動的に変化する場合も、一瞬一瞬を切り取れば、同様にある回転半径の円軌道を描いている。また、直進動作も可能であるが、これは回転半径が無限大の場合として扱うことができる。

ロボットの回転半径は

$$r_1 = \frac{V_1}{\omega_1} \quad (2)$$

で計算できる。本稿では、回転半径が正の場合はロボットが左回り、負の場合は右回りであるとして、負の回転半径も扱う。

2. 3 カートの運動の基礎的事項

固定二輪方式のカートの運動も二輪速度差方式や前輪ステアリング方式のロボットと同様に前後進と旋回だけが可能であり、 V_2 と ω_2 で規定することができる。 v_2 は

$$v_2 = V_2 \begin{bmatrix} \cos \theta_2 \\ \sin \theta_2 \end{bmatrix} \quad (3)$$

となる。

ロボットとの差は、ロボットは自身が車輪を能動的に

動かすのに対して、固定二輪方式のカートは外部からの力を受けて受動的に車輪が動くという点である。

なお本研究では扱わないが、一般には自在車を四つ備え、固定車を持たないカートも多く存在する。この場合、カートは回転対偶が動ける方向に自由度が残るため、ロボットが曲がるとカートは遠心力で大きく振られてしまう。これを防ぐにはロボットとカートを回転対偶で接続するのではなく、相対位置が変わらないように完全に固定する必要があるが、ロボットの回転半径が小さい場合にカートの横方向の速度が高くなることや、重量物の搬送時にロボットがカートに振り回されやすくなることに注意が必要である。逆にこの時、ロボットのフットプリントを広げて登録しておくだけで、ロボットの衝突回避機能で、ある程度衝突回避を実現できるという利点もある。

3 けん引時のカートの位置と姿勢の軌道の計算方法

本節では、ロボットがカートをけん引して走行したときのカートの位置と姿勢の軌道を計算する方法を説明する。

ロボットの運動の情報から計算した回転対偶部の速度と、カートの運動の情報から計算した回転対偶部の速度が一致するという関係式により、けん引時のカートの運動を考えることができる。なお、回転対偶であるので対偶の姿勢についての制約は生じず、位置の関係式だけとなる。また二輪速度差方式や前輪ステアリング方式の場合、ロボットの基準点では、ロボットの正面方向（および背面方向）にしか速度は生じないが、対偶部がロボットの基準点から離れていれば（ $l_1 \neq 0$ の場合）、対偶部においてはロボットの旋回動作により生じる横方向の速度も合成され、正面方向以外の速度も生じうる。

3. 1 基本式の導出

ロボットの情報を用いて v_p を計算すると、

$$v_p = v_1 - l_1 \omega_1 \begin{bmatrix} \cos \left(\theta_1 - \frac{\pi}{2} \right) \\ \sin \left(\theta_1 - \frac{\pi}{2} \right) \end{bmatrix} \quad (4)$$

であり、カートの情報を用いて v_p を計算すると、

$$v_p = v_2 + l_2 \omega_2 \begin{bmatrix} \cos \left(\theta_2 + \frac{\pi}{2} \right) \\ \sin \left(\theta_2 + \frac{\pi}{2} \right) \end{bmatrix} \quad (5)$$

である。式(3)、(4)、(5)から v_p と v_2 を消去して式変形すると

$$\begin{bmatrix} V_2 \\ \omega_2 \end{bmatrix} = \begin{bmatrix} v_{1x} \cos \theta_2 + v_{1y} \sin \theta_2 + l_1 \omega_1 \sin(\theta_1 - \theta_2) \\ -\frac{v_{1x}}{l_2} \sin \theta_2 + \frac{v_{1y}}{l_2} \cos \theta_2 - \frac{l_1 \omega_1}{l_2} \cos(\theta_1 - \theta_2) \end{bmatrix} \quad (6)$$

である。よって、

$$\begin{bmatrix} \dot{p}_{x2} \\ \dot{p}_{y2} \\ \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} (v_{1x} \cos \theta_2 + v_{1y} \sin \theta_2 + l_1 \omega_1 \sin(\theta_1 - \theta_2)) \cos \theta_2 \\ (v_{1x} \cos \theta_2 + v_{1y} \sin \theta_2 + l_1 \omega_1 \sin(\theta_1 - \theta_2)) \sin \theta_2 \\ -\frac{v_{1x}}{l_2} \sin \theta_2 + \frac{v_{1y}}{l_2} \cos \theta_2 - \frac{l_1 \omega_1}{l_2} \cos(\theta_1 - \theta_2) \end{bmatrix} \quad (7)$$

が得られる。これを解けばカートの位置と姿勢の軌道が得られる。

ここで、ロボットが直進方向にしか速度を発生することができない場合は、

$$\begin{bmatrix} V_2 \\ \omega_2 \end{bmatrix} = \begin{bmatrix} l_1 \omega_1 \sin(\theta_1 - \theta_2) + V_1 \cos(\theta_1 - \theta_2) \\ \frac{V_1}{l_2} \sin(\theta_1 - \theta_2) - \frac{l_1 \omega_1}{l_2} \cos(\theta_1 - \theta_2) \end{bmatrix} \quad (8)$$

であり、

$$\begin{bmatrix} \dot{p}_{x2} \\ \dot{p}_{y2} \\ \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} (l_1 \omega_1 \sin(\theta_1 - \theta_2) + V_1 \cos(\theta_1 - \theta_2)) \cos \theta_2 \\ (l_1 \omega_1 \sin(\theta_1 - \theta_2) + V_1 \cos(\theta_1 - \theta_2)) \sin \theta_2 \\ \frac{V_1}{l_2} \sin(\theta_1 - \theta_2) - \frac{l_1 \omega_1}{l_2} \cos(\theta_1 - \theta_2) \end{bmatrix} \quad (9)$$

が得られる。なお、以上の解析結果は、対偶の可動範囲 (α の値の取り得る範囲) を考慮していないことに注意が必要である。

なお、数値解に誤差がなければ

$$\|p_1 - p_p\| = l_1, \forall t \quad (10)$$

$$\|p_p - p_2\| = l_2, \forall t \quad (11)$$

である。したがって、 $\|p_1 - p_p\|$, $\|p_p - p_2\|$ が l_1, l_2 に一致していない場合は計算の誤差が存在する。これを誤差の有無を判断するための判断材料の一つとすることもできるが、誤差が存在してもこれらの値が一致する可能性もあるので過信はできない。

3. 2 ロボットの速度と角速度の軌道が既知の場合

本項ではロボットの速度 v_1 と角速度 ω_1 が既知である場合に、カートの位置と姿勢の軌道を計算する方法を示す。

3. 2. 1 運動の定式化

カートの位置 p_2 と姿勢 θ_2 の初期値と、各時刻のロボットの姿勢 θ_1 が分かれば式(7)や式(9) 下半分の成分を積分することでカートの位置と姿勢を計算することができる。

ここで、ロボットの運動を表す式は

$$\begin{bmatrix} \dot{p}_{x1} \\ \dot{p}_{y1} \\ \dot{\theta}_1 \end{bmatrix} = \begin{bmatrix} v_{1x} \\ v_{1y} \\ \omega_1 \end{bmatrix} \quad (12)$$

である。本項ではロボットの速度 v_1 と角速度 ω_1 は既知としているので、あとは p_1 と θ_1 の初期値が分かれば、これを積分することで各時刻の (p_1 と) θ_1 を得ることができる。

式(7)あるいは(9)と式(12)をまとめると

$$\begin{bmatrix} \dot{p}_{x1} \\ \dot{p}_{y1} \\ \dot{\theta}_1 \\ \dot{p}_{x2} \\ \dot{p}_{y2} \\ \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} v_{1x} \\ v_{1y} \\ \omega_1 \\ (v_{1x} \cos \theta_2 + v_{1y} \sin \theta_2 + l_1 \omega_1 \sin(\theta_1 - \theta_2)) \cos \theta_2 \\ (v_{1x} \cos \theta_2 + v_{1y} \sin \theta_2 + l_1 \omega_1 \sin(\theta_1 - \theta_2)) \sin \theta_2 \\ -\frac{v_{1x}}{l_2} \sin \theta_2 + \frac{v_{1y}}{l_2} \cos \theta_2 - \frac{l_1 \omega_1}{l_2} \cos(\theta_1 - \theta_2) \end{bmatrix} \quad (13)$$

や

$$\begin{bmatrix} \dot{p}_{x1} \\ \dot{p}_{y1} \\ \dot{\theta}_1 \\ \dot{p}_{x2} \\ \dot{p}_{y2} \\ \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} V_1 \cos \theta_1 \\ V_1 \sin \theta_1 \\ \omega_1 \\ (l_1 \omega_1 \sin(\theta_1 - \theta_2) + V_1 \cos(\theta_1 - \theta_2)) \cos \theta_2 \\ (l_1 \omega_1 \sin(\theta_1 - \theta_2) + V_1 \cos(\theta_1 - \theta_2)) \sin \theta_2 \\ \frac{V_1}{l_2} \sin(\theta_1 - \theta_2) - \frac{l_1 \omega_1}{l_2} \cos(\theta_1 - \theta_2) \end{bmatrix} \quad (14)$$

となる。これを解けば (積分すれば) 良い。

3. 2. 2 1階の微分方程式の解法

式(13), (14)を解くための一般的な方法として、ルンゲ・クッタ法と解析的な手法を簡単に紹介する。

3. 2. 2. 1 ルンゲ・クッタ法

微分方程式の数値解を得るための最も単純な方法としては前進オイラー法が知られているが、これは実装が容易である一方、精度が悪いことが知られている。そこで、前進オイラー法よりも精度がよく、しかし計算量の増加は比較的小さい手法である4次のルンゲ・クッタ法がよく用いられる。例えば、Pythonで式(13)や(14)を数値的に解く場合、SciPyライブラリの `solve_ivp()` 関数が利用できるが、`solve_ivp()` 関数のデフォルトの解法は4次のルンゲ・クッタ法である。

なお `solve_ivp()` では誤差を制御するパラメータがあるので、デフォルトでは誤差が大きくなっていると思われる場合は `atol` や `rtol` を調整するとうまくいく場合がある。また、SciPyの公式ドキュメント¹⁾では `atol` や `rtol` を小さくして高精度な計算を行いたい場合は8次のルンゲ・クッタ法を推奨すると書かれているので、必要に応じて検討されたい。このほか、状態の値が急激に変わるようなシステムの場合はまた別の手法が適していると書かれている。詳細は公式ドキュメントを参照されたい。

3. 2. 2. 2 解析的な手法

v_1 と ω_1 が解析的な形で与えられている場合、解析解を計算できる場合がある。複雑な計算が必要な場合はソフトウェアを活用すると良い。例えば、Pythonの場合はSymPyライブラリを用いて解析解を計算することができる。

3. 3 ロボットの位置 (と姿勢) が既知の場合

ロボットの速度と角速度が既知であるという条件が最初から満たされていることは実際には少ないと思われる、どちらかと言うと、位置や姿勢の軌道が分かっている場

合の方が多いと思われる。そこで位置と姿勢から速度と角速度を計算する方法を示す。ロボットのタイプによって、扱いがやや異なるので、それぞれ場合分けして考える。

まず、ロボットが全方向に移動可能である場合について述べる。この場合は位置と姿勢の軌道データを数値微分することで速度と角速度の軌道データを近似計算することができる。例えば、速度は中心差分を用いて

$$v_1(t) \cong \frac{1}{2h}(p_1(t+h) - p_1(t-h)) \quad (15)$$

で近似計算できる。

次に、ロボットが直進方向にしか速度を発生させない場合について述べる。この場合、ロボットの速度方向がロボットの姿勢を表すので、速度の軌道データがあれば

$$\theta_1 = \arctan2(v_{1y}, v_{1x}) \quad (16)$$

として姿勢の軌道データを計算することができる。なお $\arctan2(v_{1y}, v_{1x})$ はベクトル v_{1x} の方向を返す関数であり、 $\arctan(v_{1y}/v_{1x})$ の値域を $(-\pi, \pi)$ に拡張したものとす。速度の軌道データは全方位移動可能な場合と同様に位置の軌道データから数値微分で得ることができる。角速度の軌道データは上記の方法で得られた姿勢の軌道データから数値微分することで得ることができる。

なお、ロボットが直進方向にしか速度を発生させない場合で、位置の軌道が直線と円弧で定められている場合は、位置の軌道から速度を容易に計算することができる。また位置の軌道から円弧の半径も調べることができ、速度と円弧の半径を式(2)に当てはめれば角速度も知ることができる。

4 シミュレーション

本節では、直進方向のみに速度を発生させることができるロボットが通路の角を曲がるシミュレーションを行い、挙動を確認する。

4.1 問題設定

本研究ではロボットが最初直進し、その後円弧軌道に遷移して 90° ほど進み、再度直進する図3のような動作を考える。

けん引システムのパラメータは $l_1 = l_2 = 500$ とし、ロ

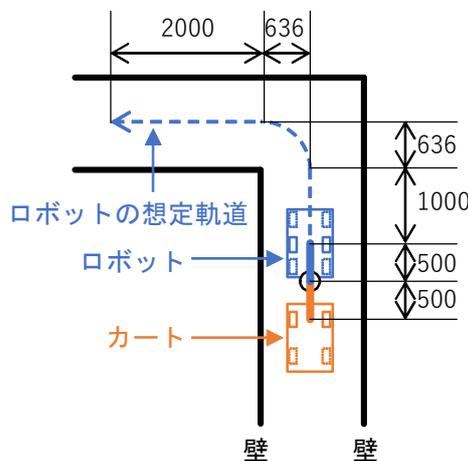


図3 例題

ボットは1秒間 $V_1 = 1000, \omega_1 = 0$ で直進し、次に1秒間 $V_1 = 1000, \omega_1 = \pi/2$ で円弧軌道を描き、最後に2秒間 $V_1 = 1000, \omega_1 = 0$ で直進するとした。初期状態は $\theta_1 = \theta_2 = \pi/2, [p_{x1}, p_{y1}]^T = [0, 0]^T$ とした。なお、このときの、円弧運動の半径は $1000/(\pi/2) \cong 636$ である。

4.2 シミュレーションの実装方法

ロボットの速度と姿勢の角速度が既知である場合の数値解によるシミュレーションと解析解によるシミュレーションの実装方法を説明する。また、ロボットの位置の軌道のみが既知の場合の数値解によるシミュレーションの実装方法についても説明する。

4.2.1 ロボットの速度と角速度の軌道が既知である場合の数値解の計算

ロボットの速度と姿勢の角速度が既知であるとして4次のルンゲ・クッタ法を用いて数値解を求めた。本研究では SciPy の `solve_ivp()` を利用して数値解の計算を行った。位置の単位を [mm]、角度の単位を [rad] として実装した。 `solve_ivp()` ではシミュレーションの時間の刻み幅を自動で設定する機能もあるが、指定しないと軌道がガタガタになったので、 `t_eval` で時間の刻み幅を 2^{-10} s とした。 `solve_ivp()` の精度に関する引数をデフォルトのままを使うと l_1 や l_2 に大きな誤差が生じたので、 `rtol=1.e-9` として実装した。(なお、4次のルンゲ・クッタ法を自力で実装し、時間の刻み幅を 2^{-10} s として試した場合や、 `atol` や `rtol` をデフォルトの値のまま8次のルンゲ・クッタ法を用いた場合には l_1 や l_2 に大きな誤差は生じなかった。)

4.2.2 ロボットの速度と姿勢が既知である場合の解析解の計算

次に解析解について述べる。この例題は直進、円弧、直進と状態が遷移するので、これら3つの状態に分割して考えた。このとき各状態におけるロボットの位置軌道や姿勢軌道の解析解は容易に計算することができ、

$$\theta_1 = \omega_1 t + \theta_{1,init} \quad (17)$$

であり、 $\omega_1 \neq 0$ のとき

$$p_1 = \frac{V_1}{\omega_1} \begin{bmatrix} \sin(\omega_1 t + \theta_{1,init}) - \sin(\theta_{1,init}) \\ -\cos(\omega_1 t + \theta_{1,init}) + \cos(\theta_{1,init}) \end{bmatrix} + p_{1,init} \quad (18)$$

であり、 $\omega_1 = 0$ のときは、

$$p_1 = V_1 \begin{bmatrix} \cos(\theta_{1,init}) \\ \sin(\theta_{1,init}) \end{bmatrix} t + p_{1,init} \quad (19)$$

である。なお、 $p_{1,init}, \theta_{1,init}$ はそれぞれ p_1, θ_1 の初期値である。

式(9)に式(19)を代入して解けばカートに関する解析解が得られるが、複雑な式であったので SymPy の `dsolve()` では解くことができなかった。そこで以下のようにして計算を単純化した。まず、カートの位置はロボットの位置・姿勢とカートの姿勢が分かれば、幾何学的に

$$p_2 = p_1 - l_1 \begin{bmatrix} \cos \theta_1 \\ \sin \theta_1 \end{bmatrix} - l_2 \begin{bmatrix} \cos \theta_2 \\ \sin \theta_2 \end{bmatrix} \quad (20)$$

として計算できるので、微分方程式を直接解かなくても p_1, θ_1, θ_2 が分かれば計算可能である。したがって、あとは式(9)の最下段の成分の式 ($\dot{\theta}_2$ に関する式) を解け

ば良い. さらに, V_1, l_1, l_2 は値を入れ, ω_1 は文字式のままで計算することとすると

$$\begin{aligned} \dot{\theta}_2 &= 2 \sin(\omega_1 t + \theta_{1,init} - \theta_2) \\ &\quad - \omega_1 \cos(\omega_1 t + \theta_{1,init} - \theta_2) \end{aligned} \quad (21)$$

であるが, これでもまだSymPyの`dsolve()`で解くことが出来なかった. そこで,

$$\theta = \theta_1 - \theta_2 \quad (22)$$

$$\theta_{init} = \theta_{1,init} - \theta_{2,init} \quad (23)$$

として, θ に関する微分方程式

$$\dot{\theta} = \omega_1 + 2 \sin \theta - \omega_1 \cos \theta \quad (24)$$

に変形し, これをSymPyの`dsolve()`で解いた. すると,

$$\theta = 2 \operatorname{atan} \left(\frac{\omega_1}{2} - \frac{\omega_1 e^{-2t}}{2} + e^{-2t} \tan \left(\frac{\theta_{init}}{2} \right) \right) \quad (25)$$

が得られた. よって,

$$\theta_2 = \omega_1 t + \theta_{1,init}$$

$$-2 \operatorname{atan} \left(\frac{\omega_1}{2} - \frac{\omega_1 e^{-2t}}{2} + e^{-2t} \tan \left(\frac{\theta_{1,init} - \theta_{2,init}}{2} \right) \right) \quad (26)$$

である.

以上で得られた式(17), (19), (20), (26)を用いて, まず直進のシミュレーションを行い, 次に初期値を更新して円弧のシミュレーションを行い, また初期値を更新して直進のシミュレーションを行った.

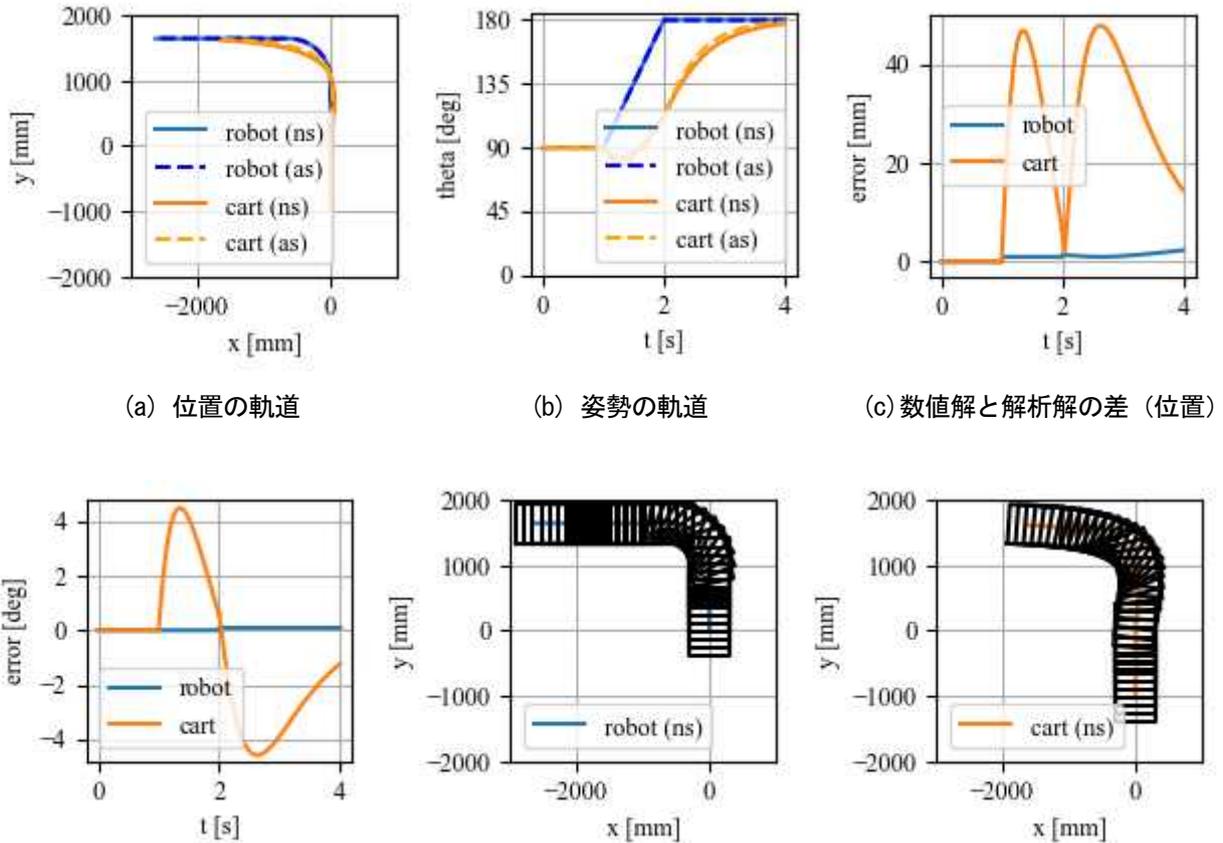
4. 2. 3 ロボットの位置の軌道のみが既知である場合の数値解の計算

最後に, ロボットの速度と角速度が未知で, 位置の軌道だけが分かっている場合のシミュレーションでは, 前目の方法で求められた解析解の 2^{-10} sごとのロボットの位置データを既知のデータとして扱い, シミュレーションを行った.

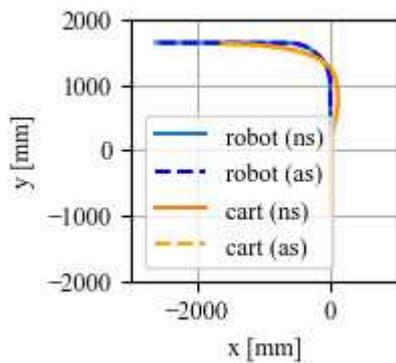
4. 3 シミュレーション結果

ロボットの速度と角速度の軌道が既知の場合の数値解によるシミュレーションと解析解によるシミュレーションを行った結果を図4に示す. また, ロボットの位置の軌道のみが既知である場合の数値解によるシミュレーション結果を図5に示す. それぞれ位置と姿勢の軌道を示すとともに, これらの解析解との差をプロットしている. また, 数値解で得られた位置と姿勢の軌道をロボットやカートが走行した際のロボットやカートを0.125 sおきに描写したのも示す. 同図ではロボットとカートのサイズは幅600 mm, 長さ900 mmと仮定した. このような図を描けば通路を走行する際にカートが壁に衝突しないかなどを確認することができる.

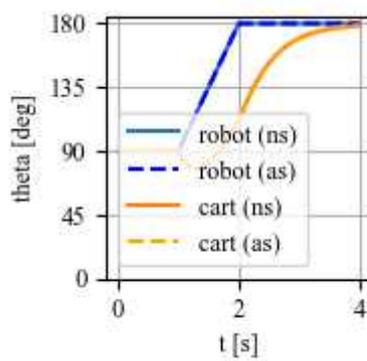
図4と図5を見ると, いずれの場合でも数値解は解析解と同様の軌道を描いていることが分かるが, ロボットの速度と角速度が既知の場合の方が精度が悪いことが分かる. これは直線から円弧, 円弧から直線に切り替わる際の角速度の不連続性が問題になっている可能性が考えられる. そしてロボットの位置軌道から速



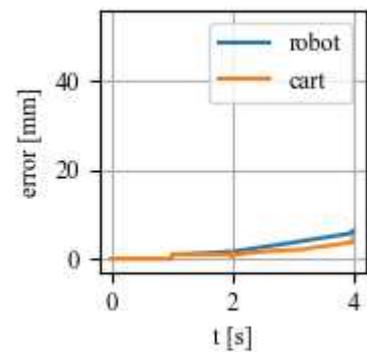
(a) 位置の軌道 (b) 姿勢の軌道 (c) 数値解と解析解の差 (位置)
 (d) 数値解と解析解との差 (姿勢) (e) 数値解のロボットの動作 (f) 数値解のカートの動作
 図4 ロボットの速度と角速度の軌道が既知のときのシミュレーション結果. 「ns」は数値解, 「as」は解析解を意味する.



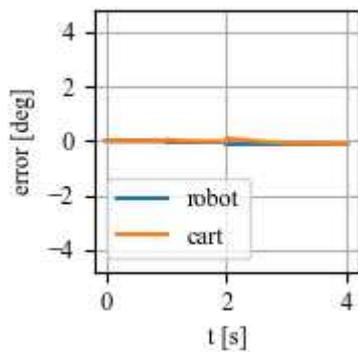
(a) 位置の軌道



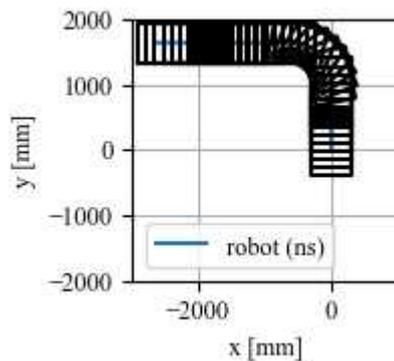
(b) 姿勢の軌道



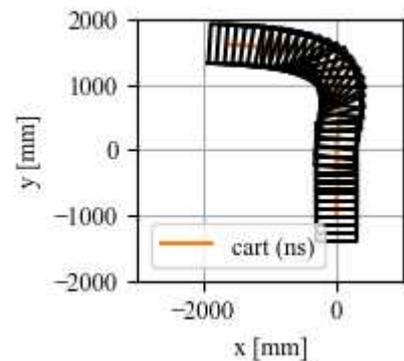
(c) 数値解と解析解の差 (位置)



(d) 数値解と解析解との差 (姿勢)



(e) 数値解のロボットの動作



(f) 数値解のカーットの動作

図5 ロボットの位置の軌道が既知のときのシミュレーション結果。「ns」は数値解、「as」は解析解を意味する。

度と角速度を数値計算した場合はその不連続性がうまく近似され、影響が少なくなった可能性が考えられる。なお、直線と円弧の繋ぎ合わせによる軌道は単純であるのでよく用いられるが、現実にはロボットが直線と円弧の繋ぎ合わせの位置軌道を誤差なく走行することはそもそも困難である。例えば、前輪ステアリング方式であれば、あるタイミングで一瞬にしてステアリング角を所望の角度に変更する必要があり、そのような動作は現実的ではない。

5 結言

本研究では自律移動ロボットや自動搬送台車などが、固定車を二つ備えるカー트를けん引するケースにおいて、カー트의軌道をシミュレーションする方法を示した。これにより、カー트가障害物に衝突しないかシミュレーションで確認することができるようになる。

ただし、ロボットがあらかじめ定めた経路を走行するのではなく、自律的に経路を生成する場合で、その経路

生成の部分にユーザーが関与することができない場合は、地図に制約を加えるなどして「自律」の裁量幅を制限しないとカー트가壁に衝突しないことを保証することは難しい。

なお、ロボットが円弧軌道を描く場合に、シミュレーションすることなくカー트의回転半径がどのような値に収束するかを計算することもできる。これらについて別の報告²⁾に示す。

参考文献

- 1) SciPy: solve_ivp, https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html (検索日: 2024. 11. 14)
- 2) 福本靖彦: 移動ロボット等によるけん引軌道の計算—シミュレーションレスでカー트의振る舞いを解析する方法—, 香川県産業技術センター研究報告, No. 25 (2025).

移動ロボット等によるけん引軌道の計算 —シミュレーションレスでカートの振る舞いを解析する方法—

福本 靖彦

自律移動ロボットや無人搬送車でカートをけん引して物品を搬送する場合がある。このときのロボット等の軌道とカートの軌道は異なる。本研究ではロボット等が円弧軌道で通路の角を曲がる場合を想定し、ロボット等が等速で円軌道を走行する場合のカートの軌道について検討を行った。特に本研究ではシミュレーションを行うことなくカートの振る舞いを大まかに把握するための解析を行った。具体的には、カートの軌道が円軌道に収束するかどうか、収束先の回転半径はいくらか、回転中心はどこか、カートが切り返し動作を行うかどうかを解析した。解析結果はシミュレーションで検証した。

1 緒言

自律移動ロボットや無人搬送車等（以降、単にロボットと記す）でカート等（以降、単にカートと記す）をけん引して物品を搬送する場合（図1）、一般にはロボットの軌道とカートの軌道は異なり、ロボットが通路の角を曲がる際に、カートがロボットよりも内側を走行する場合もあれば、外側を走行する場合もある。これにより、ロボットは通過することができても、カートが壁などに衝突するようなことが起こり得る。

これに対して著者はロボットの軌道が与えられた場合のカートの軌道をシミュレーションするための計算式を求めた結果を報告している¹⁾。本稿では、シミュレーションを行わなくてもカートの挙動を大まかに把握することを目的として、更なる解析を行った結果を報告する。特に、ロボットが円軌道を描くときに、カートの軌道が円軌道に収束するかどうか、収束先の回転半径はいくらか、回転中心はどこか、カートが切り返し動作を行うかどうかについて理論解析を行った。以上の理論解析の結果はシミュレーションにより検証した。これらの結果を応用すると、例えば図2のようにカートの軌道を大まかに解析することが可能となる。

なお、本研究の結果の一部は高野による解説記事²⁾で示されている内容と一致しているが、高野による解説記事では帰納的に示されたと思われる結果を、本研究では解析的に導いている。

本稿では、2節でロボットによるけん引の問題設定を行うとともに移動ロボットや台車の運動の基礎的事項を説明する。3節で理論解析の結果を示す。4節でシミュレーションによる検証結果を示す。5節でまとめを述べる。

2 けん引による搬送

本節では、ロボットによるけん引の問題設定と基礎的事項について説明する。

2.1 問題設定

ロボットは前後進と旋回が可能なものを考える。具体的には二輪速度差方式や前輪ステアリング方式のものが該当する。カートは二つの固定車を有するものとする。ロボットとカートは回転対偶で連結されているものとする。

ロボットやカートの運動は図3のように二次元で考える。ロボットの基準点は二つの駆動輪の中心や後輪の中心とし、カートの基準点は二つの固定輪の中心とする。対偶の位置は、ロボットの二つの駆動輪を結ぶ直線の垂直二等分線上であり、基準点からロボットの後方に下がった位置とする。また、カート側から見ると、カートの二つの固定輪の垂直二等分線上で、カートの基準点から前

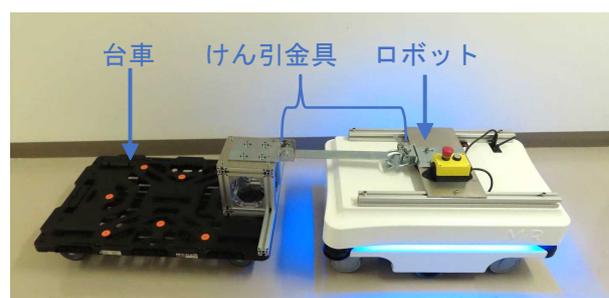


図1 ロボットによる平台車けん引システムの試作例。ロボットはMiR100 (Mobile Industrial Robots)、けん引金具はJB-702BおよびJB702C (スペースシア)、台車はBT900J3-E100 (トラスコ中山) を利用。

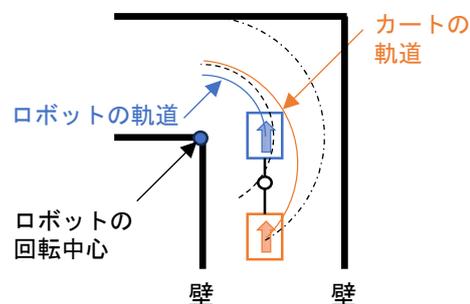


図2 ロボットが円弧軌道で角を曲がる時のカートの軌道の大まかな解析例。破線と一点鎖線の曲線はロボットの回転中心を中心とした円弧である。ある条件を満たすとカートは一点鎖線の円弧上から開始して破線の円弧に近づく軌道を描く。満たすべき条件と破線の円弧の半径の計算式は3節で示す。

方に進んだ位置とする．本稿で用いる文字を以下の通り定義する．

- $p_1, p_2 \in \mathbb{R}^2$: ロボットとカートの位置
- $p_{x1}, p_{y1}, p_{x2}, p_{y2} \in \mathbb{R}$: p_1, p_2 の X 成分と Y 成分
- $v_1, v_2 \in \mathbb{R}^2$: ロボットとカートの速度
- $V_1, V_2 \in \mathbb{R}$: ロボットとカートの速度の直進方向成分
- $\theta_1, \theta_2 \in \mathbb{R}$: ロボットとカートの姿勢
- $\omega_1, \omega_2 \in \mathbb{R}$: ロボットとカートの姿勢の時間微分
- $r_1, r_2 \in \mathbb{R}$: ロボットとカートの軌道の回転半径
- $l_1 \geq 0$: ロボットの基準点から回転対偶までの距離
- $l_2 > 0$: カートの基準点から回転対偶までの距離

なお，距離の単位は[mm]を用いる．角度の単位は基本的に[rad]を用い，角度の範囲は基本的に $(-\pi, \pi]$ の範囲を扱うが，説明の都合でこの範囲から外れた値を扱う場合もある．また， π から微小な正数 ε だけ増加すると $-\pi + \varepsilon$ となるように両端が繋がっているものとして考えるが，このループ構造により π から $-\pi + \varepsilon$ に変化することは値の増加と見なす．同様に $-\pi + \varepsilon$ から π へ変化することは減少と見なす． ω_1 はロボットが左回りの時に正で，右回りの時に負である．

2. 2 ロボットの運動の基礎的事項

一般の二次元のロボットの運動は $v_1 (\in \mathbb{R}^2)$ と ω_1 で規定することができるが，二輪速度差方式や前輪ステアリング方式のロボットは前後進の方向にしか速度を発生させることができないので，その運動は $V_1 (\in \mathbb{R})$ と ω_1 で規定することができる．このとき

$$v_1 = V_1 \begin{bmatrix} \cos \theta_1 \\ \sin \theta_1 \end{bmatrix} \quad (1)$$

である．

二輪速度差方式や前輪ステアリング方式のロボットは基本的に円軌道を描く．直進動作も可能であるが，これは回転半径が無大の場合として考えることができる．

ロボットの回転半径は

$$r_1 = \frac{V_1}{\omega_1} \quad (2)$$

で計算できる．本稿では，回転半径が正の場合はロボットが左回り，負の場合は右回りであるとして，負の回転半径も扱う．

2. 3 固定二輪方式のカートの運動の基礎的事項

固定二輪方式のカートの運動も二輪速度差方式や前輪ステアリング方式のロボットと同様に前後進と回転だけが可能であり， V_2 と ω_2 で規定することができる． v_2 は

$$v_2 = V_2 \begin{bmatrix} \cos \theta_2 \\ \sin \theta_2 \end{bmatrix} \quad (3)$$

となる．

ロボットとの差は，ロボットは自身が車輪を能動的に動かすのに対して，固定二輪方式のカートは外部からの力を受けて受動的に車輪が動くという点である．

2. 4 けん引時のカートの速度と角速度

ロボットがカートをけん引して走行したときのカートの速度と回転速度を計算する式は著者による別の報告¹⁾に示した通りで

$$\begin{bmatrix} V_2 \\ \omega_2 \end{bmatrix} = \begin{bmatrix} l_1 \omega_1 \sin(\theta_1 - \theta_2) + V_1 \cos(\theta_1 - \theta_2) \\ \frac{V_1}{l_2} \sin(\theta_1 - \theta_2) - \frac{l_1 \omega_1}{l_2} \cos(\theta_1 - \theta_2) \end{bmatrix} \quad (4)$$

である．本稿ではこれを変形して

$$\begin{bmatrix} V_2 \\ \omega_2 \end{bmatrix} = \begin{bmatrix} \omega_1 \sqrt{r_1^2 + l_1^2} \sin \theta \\ -\frac{\omega_1}{l_2} \sqrt{r_1^2 + l_1^2} \cos \theta \end{bmatrix} \quad (5)$$

$$\theta = \alpha + \beta, \quad (6)$$

$$\alpha = \theta_1 - \theta_2, \quad (7)$$

$$\sin \beta = \frac{r_1}{\sqrt{r_1^2 + l_1^2}} \quad (8)$$

$$\cos \beta = \frac{l_1}{\sqrt{r_1^2 + l_1^2}} \quad (9)$$

として解析を行う． α と β は図3の中に図示したような幾何学的意味を持つ．なお， $l_1 = r_1 = 0$ のときはロボットだけがその場旋回し，カートは静止している状態を意味する．また， β の取りうる範囲はロボットが左回りのとき $\beta \in [0, \pi/2]$ であり，ロボットが右回りのとき $\beta \in [-\pi/2, 0]$ である(図4)．

3 ロボットが等速円運動する場合の理論解析

本節では，ロボットが等速円運動する場合のカートの軌道を理論的に解析する．なお $\omega_1 = 0$ のときはロボットが直進するので，本節では $\omega_1 \neq 0$ とする．また式(5)より， $l_1 = r_1 = 0$ のときはロボットはその場旋回し，カートは静止することが分かっており，本節では l_1 と r_1 の少なくともどちらか一方は非零である場合の解析を行う．

3. 1 理論解析結果

本項では先に解析結果を示す．導出は次項で示す．まず，

$$l_2^2 \leq r_1^2 + l_1^2 \quad (10)$$

のときカートの軌道は等速円運動に収束するが，式(10)を満たさないとき，カートの軌道は等速円運動に収束しないという結果が得られた．

したがって，ロボットから回転対偶までの距離がある程度大きくなるようにけん引金具を設計する必要があることが分かる．同様に，回転対偶からカートの基準点ま

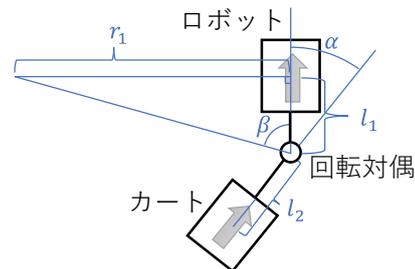
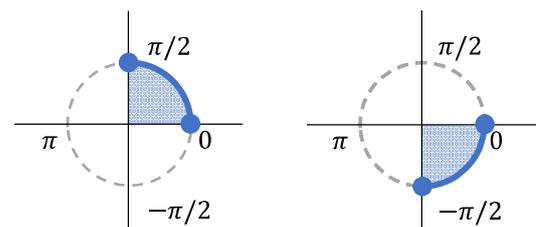


図3 けん引システムの構成



(a) ロボット：左回り (b) ロボット：右回り

図4 β の取りうる範囲

での距離がある程度短くなるようけん引金具を設計する必要があることも分かる。

続いて、等速円運動に収束する場合の収束先の軌道の回転半径は

$$r_2 = \pm \sqrt{r_1^2 + l_1^2 - l_2^2} \quad (11)$$

であり、回転中心はロボットの軌道の回転中心と一致することが分かった。なお、 r_2 はロボットが左回りの時に正で、ロボットが右回りの時に負である。また r_2 が虚数になる場合は、上述の通りカートの軌道は等速円運動にならない。

よって、 $l_1 > l_2$ でけん引金具を設計すると、十分時間が経過したときにカートはロボットよりも外側を走行し、 $l_1 < l_2$ で設計すると、カートはロボットよりも内側を走行し、 $l_1 = l_2$ で設計すると、カートとロボットの軌道は一致することが分かる。特に、 $l_1 = l_2$ となるようにけん引金具を設計すると、ロボットの回転半径がいくらであってもカートは安定で、軌道はロボットの軌道に収束するので、障害物回避のための設計が比較的容易になると考えられる。(ただし、あくまで収束後の軌道が一致するものであり、収束の過程では軌道にずれは存在する。)

更に、ロボットが左回りであり、 θ の初期値が $\theta \in (0, \pi)$ を満たすときは、ロボットの回転中心からカートまでの距離は単調増加あるいは単調減少し、 $\sqrt{r_1^2 + l_1^2 - l_2^2}$ に収束し、逆に $\theta \in (0, \pi)$ を満たさない場合は図5に示すような切り返し動作が生じるという結果が得られた(図6(a))。同様にロボットが右回りの場合は、 θ の初期値が

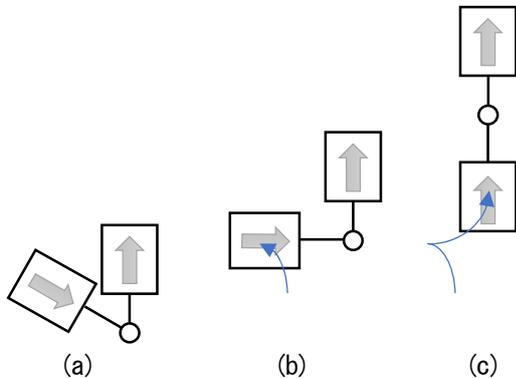
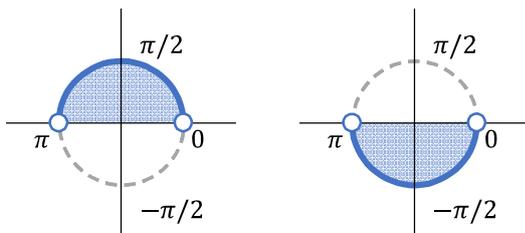


図5 カートの切り返し動作。(a)の状態からロボットが直進すると、(b)に至るまでカートが後退し、その後(c)に至るまで前進する。



(a) ロボット：左回り (b) ロボット：右回り

図6 ロボットの回転中心からカートまでの距離が単調増加あるいは単調減少して $\sqrt{r_1^2 + l_1^2 - l_2^2}$ に収束する θ の初期値の範囲。またこのとき切り返し動作も発生しない。

$\theta \in (-\pi, 0)$ を満たすときに、単調に $\sqrt{r_1^2 + l_1^2 - l_2^2}$ に収束し、 $\theta \in (-\pi, 0)$ を満たさないときに切り返し動作が生じるという結果が得られた(図6(b))。

単調に収束する場合、振る舞いが単純であるのでカートの軌道をだまかに予想することができる(図2)。一方、切り返し動作が発生する場合はカートの軌道が予想しづらく、内輪差や外輪差といった考え方とは別の検討が必要になってくる。

以上の結果の中で r_1 は用いているが、 V_1 と ω_1 は直接用いていないことに注目されたい。したがって、 r_1 が一定になる制約の下で V_1 や ω_1 を変動させても以上の結果は同様に成り立つ。式(10)と式(11)の条件は、高野による解説記事¹⁾に示されてものと同じである。ただし、同記事では「電算機によって計算してみると」と記されており、帰納的に得られた条件であることが示唆されているが、本研究では解析的に導出を行った。

3. 2 理論解析結果の導出

式(5)より、 θ が定数に収束すると $[V_2, \omega_2]^T$ が定数になる。このとき、 $\omega_2 \neq 0$ であればカートが等速円運動する。したがって、 θ のダイナミクスに着目する。

θ のダイナミクスは

$$\dot{\theta} = f(\theta) \quad (12)$$

とすると、

$$\begin{aligned} f(\theta) &= \dot{\theta}_1 - \dot{\theta}_2 + 0 \\ &= \omega_1 - \omega_2 \\ &= \omega_1 + \frac{\omega_1}{l_2} \sqrt{r_1^2 + l_1^2} \cos \theta \end{aligned} \quad (13)$$

であり、式(12)は非線形システムである。この非線形システムにおいて、 θ が何らかの定数に収束し、その収束先に $\omega_2 \neq 0$ である条件を調べる。また、カートの軌道が等速円運動に収束した場合の、カートの軌道の回転半径と回転中心についても計算する。最後に、カートの軌道の途中で切り返し動作が生じないための条件も導出する。

3. 2. 1 一次元非線形システムの振る舞い

ここで、本節で用いる一次元の非線形システムの解析に関する基礎的事項を説明しておく。なお、本目では必要に応じて変数の引数として時刻 t を明示する必要がある。例えば θ を $\theta(t)$ と書く場合がある。また、本目では $\theta \in (-\pi, \pi]$ の区間の両端のループについては考慮しない。

$\bar{\theta}$ がある定数であるとする。式(12)の非線形システムがある時刻 t_1 に

$$\theta(t_1) = \bar{\theta} \quad (14)$$

であり、それ以降の時刻 $t > t_1$ について $\theta(t) = \bar{\theta}$ を満たし続けて値が変動しないとする。このような $\bar{\theta}$ を平衡点と呼ぶ。 $\bar{\theta}$ の具体的な値は

$$\dot{\theta} = 0 \quad (15)$$

という条件から得られる。非線形システムの場合、平衡点は1個とは限らず、0個の場合もあれば、複数個の場合もある。

平衡点の安定性について説明する。本稿での安定性の説明は分かりやすさを優先し、やや厳密さを欠くので注意されたい。 $\bar{\theta}$ が漸近安定な平衡点であれば、初期値 $\theta(0)$ が $\bar{\theta}$ の近くにあるとき、時間がたつと $\theta(t)$ は $\bar{\theta}$ に収束する(図7(a))。一方、 $\bar{\theta}$ が不安定な平衡点であれば、初期値 $\theta(0)$ が $\bar{\theta}$ の近くにあっても $\theta(t)$ と $\bar{\theta}$ の距離は徐々に離れていく(図7(b))。また、両者の特性を合わせ持ち、ある

方向からはその平衡点に収束するが、反対方向ではその平衡点から遠ざかるという、半安定な平衡点というものもある (図7(c)).

ここで、不安定な平衡点であっても、一度その値になれば動かないので、不安定な平衡点も価値があるように思えるかもしれないが、現実には外乱などの影響で不安定な平衡点に留まり続けることは不可能である。半安定な平衡点も同様で、外乱が生じる場合は半安定な平衡点に留まり続けることはできない。

したがって重要なのは漸近安定な平衡点であるが、上記の説明では初期値 $\theta(0)$ が $\bar{\theta}$ の近くである場合の局所的な挙動しか議論されていない。 $\bar{\theta}$ に収束するような θ の範囲を $\bar{\theta}$ の吸引域と呼ぶが、一般に一次元の非線形システムでは平衡点は安定な平衡点と不安定な平衡点が交互に配置され (例えば、安定な平衡点が二つ連続して配置されると、間にどちらに収束するか不定な位置が発生し、そこが不安定な平衡点となる)、ある安定な平衡点 $\bar{\theta}$ の吸引域は前後の不安定な平衡点の手前までの領域である。

平衡点の安定性の判別方法について説明する。非線形システムを平衡点 $\bar{\theta}$ 周りでテイラー展開すると

$$\begin{aligned} \dot{\theta} &= f(\theta) \\ &= f(\bar{\theta}) + \left. \frac{\partial f(\theta)}{\partial \theta} \right|_{\theta=\bar{\theta}} (\theta - \bar{\theta}) + \left. \frac{\partial^2 f(\theta)}{\partial \theta^2} \right|_{\theta=\bar{\theta}} (\theta - \bar{\theta})^2 \\ &\quad + \dots \\ &= \left. \frac{\partial f(\theta)}{\partial \theta} \right|_{\theta=\bar{\theta}} (\theta - \bar{\theta}) + \left. \frac{\partial^2 f(\theta)}{\partial \theta^2} \right|_{\theta=\bar{\theta}} (\theta - \bar{\theta})^2 + \dots \end{aligned} \quad (16)$$

が得られる。ここで、 $(\theta - \bar{\theta}) \ll 1$ のとき、 $(\theta - \bar{\theta})$ の累乗は $(\theta - \bar{\theta})$ に比べて十分小さくなるので、2次以降の項を打ち切って

$$\dot{\theta} \approx \left. \frac{\partial f(\theta)}{\partial \theta} \right|_{\theta=\bar{\theta}} (\theta - \bar{\theta}) \quad (17)$$

として近似して考える。このとき

$$\frac{d}{dt}(\theta - \bar{\theta}) \approx \left. \frac{\partial f(\theta)}{\partial \theta} \right|_{\theta=\bar{\theta}} (\theta - \bar{\theta}) \quad (18)$$

となり、線形システムとなる。これを解くと

$$(\theta(t) - \bar{\theta}) \approx (\theta(0) - \bar{\theta}) \exp\left(\left. \frac{\partial f(\theta)}{\partial \theta} \right|_{\theta=\bar{\theta}} t\right) \quad (19)$$

である。したがって

$$\Re\left[\left. \frac{\partial f(\theta)}{\partial \theta} \right|_{\theta=\bar{\theta}}\right] < 0 \quad (20)$$

であれば、

$$\lim_{t \rightarrow \infty} (\theta(t) - \bar{\theta}) = 0 \quad (21)$$

であり、 θ は $\bar{\theta}$ に収束する。なお、 $\Re[\cdot]$ は引数の実部を取り出す操作である。逆に $\Re[\partial f(\theta)/\partial \theta|_{\theta=\bar{\theta}}]$ が正であれば $\theta(t) - \bar{\theta}$ が発散するので θ も発散する。

なお、 $\Re[\partial f(\theta)/\partial \theta|_{\theta=\bar{\theta}}]$ が0であるとき、式(16)のテイラー展開の $(\theta - \bar{\theta})$ の項が $(\theta - \bar{\theta})^2$ よりも十分大きくても $(\theta - \bar{\theta})$ の係数が0であるので、 $(\theta - \bar{\theta})$ の二乗以上の項の影響を無視することができなくなる。したがって、線形近似したシステムを解析しても安定性を判定することができない。このような場合、一般的な判定手法はなく、個別に判断を行う必要がある。

最後に、システムの分岐現象について説明する。分岐現象とは、システムのパラメータ (例えば l_1 や l_2) が変動したことでシステムの振る舞いが大きく変わることを言う。具体的には、平衡点の数や安定性が変わることなどを指す (図8)。安定な平衡点と不安定な平衡点が合わさって消滅するものをサドル・ノード分岐と呼ぶ。また、サドル・ノード分岐とトランスクリティカル分岐が合わさったピッチフォーク分岐と呼ばれる分岐現象もある。

3. 2. 2 カートの平衡点

式(13)より、

$$\frac{\omega_1}{l_2} \sqrt{r_1^2 + l_1^2} \cos \bar{\theta} = -\omega_1 \quad (22)$$

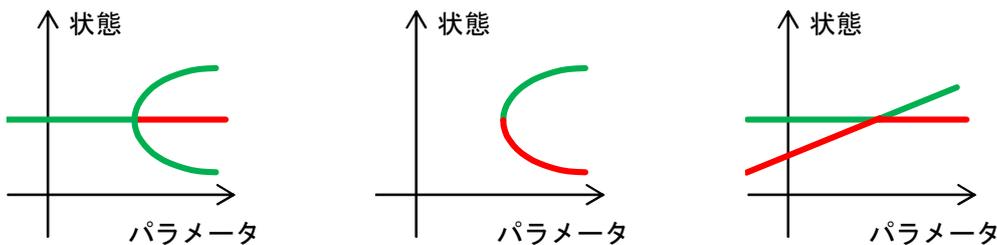
を満たす $\bar{\theta}$ が平衡点である。このとき、

$$\cos \bar{\theta} = -\frac{l_2}{\sqrt{r_1^2 + l_1^2}} \quad (23)$$

である。よって、



図7 平衡点の種類。矢印はベクトル場を表し、 θ がその値にあるときに θ が移動する方向を示す。



(a) サドル・ノード分岐 (b) トランスクリティカル分岐 (c) ピッチフォーク分岐

図8 分岐現象の種類。緑は安定な平衡点を表し、赤は不安定な平衡点を表す。システムのパラメータの値を変えると平衡点の位置が変わり、ある値のところで平衡点の数や安定性が変わっている。

$$\bar{\theta} = \begin{cases} \arccos\left(-\frac{l_2}{\sqrt{r_1^2 + l_1^2}}\right) & l_2 < r_1^2 + l_1^2 \\ \pi & l_2 = r_1^2 + l_1^2 \\ \text{解なし} & l_2 > r_1^2 + l_1^2 \end{cases} \quad (24)$$

である。なお、 $l_2 = r_1^2 + l_1^2$ の場合は、解が一つであることを明示したもので、 $l_2 < r_1^2 + l_1^2$ と同じく $\arccos\left(-\frac{l_2}{\sqrt{r_1^2 + l_1^2}}\right)$ と考えて問題はない。cosが0以下になるので $\bar{\theta} \in (\pi/2, 3\pi/2)$ であり(図9)、解が二つの場合は $\bar{\theta}$ は $(\pi/2, \pi)$ と $(\pi, 3\pi/2)$ に一つずつ解をもつ。

また、 $l_2 < r_1^2 + l_1^2$ のときに平衡点が二つあり、 $l_2 = r_1^2 + l_1^2$ となるとときに二つの平衡点が合わさって一つとなり、 $l_2 > r_1^2 + l_1^2$ のときには平衡点がないことから、サドル・ノード分岐が発生していると考えられる。

最後に、式(13)の二段目に着目すると、平衡点において

$$\dot{\theta} = \omega_1 - \omega_2 = 0 \quad (25)$$

である。本節では $\omega_1 \neq 0$ として考えているので、 θ が平衡

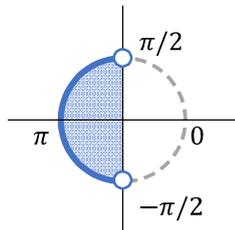
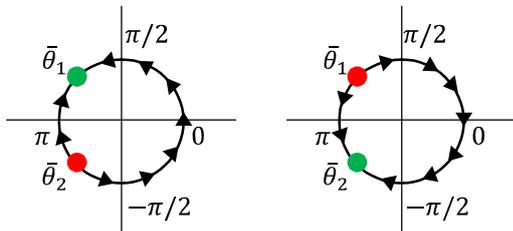


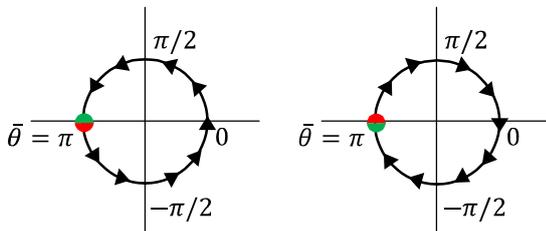
図9 $\bar{\theta}$ の取りうる範囲



(a) ロボット左回り

(b) ロボット右回り

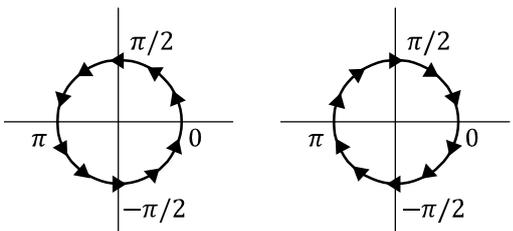
図10 平衡点が二つある場合の θ のベクトル場



(a) ロボット左回り

(b) ロボット右回り

図11 平衡点がある場合の θ のベクトル場



(a) ロボット左回り

(b) ロボット右回り

図12 平衡点がない場合の θ のベクトル場

点にあるとき $\omega_2 \neq 0$ を満たしていることが分かる。したがって、あとは θ が安定な平衡点に収束することだけを示せば、カートの軌道は円軌道に収束することになる。

3. 2. 3 θ の平衡点の安定性

$\Re[\partial f(\theta)/\partial \theta|_{\theta=\bar{\theta}}]$ を計算すると

$$\left. \frac{\partial f(\theta)}{\partial \theta} \right|_{\theta=\bar{\theta}} = -\frac{\omega_1}{l_2} \sqrt{r_1^2 + l_1^2} \sin \bar{\theta} \quad (26)$$

である。前目で示したとおり $\bar{\theta} \in (\pi/2, 3\pi/2)$ であるので、ロボットが左回りのときは $\bar{\theta} \in (\pi/2, \pi)$ となる平衡点が漸近安定である。また、ロボットが右回りのときは $\bar{\theta} \in (\pi, 3\pi/2)$ の平衡点が漸近安定である。

漸近安定な平衡点の吸引域を考える。ここでは角度を $[0, 2\pi)$ の範囲で考える。以下は図10を参照しながら読みたい。まずロボットが左回りの場合を考える。二つの平衡点を $\bar{\theta}_1$ 、 $\bar{\theta}_2$ とし、 $\bar{\theta}_1 < \bar{\theta}_2$ とする。このとき $\bar{\theta}_1$ は安定な平衡点であり、 $\bar{\theta}_2$ は不安定な平衡点である。 θ が $(\bar{\theta}_1, \bar{\theta}_2)$ の範囲内にあるとき式(21)より $\dot{\theta} < 0$ であり、 θ は単調減少して安定な平衡点 $\bar{\theta}_1$ に向かう。一方、 $\theta \in [0, \bar{\theta}_1)$ のとき $\dot{\theta} > 0$ であり、 θ は単調増加し安定な平衡点 $\bar{\theta}_1$ に向かう。また、 $\theta \in (\bar{\theta}_2, 2\pi)$ のときも $\dot{\theta} > 0$ であり、 θ は単調増加して 2π を超えて $[0, \bar{\theta}_1)$ の範囲に入り、やはり安定な平衡点 $\bar{\theta}_1$ に向かう。つまり、不安定な平衡点を除くすべての領域が安定な平衡点 $\bar{\theta}_1$ の吸引域である。ロボットが右回りの場合もまた、同様の議論により不安定な平衡点を除くすべての領域が安定な平衡点 $\bar{\theta}_2$ の吸引域であることを示すことができる。

ここで、 $\bar{\theta} = \pi$ のときは $\partial f(\theta)/\partial \theta|_{\theta=\bar{\theta}} = 0$ になるので、これだけでは挙動が分からない。そこで $\bar{\theta} = \pi$ のときを別途考える。以下は図11を参照しながら読みたい。このとき、 $l_2 = r_1^2 + l_1^2$ なので平衡点の一つだけであり、また式(13)と $l_2 = r_1^2 + l_1^2$ から

$$\dot{\theta} = \omega_1(1 + \cos \theta) \quad (27)$$

が得られる。よって $\theta = \bar{\theta} = \pi$ で $\cos \theta = -1$ となると $\dot{\theta} = 0$ であるが、それ以外の場合は

$$\text{sign}(\dot{\theta}) = \text{sign}(\omega_1) \quad (28)$$

である。つまり、ロボットの円運動が左回りの場合であれば、 θ は単調増加であり初期値から増加し、(場合によっては π を超えて $-\pi$ に至り、更に増加して)平衡点 $\bar{\theta}$ に収束する。逆に、ロボットが右回りの場合は単調減少であり、初期値から減少し、(場合によっては $-\pi$ を超えて π に至り、更に減少して)平衡点 $\bar{\theta}$ に収束する。したがって、 θ は任意の初期値で平衡点 $\bar{\theta}$ に収束する。ただし現実的には、外乱の影響で平衡点から値がずれた際に、ずれる方向によってはすぐに $\bar{\theta}$ に戻らずに更に θ が一周するまで運動する、ということが起こり得ることに注意が必要である。

なお、平衡点がない場合の挙動も考えてみる。このときは任意の θ で式(28)が成り立つ。したがって、図12のようにロボットが左回りの時は θ は単調増加し、ロボットが右回りの時は θ は単調減少する。平衡点がないので、停止することなく値は変動し続ける。

3. 2. 4 カートの軌道

本目では θ が漸近安定あるいは半安定な平衡点に収束するときの、ロボットの回転中心からカートまでの距離を計算する。

図13のように、ロボットの回転中心を原点に取り、ロボットの初期位置が $[[r_1], 0]^T$ になるように座標系を取ると、ロボットの回転中心からカートまでの距離は $\|p_2\|$ で表される。このとき、ロボットの位置 p_1 と姿勢 θ_1 を以下のように書くことができる。

$$p_1 = |r_1| \begin{bmatrix} \cos\left(\theta_1 - \frac{\pi}{2}\right) \\ \sin\left(\theta_1 - \frac{\pi}{2}\right) \end{bmatrix}, \quad (29)$$

$$\theta_1 = \omega_1 t. \quad (30)$$

また、 p_2 は幾何学的関係から

$$p_2 = p_1 - l_1 \begin{bmatrix} \cos\theta_1 \\ \sin\theta_1 \end{bmatrix} - l_2 \begin{bmatrix} \cos\theta_2 \\ \sin\theta_2 \end{bmatrix} \quad (31)$$

と書けるので、

$$\|p_2\| = \left\| p_1 - l_1 \begin{bmatrix} \cos\theta_1 \\ \sin\theta_1 \end{bmatrix} - l_2 \begin{bmatrix} \cos\theta_2 \\ \sin\theta_2 \end{bmatrix} \right\| \quad (32)$$

であり、これを地道に変形すると

$$\|p_2\| = \sqrt{r_1^2 + l_1^2 + l_2^2 + 2l_2 \sqrt{r_1^2 + l_1^2} \cos\theta} \quad (33)$$

が得られる。

ここで $t \rightarrow \infty$ の極限を考えると

$$\begin{aligned} \lim_{t \rightarrow \infty} \|p_2\| &= \sqrt{r_1^2 + l_1^2 + l_2^2 + 2l_2 \sqrt{r_1^2 + l_1^2} \left(-\frac{l_2}{\sqrt{r_1^2 + l_1^2}}\right)} \\ &= \sqrt{r_1^2 + l_1^2 - l_2^2} \end{aligned} \quad (34)$$

である。つまり、カートはロボットの回転中心を回転中心とした半径 $\sqrt{r_1^2 + l_1^2 - l_2^2}$ の円軌道に収束する。

次に、収束過程について考える。ロボットが左回りの場合、図10(a)や図11(a)に示した通り、 θ は初期値が $\theta \in (0, \pi)$ であれば $\theta \in (0, \pi)$ を満たし続けて、単調に安定な平衡点あるいは半安定な平衡点へ向かう。このことを考慮して式(33)を見ると、 θ の初期値が $\theta \in (0, \pi)$ であれば $\|p_2\|$ は単調減少あるいは単調増加して $\sqrt{r_1^2 + l_1^2 - l_2^2}$ に収束することが分かる。ロボットが右回りの場合も同様であり、 θ の初期値が $\theta \in (-\pi, 0)$ であれば $\|p_2\|$ は単調減少あるいは単調増加して $\sqrt{r_1^2 + l_1^2 - l_2^2}$ に収束する。

なお、 $\|p_2\|$ はロボットの回転中心からカートまでの距離であるが、 r_2 はカートの回転中心からカートまでの距離に符号を付与したものであり、回転中心が一致する場合は $\|p_2\| = |r_2|$ である。よって

$$\lim_{t \rightarrow \infty} r_2 = \pm \sqrt{r_1^2 + l_1^2 - l_2^2} \quad (35)$$

である。複合はロボットが左回りのとき正で、右回りのとき負である。なお、ロボットが左回りで θ の初期値が

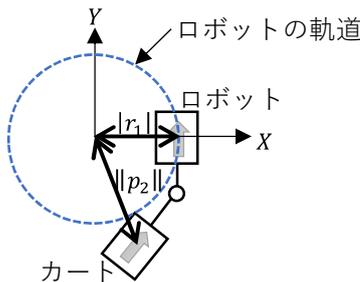


図13 ロボットの回転中心からカートまでの距離

$\theta \in (0, \pi)$ であったとしても、 r_2 が単調増加あるいは単調減少するとは限らない。

3. 2. 5 カートの切り返し動作

カートの切り返し動作が発生しない条件について考える。なお本目でも、カートの軌道が等速円運動に収束することを前提条件とする。すなわち、 θ が漸近安定な平衡点か半安定な平衡点を持つとする。

切り返し動作は速度が正から負に切り替わる、あるいは負から正に切り替わるポイントで起こる。したがって、カートの速度に着目する。まず、ロボットが等速円運動するとき、カートも前進で等速円運動することを想定すると、 V_2 の初期値が正であることが必須である。

ただ、これだけでは不十分で、これに加えて、カートが等速円運動に収束する中で、一度速度が負になり、再度正に戻るといった動作が起こる可能性も検討する必要がある。つまり、初期状態を含め

$$V_2 > 0 \quad (36)$$

を満たす必要がある。ただし、収束先では必ずしも満たす必要がない。なぜならば、収束先で $V_2 = 0$ であっても、それ以降カートは $V_2 = 0$ を保ち続けて停止し、反転して走行することはないためである。

ロボットが左回りで、漸近安定な平衡点を持つときを考える。このときは、式(5)より $\theta \in (0, \pi)$ のとき $V_2 > 0$ を満たす。安定な平衡点は $\bar{\theta} \in (\pi/2, \pi)$ であり、 $(0, \pi)$ に含まれている。そして θ は初期値が $\theta \in (0, \pi)$ をみたすとき、初期値が安定な平衡点より小さければ単調増加して安定な平衡点へ収束し、初期値が安定な平衡点より大きければ単調減少して安定な平衡点へ収束する(図10(a))。したがって、初期値において $\theta \in (0, \pi)$ が満たされていれば、 $\theta \in (0, \pi)$ を満たし続けるので、切り返し動作が生じないことが保証される。

ロボットが左回りで、半安定な平衡点を持つときを考える。このとき平衡点は $\bar{\theta} = \pi$ であり、 θ は初期値が $\theta \in (0, \pi)$ をみたすとき、 θ は単調増加して半安定な平衡点 π へ収束する(図11(a))ため、収束先以外で $\theta \in (0, \pi)$ を満たし、やはり切り返し動作が生じないことが保証される。

同様にロボットが右回りで、漸近安定な平衡点を持つときを考える。このとき、 $\theta \in (-\pi, 0)$ のとき $V_2 > 0$ を満たす。右回りのときの安定な平衡点は $\bar{\theta} \in (-\pi, -\pi/2)$ であり、 $(-\pi, 0)$ に含まれている。そして初期値が $\theta \in (-\pi, 0)$ を満たすとき、初期値が平衡点の値より大きければ単調減少して安定な平衡点に収束し、小さければ単調増加して安定な平衡点に収束する(図10(b))。したがって、初期値において $\theta \in (-\pi, 0)$ が満たされていれば、切り返し動作が生じないことが保証される。

同様にロボットが右回りで、半安定な平衡点を持つときを考える。このときも同様の議論で、初期値が $\theta \in (-\pi, 0)$ を満たすとき、 θ は初期値から単調減少して $-\pi$ に到達する(図11(b))ため、その間、 $\theta \in (-\pi, 0)$ を満たし、切り返し動作が発生しないことが保証される。

よってロボットが左回りのとき、 θ の初期値が $\theta \in (0, \pi)$ を満たしていれば切り返し動作が生じないことが分かった。またロボットが右回りのときは、 θ の初期値が $\theta \in (-\pi, 0)$ を満たしていれば、切り返し動作が生じないことが分かった。

4 数値実験

これまでに示した解析結果が正しいことを検証するためにシミュレーションを行う。本研究では、著者による別の報告¹⁾に示した4次のルンゲ・クッタ法で5つの数値例のシミュレーションを行った。シミュレーションの条件を表1に示す。各数値例の結果を4.1項から4.5項に示すとともに、4.6項では分岐解析の例を示す。

4.1 数値例1 (カートの軌道がロボットの円軌道より内側の円に収束する例)

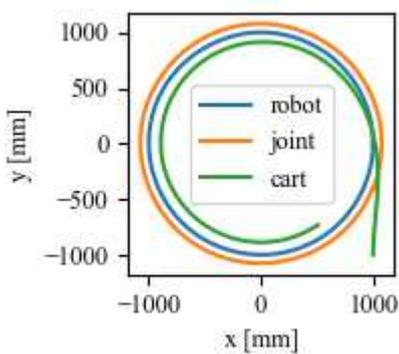
ロボットが1周するシミュレーションを行った際の軌道、 θ の時間変化、原点からカートまでの距離を図14に示す。カートの姿勢の初期値はロボットとカートが直進

してきたときを想定したもので、ロボットの姿勢の初期値と合わせた。理論解析で示された通り、カートの軌道は円軌道に収束しており、ロボットと同じく原点を中心としているように見える。最終のカート位置と原点との距離は889.1 mmであり、理論値の894.4に近い値であった。また、 θ の最終値は $\theta = 123.8^\circ$ であり、これも理論値の 123.9° に近い値であった。カートの切り返し動作も発生しておらず、原点からカートまでの距離は単調減少であった。

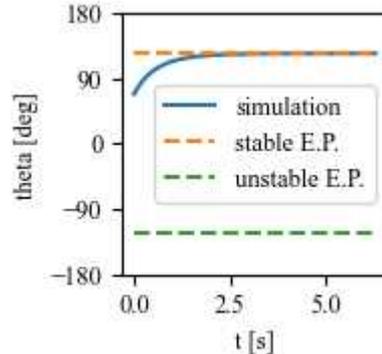
4.2 数値例2 (ロボットが右回り、カートとロボットの回転半径が一致する例)

表1 数値実験のパラメータ。括弧で囲んだ項目は他のパラメータから計算したものの、二重線より下は4節の理論計算により得られたもの。 $\bar{\theta}$ で、実線で下線を引いたものは漸近安定な平衡点であり、点線で下線を引いたものは半安定な平衡点である。添え字として“init”と付したものは初期値を表す。

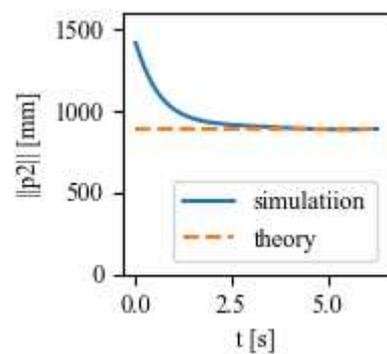
	数値例1	数値例2	数値例3	数値例4	数値例5
V_1	1000	1000	1000	1000	1000
ω_1	1	-1	1	2.24	3.33
(r_1)	1000	-1000	1000	447.2	300
(ロボット 左回り/ 右回り)	左回り	右回り	左回り	左回り	左回り
l_1	400	500	600	400	400
l_2	600	500	400	600	600
$p_{1,init}$	$(r_1, 0)$	$(r_1, 0)$	$(r_1, 0)$	$(r_1, 0)$	$(r_1, 0)$
$\theta_{1,init}$	90°	-90°	90°	90°	90°
$\theta_{2,init}$	90°	-90°	$-101.9^\circ / -99.9^\circ$	$90^\circ / -42.8^\circ$	90°
θ_{init}	68.2°	-63.4°	$-109.1^\circ / -111.1^\circ$	$48.2^\circ / -179.0^\circ$	36.9°
α_{init}	0°	0°	$-168.1^\circ / -170.1^\circ$	$0^\circ / -227.2^\circ$	0°
β	68.2°	-63.4°	59.0°	48.2°	36.9°
$\bar{\theta}$	<u>$123.9^\circ, -123.9^\circ$</u>	<u>$116.6^\circ, -116.6^\circ$</u>	<u>$110.1^\circ, -110.1^\circ$</u>	<u>180.0°</u>	-
(平衡点の 数)	2	2	2	1	0
r_2	894.4	1000.0	1095.4	0.0	-
切り返し	なし	なし	あり/あり	なし/あり	-



(a) 軌道



(b) θ のダイナミクス



(c) 原点からの距離

図14 数値例1 (カートの軌道がロボットの円軌道より内側の円に収束する例)。なお，“joint”は回転対偶部を示し，“E.P.”は平衡点 (Equilibrium Point) を意味する。

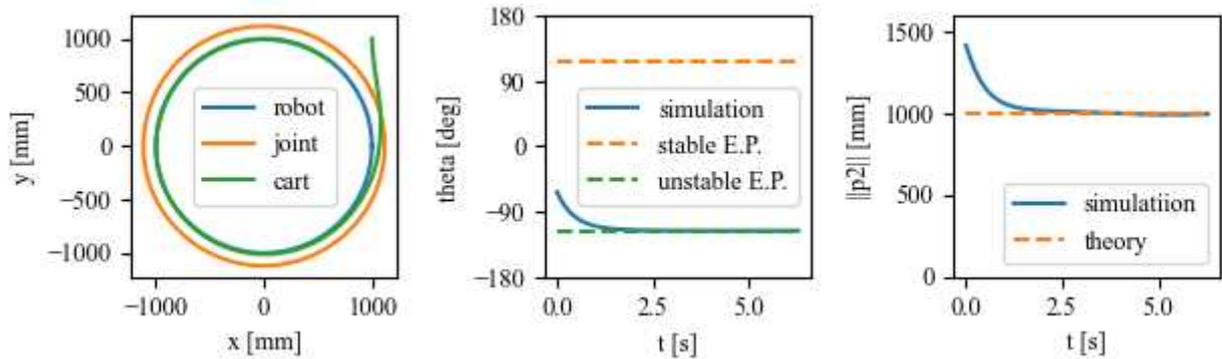
理論解析により、ロボットが前進右回りの等速円運動を行い、またカートの軌道がロボットの軌道に収束するよう前項の条件を調整した条件で、ロボットが一周するシミュレーションを行った結果を図15に示す。カートが半周程度したところからは、カートとロボットの軌道がよく重なっていることが観測された。

4.3 数値例3 (ロボットより回転半径が大きい, 不安定平衡点の近傍から開始する例)

理論解析の結果を用いてカートの回転半径がロボットの回転半径より大きくなるよう4.1項の条件を調整し

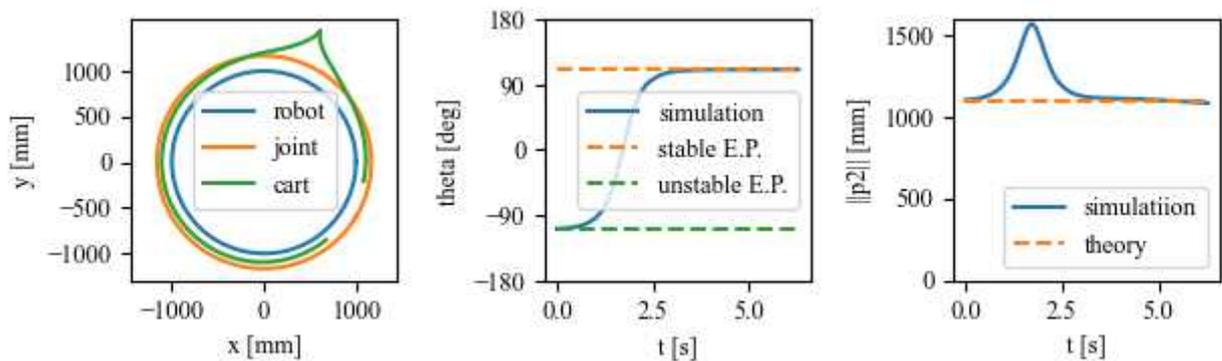
て、ロボットを一周させるシミュレーションを行った結果を図16と図17に示す。なお、カートの初期位置は不安定な平衡点より 1° 大きい場合と 1° 小さい場合の二パターンでシミュレーションを行った。これらの初期値はどちらも理論解析により繰り返し動作が生じる初期値とされるものである。

θ の初期値をどちらにした場合も、想定されていた通り、不安定な平衡点である 111.6° から遠ざかり、安定な平衡点 -111.6° へ収束し、カートの軌道は円軌道に収束している。カートの円軌道の半径は想定通りロボットの軌



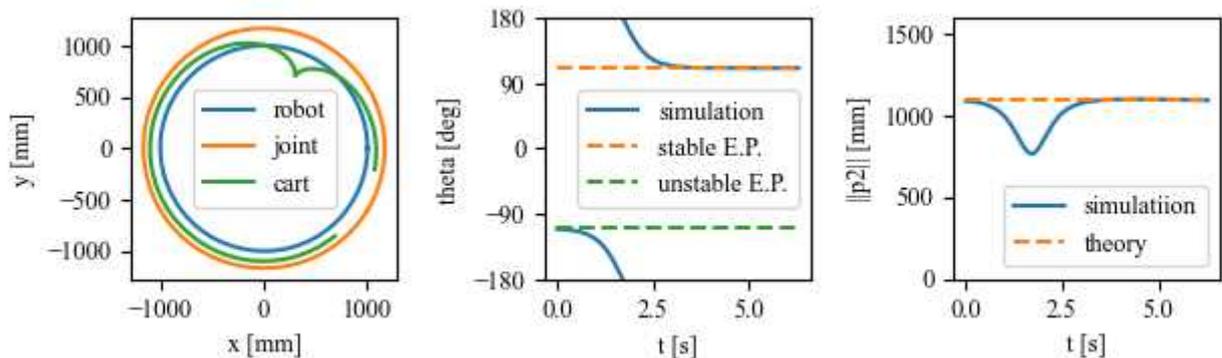
(a) 軌道 (b) θ のダイナミクス (c) 原点からの距離

図15 数値例2 (ロボットが右回り, カートとロボットの回転半径が一致する例)



(a) 軌道 (b) θ のダイナミクス (c) 原点からの距離

図16 数値例3 (ロボットより回転半径が大きい, 不安定平衡点の近傍 (-109.1°) から開始する例)



(a) 軌道 (b) θ のダイナミクス (c) 原点からの距離

図17 数値例3 (ロボットより回転半径が大きい, 不安定平衡点の近傍 (-111.1°) から開始する例)

道の半径よりも大きく、ロボットと同じく原点を中心としているように見える。なお、平衡点近くから開始したため、カートの初期位置は最終の円軌道に非常に近い位置から始まっているが、動き始めてすぐにカートが外側あるいは内側にずれて切り返し動作が行われており、想定されていた切り返し動作が実際に起こることが確認された。

4. 4 数値例 4 (平衡点が半安定になる例)

理論解析により、平衡点が一つだけになる条件としてシミュレーションを行った。このとき、カート軌道は回転半径 0 に収束する、すなわち原点でその場旋回する状態に収束する。 θ の初期値は $\alpha = 0$ である 48.2° と、半安定な平衡点の不安定側近傍である -179.0° の二パターンを検討した。理論解析によると、前者では切り返し動作が発生せず、後者では切り返し動作が発生する。本項ではロボットが 1 周するシミュレーションに加え、ロボットが 30 周するシミュレーションも行った。

シミュレーション結果を図 18 と図 19 に示す。まず、 θ の初期値が 48.2° の場合、図 18(a) に示すように、ロボットが 1 周したただけではカートはロボットの回転中心に近づいているもののまだ十分収束しているとは言えない。図 18(b) に示すようにロボットが 30 周すると原点まであと 4.3 mm のところまで到達している。

次に θ の初期値が -179.0° の場合のシミュレーション結果 (図 19) に着目すると、初期値が平衡点近くなので、ロボットの初期位置も原点近く (10.5 mm) であり、カートが 1 周したただけではまだ原点から 11.1 mm の位置に留まっている。ただし、30 周すると、一度大きく外に広がって切り返し動作を行い、最終的には再度原点近く (原

点から 20.4 mm の位置) に戻っている。以上のシミュレーションにより、半安定な平衡点の不安定側に初期値を置いた場合、 θ は 1 周の違いの半安定な平衡点に安定側から収束の様子が確認できた。なお、切り返し動作を含む大きな位置の変動はロボットが 20 周以上した時点で起こっており、実用上そのような運用をすることは考えにくいので、あまり問題にならない可能性がある。ただし、収束速度はパラメータに依存するので十分注意は必要である。

4. 5 数値例 5 (円軌道に収束しない例)

本項では数値例 1 に対して、ロボットの回転半径を小さくして、理論解析では平衡点を持たないような条件とした場合の挙動を解析する。

シミュレーション結果を図 20 に示す。なお、 θ の変化を見やすくするよう、ここでは $-\pi$ 以下になった場合に π にループせずプロットしている。シミュレーションでも θ は単調減少であり、カートの位置は円軌道にはならず、フラクタル図形を描いている。

4. 6 分岐解析

数値例 1, 4, 6 の挙動は l_1, l_2, V_1 をそれぞれ 400, 600, 1000 に固定して r_1 を変動させたときの振る舞いの変化を見たものと言える。本項では r_1 を三パターンだけではなく 0 から 1000 まで連続的に変動させたときの平衡点の理論値を調べる。 ω_1 の値は r_1 の値に合わせて調整する。

図 21 に r_1 を 0 から 1000 まで変動させたときのベクトル場と平衡点の位置と安定性を描画したものを示す。 $r_1 = 447.2$ のところでサドル・ノード分岐が発生していることが分かる。

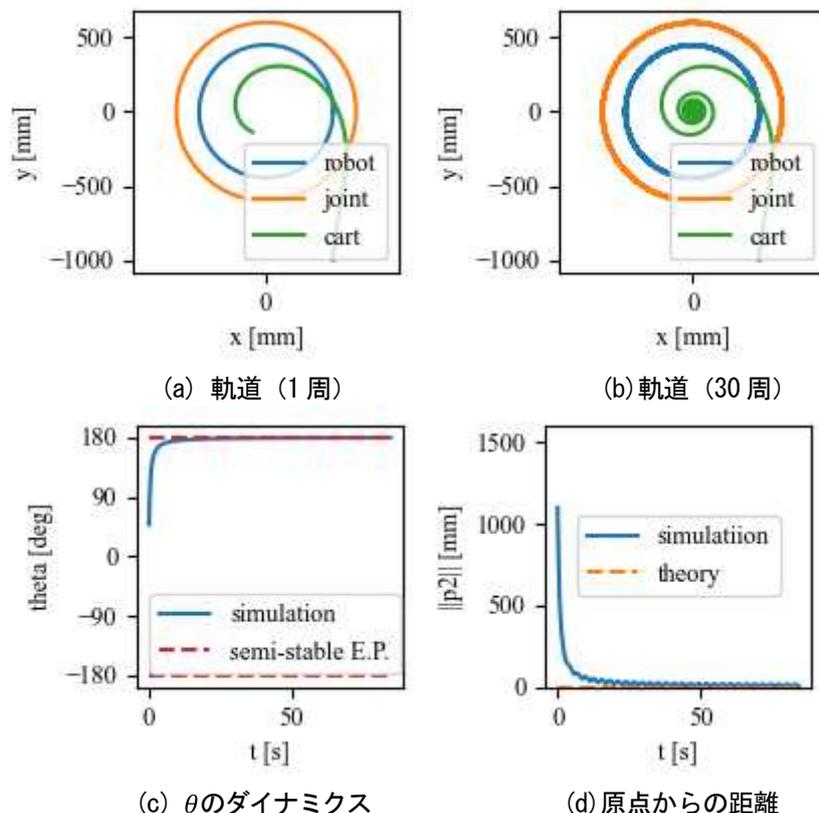
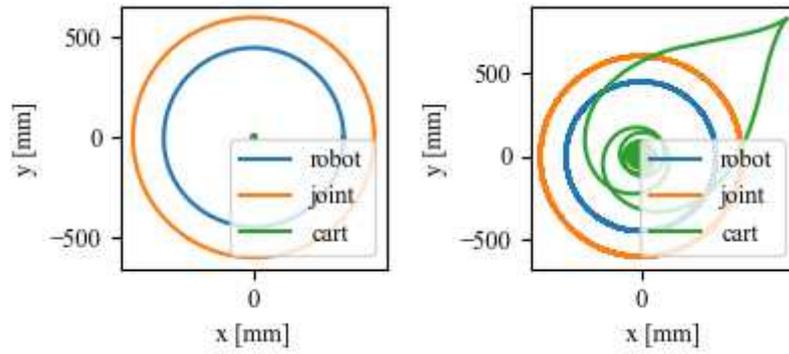
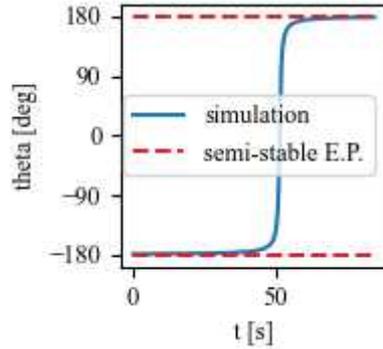


図 18 数値例 4 (平衡点が半安定になる例)

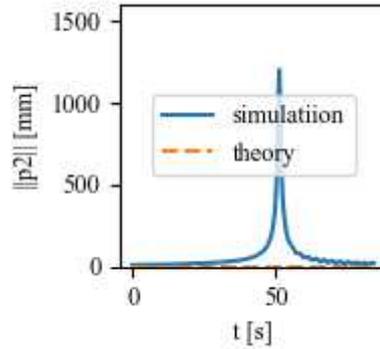


(a) 軌道 (1周)

(b) 軌道 (30周)

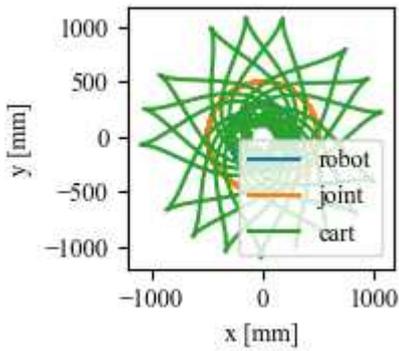


(c) θ のダイナミクス

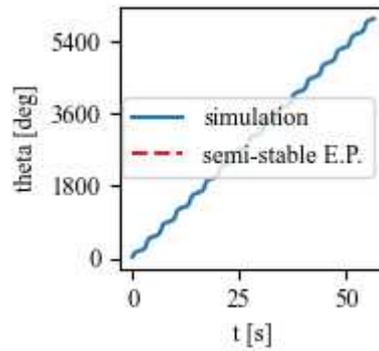


(d) 原点からの距離

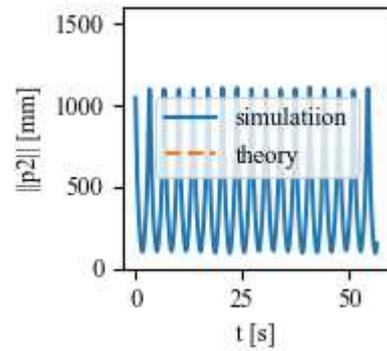
図 19 数値例 4 (半安定な平衡点の不安定側近傍を初期値とした例)



(a) 軌道



(b) 姿勢の差



(c) 原点からの距離

図 20 数値例 5 (円軌道に収束しない例)

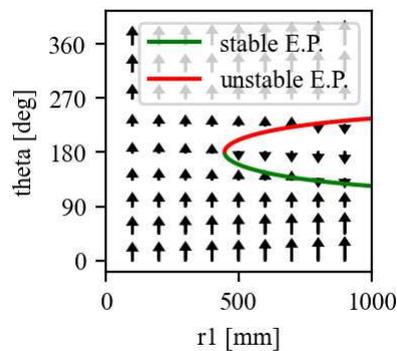


図 21 平衡点とベクトル場. 実線が平衡点であり, 矢印がベクトル場である. 矢印は矢印の根本の位置における θ の向きと大きさを表している. 見やすさを優先し, 角度の範囲を $[0, 2\pi)$ として描画している.

5 結言

本研究では二輪速度差方式や前輪ステアリング方式の自律移動ロボットや自動搬送車が、固定車を二つ備えるカートをけん引するケースにおいて、ロボット等が等速円運動を行うときのカートの振る舞いを解析した。具体的には、以下のことを示した。

- カートの運動が等速円運動に収束するための条件を示した。
- カートの軌道が円軌道に収束するとき、
 - 収束先の円軌道の半径の計算式を示した。
 - 収束先の円軌道の回転中心がロボットの回転中心と一致することを示した。
 - ロボットの回転中心からカートまでの距離が単調増加あるいは単調減少する条件を示した。また、その条件を満たさない場合は切り返し動作(図4)が発生することを示した。

これらの理論解析の結果はシミュレーションにより検証した。

本研究の結果を用いれば、ロボットが直線と円弧を繋ぎ合わせた軌道を走るときに以下のようなことが可能となる。

- カートが不安定化しないような、けん引バーの長さや取り付け位置を検討することができる。
- (シミュレーションを行うことなく) カートの軌道をだまかに見積もることができる(図2)。

今後の課題は、本研究の結果を用いてどのように設計するとカートが障害物に衝突しないシステムが構築できるかといったエンジニアリングが挙げられる。例えば、ロボットやカートのサイズを考慮することや、ロボットとカートを接続する回転対偶の可動範囲の制限を考慮することなどが挙げられる。特に既存のロボットを用いる場合には、ユーザーが軌道のプランナーにほとんど手を入れることができない中で、どのように実装するかということも課題となる。

参考文献

- 1) 福本靖彦: 移動ロボット等によるけん引軌道の計算-軌道のシミュレーション方法-, 香川県産業技術センター研究報告, No. 25 (2025).
- 2) 高野政晴: 車輪移動機構のABC(第3回):運動学, 日本ロボット学会誌, Vol. 13, No. 3, pp. 355-360 (1995).