マニピュレーション実験

C言語に自信のない人は本を持参すること.

人が字や図形を描く際，頭の中でそれらの形を描き，これを紙面に表出させなければならない．本実験では，本研究室で開発した組み立て可能なロボットアームを用いて，このような書字を行うためには，これらを書く位置や姿勢，大きさ等を変形する必要がある．

1.1 字を書くための動作

文字をある紙に書くことを考える．このためには大きく 2 つのことを行う必要がある．

まず，基準となる文字の形が必要となる．そこで，書き取りの時のように，図 1.1 のように基準となる枠を考える．この文字を書くためには経由点を考え，それを線で繋げればよい．点と点をつなぐ線の取りうる可能性は無限にあり，綺麗な字を書くためには滑らかな線でつなぐ必要がある．そのような滑らかな線の一つとして，直線を描くための方法を実現する．

次に，字は紙面に書くためには，上記の基準となる文字を適当な大きさに変え，適切な位置や姿勢にして配置する必要がある．また，場合によっては，文字を傾けたり，することでイタリック体などの書体をつくることができる．このような変換はアフィン変換を応用することで実現できる．

1.2 ロボットアームの運動計画と軌道制御

ロボットアームは多くの場合，関節にアクチュエータが取り付けられ，関節の動きはアクチュエータにより直感的に操作できる．一方，目的の動作を行うためには，その手先位置を思い通りに動かす必要がある．しかしながら，一般的に手先の動きは関節の動きと直感的に対応しておらず，手先の運動を制御するための手法が必要となる．図 1.2 は，この手先を制御するための代表的な手法の一つである逆運動学を持ちいた手

図 1.1:

法を示している．関節角から手先位置を求める方法を順運動学といい，逆に手先位置に相当する関節角を求める方向を逆運動学という．逆運動学を持ちいる方法では，与えられた手先位置に相当する目標関節角を求め，制御系により実現することが求められる．そのためには，以下の 4 つの問題を考える必要がある．

A1 実現したい作業を考える．

A2 ロボットが動ける範囲内 (作業空間) で作業の実現のためのロボットの手先位置の動きを決定する．

A3 この手先の動きに相当するロボットの関節の動きを計算する (逆運動学)．

A4 制御系により，この関節の動きを確実に実現する．

A1 は，例えば，ロボットアームで黒板に M という字を書かせると決めることに相当する．通常，この過程はロボットの構造とは独立に決定できる．

A2 は，黒板に書く時の大きさと位置を決めることに相当する．そのためには，黒板面に沿った直交座標系を考え，字を書くために必要な軌道や通過点をこの座標系の位置として指定する必要がある．また，制御のためには各通過点の到達時間も重要となる．また，確実に作業を実現するためには，ロボットの手が届く範囲を知る必要がある．

A3 は，目標の手先運動に相当する関節運動を決定することである．目的の手先位置は直交座標系で記述されるが，それを実現する関節角は殆どのロボットでは回転型の関節であるため，直感的には決まらない．そこで，目標手先位置から目標関節角を逆運動学と呼ばれる手法により求める．

ロボットが運動する際，重さなどの動的特性やセンサノイズ等により，目標の関節の動きをうまく実現できるとは限らない．そこで，**A4** では，この目標関節角に追従するための適切な制御系が必要となる．

1.3 軌道計画問題

1.3.1 PTP 制御

前節のように，動作目標が複数与えられたり，目標軌道が複雑になったりする場合，図 1.5 のように，多くの中継点を用意し，その中継点を通過するように制御する **PTP 制御**[*] という方法が取られる．

一連の中継点に沿って制御するために，時間と位置がセットになった $N+1$ 個のデータ系列

$$\begin{aligned} \mathcal{T} &= \{t_0, t_1, \cdots, t_N\} \\ \bar{\mathcal{X}} &= \{\bar{\boldsymbol{x}}_0, \bar{\boldsymbol{x}}_1, \cdots, \bar{\boldsymbol{x}}_N\} \end{aligned} \tag{1.1}$$

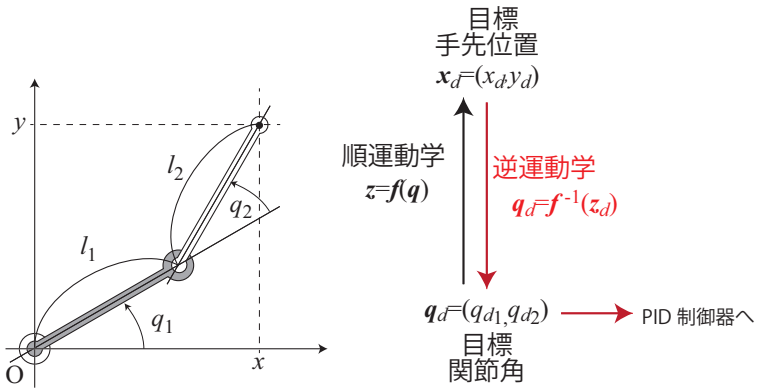

図 1.2: 順運動学と逆運動学による制御

[*] Point To Point

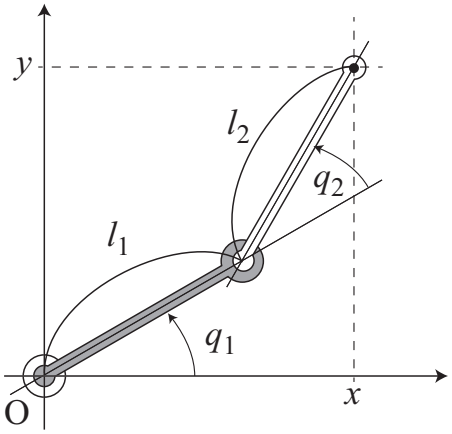

図 1.3: 平面を動く 2 自由度回転関節型ロボットアーム

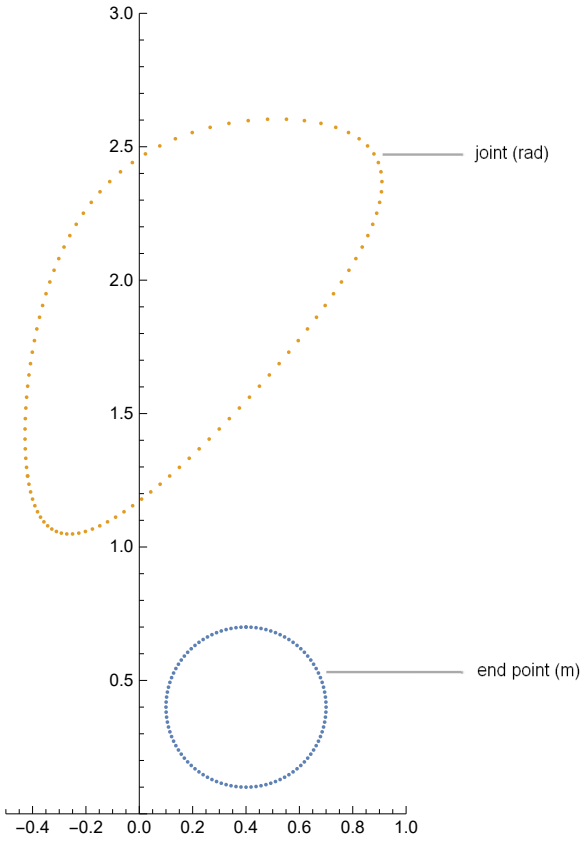

図 1.4: 関節軌道と手先軌道

を用意する．この時，第 i 番目の軌道 $(i = 0, 1, \cdots, N-1)$ は $\boldsymbol{x}_i(t)$ は

$$\boldsymbol{x}_i = \boldsymbol{x}(t; t_i, t_{i+1}, \bar{\boldsymbol{x}}_i, \bar{\boldsymbol{x}}_{i+1}) \tag{1.2}$$

により求めることができる．位置 $\bar{\boldsymbol{x}}_i$ から出発し，位置 $\bar{\boldsymbol{x}}_{i+1}$ に到達する軌道の関数 $\boldsymbol{x}_i(t)$ は，時刻 $t_i \le t \le t_{i+1}$ で有効である．そのため，前後に軌道の無い t_0 以前と t_N 以降の軌道がどうなるかは保証されていない．そ

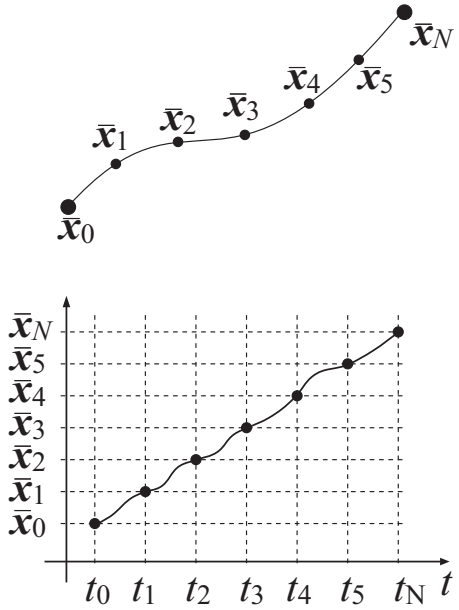

図 1.5: PTP 制御

こで，区間の前後に別の軌道が存在しない端点では，位置を保持するものとする．即ち，

$$\boldsymbol{x}(t) = \begin{cases} \bar{\boldsymbol{x}}_0 & \text{if } t < t_0 \\ \bar{\boldsymbol{x}}_N & \text{if } t \geq t_N \\ \boldsymbol{x}_i(t) & \text{otherwise } (t_i \leq t < t_{i+1}) \end{cases} \tag{1.3}$$

とすることで，区間外の軌道の連続性を保証することができる．

1.3.2 軌道計画の種類

前節のように，関節角や手先位置，目標の接触力などの一連の目標が与えられた場合，$\bar{\boldsymbol{x}}_i$ と $\bar{\boldsymbol{x}}_{i+1}$ へ移動させる方法を考える必要がある．ロボットが確実に動作を実現するためには，二点間の移動方法が重要となる．図 1.6 に示すように，この移動には大きく 2 通りの方法が考えられる．

T1 定値制御: 最終目的点を定値として与える．最終目的点へ到達することが重要であり，そこに至る軌道にはあまり関心がない場合，もしくは，次の点への移動が与える影響が小さいと考えられる場合に用いられる．

T2 連続軌道 (CP[†]) 制御: 最終目的点に到達する滑らかな軌道が必要な時，それを関数として与える方法である．

定値制御の場合，最終目的点さえ分かれば良いので，軌道計画は必要ない．但し，到達点に至る実際の軌道は動力学や制御の方法に依存する．**CP 制御**の場合，軌道を連続軌道 $\boldsymbol{x}(t;\boldsymbol{\phi})$ という形で与えることができる．ここで，$\boldsymbol{\phi}$ は軌道に用いられる未知パラメータであり，; は軌道の設計後と設計段階でのパラメータを区別している．未知パラメータ $\boldsymbol{\phi}$ は初期や終端点での位置や速度，加速度などに関する拘束 (境界条件) により決定される．

[†]Continuous Path

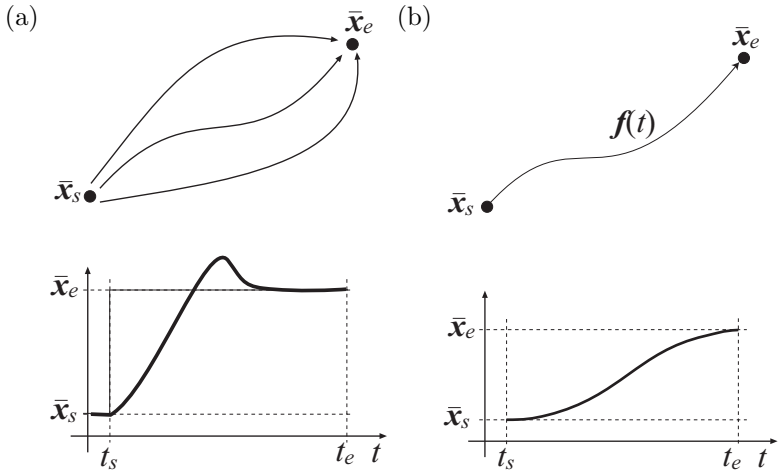

図 1.6: 空間軌道の構成法. (a) 定値制御. (b)CP 制御.

例えば，時刻 t_s, t_e における位置がそれぞれ $\bar{\boldsymbol{x}}_s$, $\bar{\boldsymbol{x}}_e$ である時，境界条件

$$\begin{cases} \boldsymbol{x}(t_s;\boldsymbol{\phi}) &= \bar{\boldsymbol{x}}_s, \\ \boldsymbol{x}(t_e;\boldsymbol{\phi}) &= \bar{\boldsymbol{x}}_e \end{cases} \tag{1.4}$$

から $\boldsymbol{\phi}$ を求める．一度 $\boldsymbol{\phi}$ が決まると，その後は $\boldsymbol{x}$ が t の関数として表せる．一般に中の要素数より等しい数の独立な境界条件が必要となる．

$\boldsymbol{x}_s$ と $\boldsymbol{x}_e$ をつなぐ区間の軌道はどのような関数でも良いが，典型的には 2 点間をつなぐ直線‡により実現する場合が多い．そこで，次節では直線軌道について考える．

1.3.3 2 点間をつなぐ直線軌道

図 1.7 のように，p 次元の空間の 2 点 $\bar{\boldsymbol{x}}_s, \bar{\boldsymbol{x}}_e$ があり，$\bar{\boldsymbol{x}}_s$ から $\bar{\boldsymbol{x}}_e$ に向かって直線上を移動することを考える．移動中の通過点 $\boldsymbol{x}(t)$ が直線上にあるための条件は，

$$\boldsymbol{x}(t) = \alpha_0\bar{\boldsymbol{x}}_s + \alpha_1\bar{\boldsymbol{x}}_e \tag{1.5}$$

$$= (1-\beta)\bar{\boldsymbol{x}}_s + \beta\bar{\boldsymbol{x}}_e \tag{1.6}$$

である．ここで β は $0 \leq \beta \leq 1$ を満足する変数である．もし, $\alpha_0 + \alpha_1 \neq 1$ の場合，$\boldsymbol{x}$ は $\bar{\boldsymbol{x}}_s$ と $\bar{\boldsymbol{x}}_e$ を結ぶ線上から外れてしまう．そのため，式 (1.6) の条件が必要となる．しかしながら，制御の場面では，この条件を満足するだけでは不十分である．

実際のロボットの動きは動特性 (重さなど) に支配され，空間軌道が正しくても (即ち，式 (1.6) を満足している状況でも)，急な加減速などがあるとオーバーシュートなどが生じ，実現軌道は理想の空間軌道から外れてしまう．これを避けるためには，目標の位置だけでなく，その速度や加速度などの条件も決めることが重要となる．言い換えると，2 点間を滑らかな軌道でつなぐための単調増加な時間関数 $\beta(t)$ の設計問題が重要となる．

‡ここでの直線は手先の空間とは限らないことに注意する．例えば，図**??**(b) のように，2 点の関節角 (例えば，P1, P2) を直線で結んでも手先軌道 (Q1-Q2 間の縁) は直線とならない．逆に，手先位置が直線となる場合，これに対応する関節姿勢を結ぶ線は直線にならない．どちらの直線の設計でも使える．

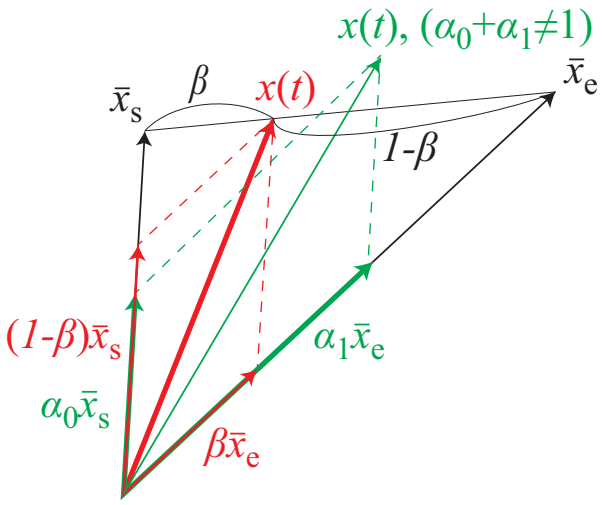

図 1.7: 2 点間をつなぐ直線

$\boldsymbol{x}$ がベクトルの場合，$\boldsymbol{x}$ の各要素を同じ $\beta(t)$ によって重みづければよい．例えば，平面において，$\bar{\boldsymbol{x}}_s = (\bar{x}_s, \bar{y}_s)$ と $\bar{\boldsymbol{x}}_e = (\bar{x}_e, \bar{y}_e)$ をつなぐ軌道 $\boldsymbol{x}(t) = (x(t), y(t))$ は，

$$\boldsymbol{x}(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} z(t; t_s, t_e, \bar{x}_s, \bar{x}_e) \\ z(t; t_s, t_e, \bar{y}_s, \bar{y}_e) \end{bmatrix} \tag{1.7}$$

となる $z(t; t_s, t_e, \bar{a}, \bar{b})$ を設計できればよい．ここで，

$$z(t; t_s, t_e, \bar{a}, \bar{b}) = (1 - \beta(t; t_s, t_e))\bar{a} + \beta(t; t_s, t_e)\bar{b} \tag{1.8}$$

である．$\bar{a}, \bar{b}$ は端点であるため，直線軌道には $\beta(t)$ の設計をすればよい．この方法としてよく用いられるのは次の二つである．

C1 多項式軌道

C2 台形速度軌道

多項式軌道 (**C1**) は，時間に関する多項式により軌道を設計する方法である．多項式の次数を上げることで滑らかさを調整できる．一方，次数にもよるが，ロボットが最大速度で移動できる時間が短くなり，性能を十分引き出せない．台形速度軌道 (**C2**) とは，なるべく長い時間を予め決められた速度で移動する方法である．この時の速度プロファイルが台形になる．ロボットの軌道設計を行う場合，最大速度や加速度はモータの能力などのハードウェアの制限と対応がつけやすいために，産業用ロボットではこの台形速度軌道による PTP 制御がよく用いられる．しかしながら，軌道は 3 種類の方程式からなり，拘束条件が複雑になる．

残りの小節では，多項式軌道と台形速度軌道に関する軌道計画を示す．

1.3.4 β による多項式軌道計画

$z(t)$ を直線軌道にするために，式 (1.8) の $\beta(t)$ を時間に関する多項式軌道として考えていく．β は時刻 t_s に起点 0 から出発し，時刻 t_e に終点 1 になる．この軌道は，次のように表すことができる．

$$\beta(t) = \sum_{i=0}^{n} a_i \left(\frac{t - t_s}{t_e - t_s} \right)^i, \tag{1.9}$$

ここで $a_i(i=0,1,\cdots,n)$ は定数である．1 次，3 次，5 次多項式などがよく用いられる．いずれの軌道も始点から終点まで滑らかにつなぐ軌道を構成することができるが，次数によりその速度や加速度に大きな違いが現れる．n 次の多項式に含まれる $n+1$ 個の未知変数の数は，解くべき拘束条件の数に等しく，例えば，次のように与えられる．

$$\beta(t_s)=0, \qquad \beta(t_e)=1, \tag{1.10}$$

$$\dot{\beta}(t_s)=0, \qquad \dot{\beta}(t_e)=0, \tag{1.11}$$

$$\ddot{\beta}(t_s)=0, \qquad \ddot{\beta}(t_e)=0. \tag{1.12}$$

1 次の場合，終点と始点の位置に関する二つの拘束条件 (1.10) のみとなる．図 1.8(a) のように，この場合，速度が零の状態から一気に最高速になり，停止時には一気に速度が零になる．すなわち，無限の加速度が必要となる．質量のあるロボットアームにはこのような加減速は不可能であり，滑らかに加速する必要がある．3 次の場合，両端点の速度の連続性に関する 4 つの拘束条件 (1.10), (1.11) を考慮できる．図 1.8(b) のように，この場合，速度は滑らかになるが，加速度の絶対値は両端点が大きく効率が悪い．5 次の場合，位置，速度，加速度を含めた 6 つの拘束条件 (1.10)-(1.12) を規定でき，図 1.8(c) のように，加速度も滑らかにすることができる．

ここでは，5 次の場合を考える．Eq. (1.9)-(1.12) を a_i について解くと，

$$\begin{bmatrix} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 10 & -15 & 6 \end{bmatrix} \tag{1.13}$$

なることが分かる．よって，

$$\beta(t;t_s,t_e)=10\left(\frac{t-t_s}{t_e-t_s}\right)^3-15\left(\frac{t-t_s}{t_e-t_s}\right)^4+6\left(\frac{t-t_s}{t_e-t_s}\right)^5 \tag{1.14}$$

となる．時刻 t_s,t_e での位置をそれぞれ $\bar{z}_s,\bar{z}_e$ とした時，この式 (1.14) を式 (1.8) に代入することで

$$z(t;t_s,t_e,\bar{z}_s,\bar{z}_e)=\{1-\beta(t;t_s,t_e)\}\,\bar{z}_s+\beta(t;t_s,t_e)\bar{z}_e. \tag{1.15}$$

とかけ，軌道が求まる．

問題 1.1 $\beta(t)$ の多項式が (1.9) で与えられているとする．

- β が 1 次の場合，a_0,a_1 を求めよ．但し，境界条件は (1.10) で与えられる．
- β が 3 次の場合，$a_0,\cdots,a_3$ を求めよ．但し，境界条件は (1.10)，(1.11) で与えられる．

1.3.5 台形速度軌道

目標軌道の関数 (1.15) の速度軌道を台形にすることを考える．この時間微分は，

$$\dot{z}(t;t_s,t_e,\bar{z}_s,\bar{z}_e)=\dot{\beta}(t;t_s,t_e)(\bar{z}_e-\bar{z}_s) \tag{1.16}$$

となる．$\bar{z}_e-\bar{z}_s$ は定数であるため，$\dot{\beta}(t)$ を台形になるように決定すれば，台形速度軌道を実現できる．

時刻 t_s に位置 $\bar{z}_s$ から速度 0 で出発した軌道が，時刻 t_s+T_b まで加速し，速度 v_0 に達する．この速度で時刻 t_e-T_b まで移動し，その後，最初と同じ大きさの加速度で減速し，時刻 t_e で速度が零になり，位置 $\bar{z}_e$ に達するとする．開始・終端時刻 t_s,t_e と定常状態での速度 v_0 は設計パラメータで指定できるものとす

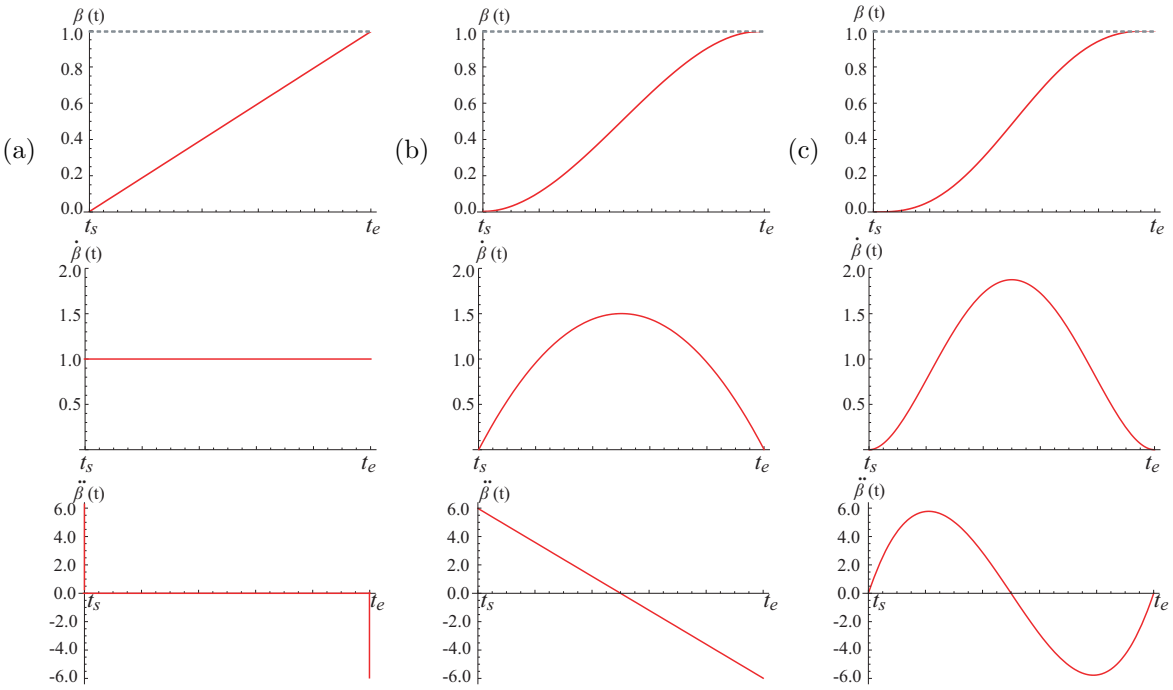

図 1.8: 多項式補間軌道の位置，速度，および加速度．(a)1 次多項式補間．(b) 3 次多項式補間．(c)5 次多項式補間

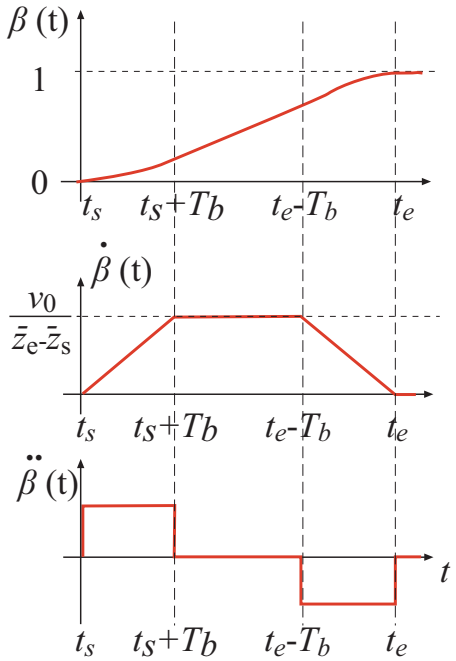

図 1.9: 台形速度軌道．両端での位置情報に加え，最大速度 $v_{\max}$ と最大加速度 $a_{\max}$ が与えられると最大加速度，速度で目的地に移動する方法である．

る．この時，図 1.9 のように，$\beta(t)$ は 3 つの区間 (加速，一定速，減速) に分けることができ，それぞれ 2

次，1 次，2 次の多項式として表現できる．これは，

$$\beta(t) = \begin{cases} \hat{\beta}_1(t) = a + b(t - t_s) + c(t - t_s)^2 & (t_s \leq t < t_s + T_b) \\ \hat{\beta}_2(t) = d + v_0(t - t_s - T_b) & (t_s + T_b \leq t < t_e - T_b) \\ \hat{\beta}_3(t) = e + f(t - t_e) + g(t - t_e)^2 & (t_e - T_b \leq t \leq t_e) \end{cases} \tag{1.17}$$

と表現できる．

式 (1.16) から，

$$\dot{\beta} = \frac{\dot{z}}{\bar{z}_e - \bar{z}_s} \tag{1.18}$$

となるので，β の時間変化は速度を $\bar{z}_e - \bar{z}_s$ で割ったものになる．$t_s \leq t \leq t_e$ の区間の台形状の $\dot{\beta}$ の積分は β の最大値 1 となる．また，図 1.9 より，次の関係が成り立つ．

$$(t_e - t_s - T_b)\frac{v_0}{\bar{z}_e - \bar{z}_s} = 1 \tag{1.19}$$

これを利用すると，未知変数であった加減速の時間 T_b は，次のように求まる．

$$T_b = t_e - t_s - \frac{\bar{z}_e - \bar{z}_s}{v_0} \tag{1.20}$$

β を決定するためには，式 (1.17) の残りの 7 つのパラメータ a, b, c, d, e, f, g を決定する必要がある．境界条件は

- $t = t_s$ の時,

$$\hat{\beta}_1(t_s) = 0, \text{ and } \dot{\hat{\beta}}_1(t_s) = 0,$$

- $t = t_s + T_b$ の時,

$$\hat{\beta}_1(t_s + T_b) = \hat{\beta}_2(t_s + T_b), \text{ and } \dot{\hat{\beta}}_1(t_s + T_b) = \dot{\hat{\beta}}_2(t_s + T_b)$$

- $t = t_e - T_b$ の時,

$$\dot{\hat{\beta}}_2(t_e - T_b) = \dot{\hat{\beta}}_3(t_e - T_b),$$

- $t = t_e$ の時,

$$\hat{\beta}_3(t_e) = 1, \text{ and } \dot{\hat{\beta}}_3(t_e) = 0,$$

となる．それぞれの境界条件を連立して解くと，次のようにパラメータが求まる．

$$\begin{cases} a &= 0, \\ b &= 0, \\ c &= \frac{v_0}{2T_b(\bar{z}_e - \bar{z}_s)}, \\ d &= \frac{v_0 T_b}{2(\bar{z}_e - \bar{z}_s)}, \\ e &= 1, \\ f &= 0, \\ g &= -\frac{v_0}{2T_b(\bar{z}_e - \bar{z}_s)} \end{cases} \tag{1.21}$$

よって，求める方程式は

$$\beta(t) = \begin{cases} \frac{v_0}{2T_b(\bar{z}_e - \bar{z}_s)}(t - t_s)^2 & (t_s \le t < t_s + T_b) \\ \frac{v_0 T_b}{2(\bar{z}_e - \bar{z}_s)} + \frac{v_0}{\bar{z}_e - \bar{z}_s}(t - t_s - T_b) & (t_s + T_b \le t < t_e - T_b) \\ 1 - \frac{v_0}{2T_b(\bar{z}_e - \bar{z}_s)}(t - t_e)^2 & (t_e - T_b \le t \le t_e) \end{cases}, \tag{1.22}$$

$$\text{where} \qquad T_b = t_e - t_s - \frac{\bar{z}_e - \bar{z}_s}{v_0}$$

となる．これを式 (1.15) に代入することで，目的の直線軌道を得ることができる．

問題 1.2 初期と終端時刻 t_s と t_e とその時の位置 $\bar{z}_s$, $\bar{z}_e$，および最高速度 v_0 を適当に与え，$\beta(t)$ と $z(t)$ の位置，速度，加速度のグラフを描くこと．

1.4 ロボットの軌道の記録と再現

1.4.1 軌道の記録

ロボットの手先運動を記録し，再現する場合，最も簡単な方法はロボットに目標の姿勢を取らせ，その時の関節角の系列 $\boldsymbol{q}_i (i = 0, 1, \cdots)$ を記録し，それを再現すれば良い．この方法はティーチングプレイバックと言われ，予め，手先軌道を計算する必要がないため，手軽に複雑な軌道計画などを行うことができる．

この記録の仕方は大きく二つある．

1. 目的の運動をさせた際の時間と姿勢 $(t, \boldsymbol{q})$ の時系列情報を記録することにより，記録した時の動きをそのまま再現する．
2. 目的の運動をさせた際の姿勢 $(\boldsymbol{q})$ の系列を記録し，各姿勢を再現するタイミングは別の方法で決定する．

前者は，決められた時間に目標姿勢を再現すれば良い．但し，記録の際に時間とタイミングよく提示しなければならないので難しい．一方，後者では，記録の際に時間を気にする必要がなく，**??**節のような方法を用いて再生すれば良い．時間に依存しないように記録する場合，大きく二つの方法が考える．一つは，記録するタイミングをスイッチのような外部からの指令によって記録する方法である．もうひとつは，連続的に動かす過程で状態を切り出す方法である．例えば，前に保存した状態 $\boldsymbol{q}_p$ と次の状態の間の距離がある κ より大きくなった場合，即ち，

$$\|\boldsymbol{q}_p - \boldsymbol{q}(t)\| > \kappa \tag{1.23}$$

を満足した時，この状態を関節角系列に加え，$\boldsymbol{q}_p = \boldsymbol{q}$ として，以降の比較をすればよい．

問題 1.3 次の問題を考えよ．

- 縦横比が 1:0.75 (または，4:3) の枠に一文字づつ，イニシャルのアルファベットを書け．
- 枠内のアルファベットを書くために，適切な中継点を設定せよ．また，枠の左下を原点とし，中継点の座標系を求めよ．
- アフィン変換を用いて，ロボットの可動範囲内にイニシャルを繰り返し 3 回隣り合わせに並べよ．その時の中継点の座標を求めよ．

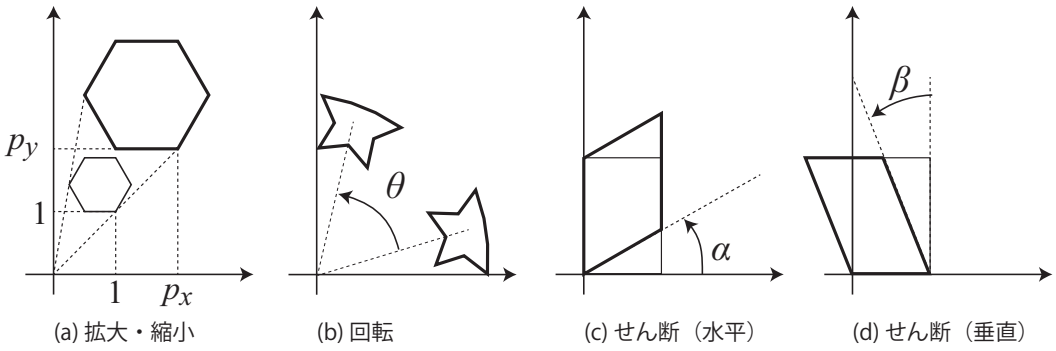

図 1.10: 図形の変形

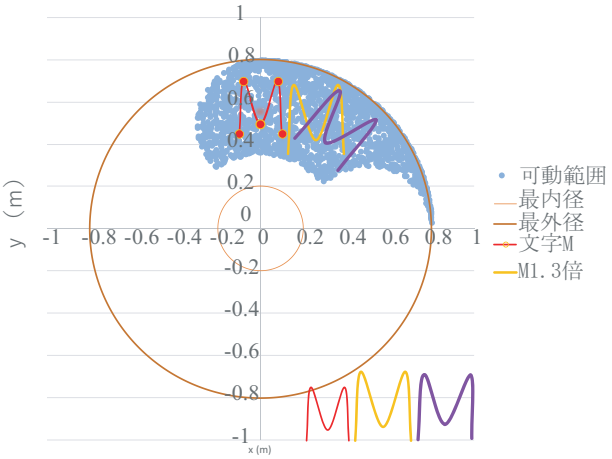

図 1.11: 手先空間での文字のアフィン変換

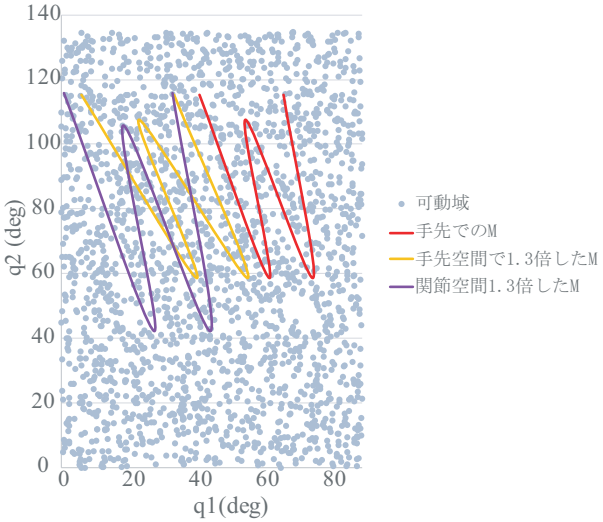

図 1.12: 関節空間でのアフィン変換

1.4.2　手先軌道の変形と移動

前節では，関節角の系列 $\boldsymbol{q}_i$ を記録・再生することで同じ軌道が生成できることを示した．次に，この軌道を再利用することを考える．例えば，ノートに文字や絵を描く場合，同じ文字を位置や大きさや姿勢を変

えて再現することが有る．また，ノーマル体をイタリックの体ような斜体で描く場合には，形を斜めに変形する．この実現方法を考える．

まず，記録した姿勢 $\boldsymbol{q}_i(i=0,1,\cdots)$ を再生する手先軌道 $\boldsymbol{x}_i$ を考える．これは，順運動学 (**??**) により次のように計算できる．

$$\boldsymbol{x}_i = \boldsymbol{f}(\boldsymbol{q}_i) \text{ for } i = 0, 1, \cdots, \tag{1.24}$$

これを繋いだ軌道が手先の連続軌道という形で与えられる．次に，大きさや形などを変換した後の図形の点 $\boldsymbol{y}_i$ とする．これは

$$\boldsymbol{y}_i = \boldsymbol{A}\boldsymbol{x}_i + \boldsymbol{b} \tag{1.25}$$

という変換をかければ良い．

ここで x, y の各軸方向にそれぞれ p_x, p_y 倍に拡大縮小したい場合は

$$\boldsymbol{A}_p = \begin{bmatrix} p_x & 0 \\ 0 & p_y \end{bmatrix} \tag{1.26}$$

となる (図 1.10(a) 参照)．$\boldsymbol{A}$ は回転，縮小拡大，剪断を表す行列であり，$\boldsymbol{b}$ は原点位置の移動を表す．例えば，2 次元の平面座標系の場合，θ 回転させたい場合，

$$\boldsymbol{A}_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{1.27}$$

と表す (図 1.10(b) 参照)．イタリック体のように文字を傾ける剪断による変換を行う場合，

$$\boldsymbol{A}_\alpha = \begin{bmatrix} 1 & -\tan\beta \\ \tan\alpha & 1 \end{bmatrix} \tag{1.28}$$

により表せる (図 1.10(c-d) 参照)．ここで α は x 軸を半時計回りに傾けた角度，β は y 軸を反時計回りに傾けた角度である．$\boldsymbol{A}$ は適切な順番で $\boldsymbol{A}_\theta, \boldsymbol{A}_p, \boldsymbol{A}_\alpha$ を掛けあわせればよい．例えば，剪断による変形をさせ，大きさを変えてから姿勢を変える場合，

$$\boldsymbol{A} = \boldsymbol{A}_\theta \boldsymbol{A}_p \boldsymbol{A}_\alpha \tag{1.29}$$

によって計算することができる．

図形をアフィン変換の際，変形の仕方 $\boldsymbol{A}$ は設計変数として決定される．一方，この変換は原点周りに行われるため，図形の位置が定まらない．そこで，ある特定の位置に図形を移動する場合，移動前と移動後の図形上で一致させる基準位置 $\boldsymbol{x}_c$ と移動後の位置 $\boldsymbol{y}_c$ を決め，平行移動量 $\boldsymbol{b}$ を決定する必要がある．これは，

$$\boldsymbol{b} = \boldsymbol{y}_c - \boldsymbol{A}\boldsymbol{x}_c \tag{1.30}$$

により求まる．よって，もし移動後の位置を決める基準位置 $\boldsymbol{x}_c$ を常に原点とすると，

$$\boldsymbol{b} = \boldsymbol{y}_c \tag{1.31}$$

となり，図形の変形と位置の移動を分離することができる．

図 1.11 は，赤い M とこれを 1.3 倍にしてすぐ横に並べた黄色の M を示している．これらの軌道を逆運動学によって関節空間へ写像したものが図 1.12 の赤と黄色の線になる．手先空間では大きさが変わっているにも関わらず，関節空間でのこれらの軌道は殆ど同じ大きさである．実際，関節空間で赤い軌道を 1.3 倍にした軌道が紫のような軌道であり，明らかに黄色の軌道は小さいことが分かる．

実際，紫の関節軌道を手先空間に戻した時，図 1.11 のように大きな回転が起こるが大きさは黄色の文字と変わらないことがわかる (右下に大きさの比較のために三つの文字を並べた)．即ち，記録した関節軌道を直接アフィン変換しても，手先空間の変換の意味が保存されず，軌道の再構成には適さない．そのため，一度手先空間に変換してから変換を扱う必要があることが分かる．

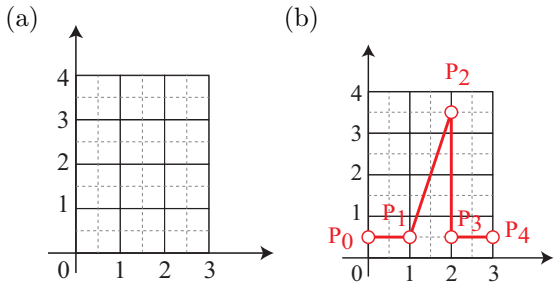

図 1.13: 文字の構成．(a) 文字を記述する枠．(b) 枠に記述した点．

1.4.3　図形の変形と再現

1.4.2 節では，図形の変形と移動の方法を学んだ．この方法を用いると予め用意しておいた図形や文字を変形し，任意の位置で欠かすことができる．

この方法を使い，制御を行うためのプロセスをまとめると以下のようになる．

1. 理想的な運動軌道の系列 $\boldsymbol{x}_i(i=0,1,\cdots)$ を記録する．($\boldsymbol{q}_i$ を記録し，順運動学 (??) で求めても良い．)
2. 各 $\boldsymbol{x}_i$ に対して，回転・剪断・拡大縮小を決定し $\boldsymbol{A}$ を決める．
3. 基準点を零とし，式 (1.31) により平行移動量 $\boldsymbol{b}$ を求める．(変換前後の基準点を零以外とし，式 (1.30) により求めても良い)．
4. 上記で求まった $\boldsymbol{A},\boldsymbol{b}$ を用いて変換 (1.25) を適用し，$\boldsymbol{y}_i$ を求める．
5. 逆運動学により，$\boldsymbol{y}_i$ に対応する関節角 $\boldsymbol{q}_i(i=0,1,\cdots)$ を求め，関節角制御を行う．

1.4.4　アフィン変換による図形の描写

文字を描かせるためには，図 1.13(a) のように基本となる枠をつくる．ここでは縦横が 4×3 の枠を用意する．この枠に描きたい図形の経由点を位置を決め，これを順番にたどればよい．図 1.13(b) では，P0 から P4 までの 5 点の座標 $(x_i,y_i)(i=0,1,2,3,4)$ を経由点として設定し，順番に辿ることで目的の字や図形を描くことができる．例えば，英単語を描きたい場合，全てのアルファベットをこの枠に合わせて作っておけば，それらを組み合わせることで文字を描ける．

次に，これらを用いて文字列をつくることを考える．図 1.14(a) は 4×12 の大きさの枠であり，先ほどの 4×3 の枠を 4 つ並べることができる．そこでこの枠に，図 1.14(b) のように，文字を並べることを考える．この場合，この文字はアフィン変換により，一文字分づつ右手に平行移動する必要がある．平行移動すると各文字の経由点の座標の値が全て同じだけ平行移動することに気をつける．この場合，平行移動する点距離は 3 であるので，全ての座標は (x_i,y_i) から (x_i+3i,y_i+3i) に移動することとなる．このようにして設定した文字の経由点は，文字列の全体の経由点集合としてひとまとまりに与える．

次に，文字列全体を回転させることを考える．これは文字列の経由点全体をアフィン変換により回転させることを意味する．

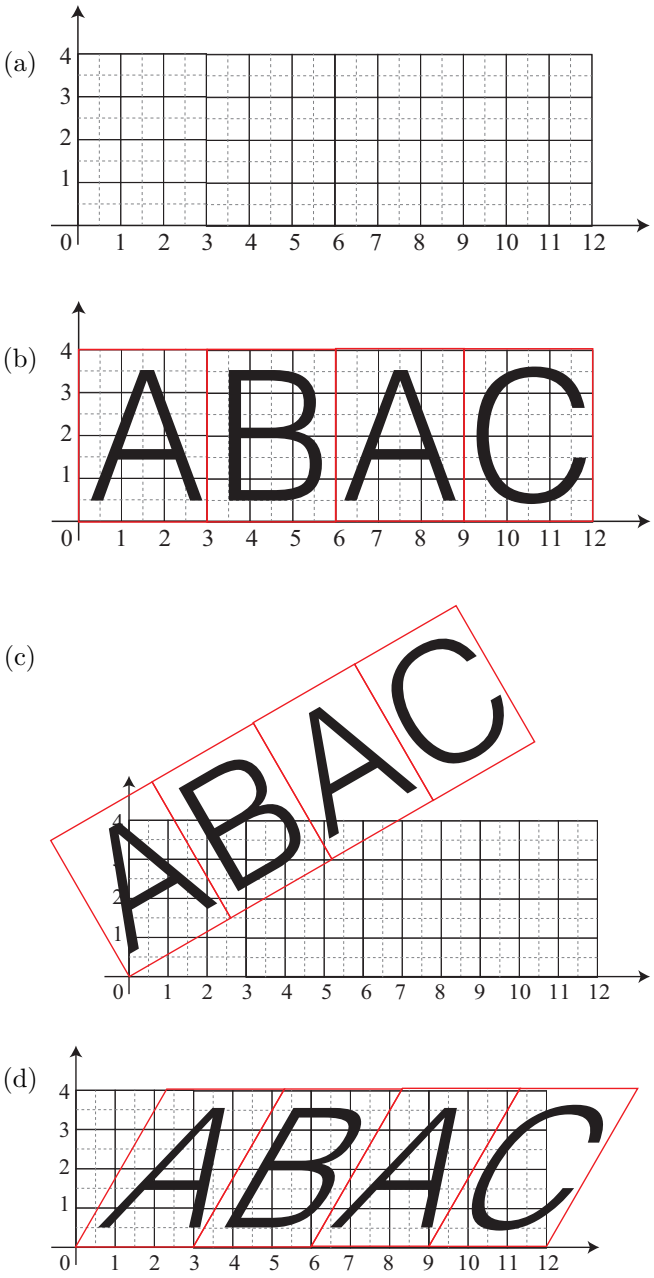

図 1.14: 文字列のアフィン変換 (a) フレームの構成．(b) 文字列の構成．(c) 文字列の回転．(d) 水平剪断によるイタリック体の構成．

付録 A　付録

A.1　ロボットプログラミング

A.1.1　Arduino IDE によるプログラミング

ロボットを制御するためのプログラムはマイコンの開発環境 Arduino IDE を用いて行う．

A.1.2　初期設定

下記のサンプルプログラムのように，プログラムはメイン関数 main() の代わりに次の二つの関数から構成されている．void setup() は，マイコンの電源を入れた際，最初に 1 回だけ呼び出される関数であり，主に初期設定を行うために用いられる．void loop() は，何度も繰り返し呼び出される関数であり，ここに実際のロボットの運動等を記述する．

```
#include <DynamixelSDK.h>
#include <Dynamixel_def_autumn.h>
#include <math.h>

#define DOF 2//関節数 (Degrees Of Freedom)
//可動範囲をRadian で設定しないと動かない
#define Q1MIN 0
#define Q1MAX 0
#define Q2MIN 0
#define Q2MAX 0

//リンク長
#define L1 0.068 //自分の値に変えること
#define L2 0.070 //自分の値に変えること

#define PLANE 2 //平面の自由度
#define MAX_POINTS_IN_WORDS 50 //想定する最大の経由点数(増やしても良い)

// Initialize PortHandler & PacketHandler instance
dynamixel::PortHandler *portHandler;
dynamixel::PacketHandler *packetHandler;

//Prototype Declaration of Affine Transformations(AT)
void AffineTransform(double A[PLANE][PLANE], double b[PLANE],
      double ch[][PLANE], int num_point, double word[][PLANE]);

// global variables
 int working_mode; //ロボットが制御中か、否か ;ON or OFF
/*********************************************************
```

```
30  Initial setup
31  ************************************************************/
32 void setup() {
33   portHandler = dynamixel::PortHandler::getPortHandler(DEVICENAME);
34   packetHandler = dynamixel::PacketHandler::getPacketHandler(PROTOCOL_VERSION);
35   Initialization(portHandler, packetHandler);//ON if it is normal.
36
37   working_mode = ON;
38   pinMode((int) BOARD_LED_PIN_LEFT,OUTPUT);
39   digitalWrite((int) BOARD_LED_PIN_LEFT,LED_ON);
40   delay(500);
41 }
42 /**********************************************************************
43  *
44  **********************************************************************/
45 void loop() {
46   int i,ch=0;
47   double th=0.0;
48   int num_points = 0, num_points_in_textbox=0;
49   double hight, width, enlarge;
50   double test[5][DOF]={{}};//test 用文字
51   double textbox1[MAX_POINTS_IN_WORDS][DOF]={{}}, textbox2[MAX_POINTS_IN_WORDS][DOF
         ]={{}};//endpoint trajectory
52   double drawpoints[MAX_POINTS_IN_WORDS][DOF]={{}};//endpoint trajectory
53   double Id[PLANE][PLANE]={{1.0, 0.0},{0.0, 1.0}}, zeros[PLANE]={0.0};
54   double A[PLANE][PLANE], b[PLANE];
55
56   if(working_mode != OFF){
57     //MENU
58     Serial.println("************* Command ***************");
59     Serial.println("e: Example");
60     Serial.println("c: Continuous Motion");
61     Serial.println("t: Drawing Textbox");
62     Serial.println("d: Drawing Transformed Textbox");
63     Serial.println("f: Free Drawing");
64     Serial.println("w: Workspace Drawing");
65     Serial.println("z: Zero Position");
66     Serial.println("q: Quit");
67     while(Serial.available()==0);
68     ch=Serial.read();
69
70     //Motion Selection
71     switch (ch){
72       /******* Example ******/
73       case 'e':
74         //ここにプログラムを書く
75         break;
76
77      /******* Continuous Motion ******/
78       case 'c':
79         //ここにプログラムを書く
```

```
80          break;
81
82         /******* Drawing a Textbook******/
83        case 't':
84          //ここにプログラムを書く
85          break;
86
87        /******* Drawing a Transformed Textbook******/
88        case 'd':
89          //ここにプログラムを書く
90          break;
91        /******* Free Drawing ******/
92        case 'f':
93          //ここにプログラムを書く
94          break;
95
96       /******* Workspace Drawing ******/
97        case 'w':
98          //ここにプログラムを書く
99          DrawingWorkspace(portHandler, packetHandler);
100         break;
101       /******* Zero Position *****/
102       case 'z':
103         GoToZeroPosition(1500, portHandler, packetHandler);//Go To Zero Position within
                1500 ms.
104         break;
105       /******* End motion *******/
106       case 'q':
107         GoToZeroPosition(1500, portHandler, packetHandler);
108         ClosePacketAndPort(portHandler, packetHandler);
109         digitalWrite((int)BOARD_LED_PIN_LEFT,LED_OFF);
110         working_mode = OFF;
111         break;
112       otherwise:
113         break;
114     }//end of switch()
115   }//end of if(working_mode)
116 }
117 /************************************************************
118 AffineTransform:アフィン変換
119 ch[][]の点にアフィン変換 Ax+b を施して、word[][]に保存する。
120 ************************************************************/
121 void AffineTransform(double A[PLANE][PLANE], double b[PLANE],
122                      double ch[][PLANE], int num_point, double word[][PLANE]){
123    int i,j,k;
124     for(i=0;i<num_point;i++){
125       //ここにプログラムを書く
126       }
127 }
128
129 }
```

モータに電流が流れて制御されているときは、マイコン上の青い LED が光る。Zero Position はロボットの初期姿勢 $\boldsymbol{q} = (0.0, 0.0)(\mathrm{rad})$ に移動する動作である．プログラムを終了する際は、必ず End motion を選択し、マイコン上の青い LED が消灯していることを確認すること。光ったままでコンパイルを行うと、マイコンが破壊される原因となる。

A.1.3 連続軌道問題

時刻 ts $\leq$ t $\leq$ te の間に xs[DOF] から xe[DOF] へ直線的に移動する 5 次補間軌道を完成させよ．

```
void continuous_motion(double t, double ts, double te, double xs[], double xe[], double
     x[]){
 double a[3] = {-10.0, 15.0, -6.0};
  double DeltaTn,Dt,beta;
  int i;
  if(t<ts){//範囲外 (t<ts)の時
    for(i=0;i<DOF;i++)
      x[i] = xs[i];
    }
  else if(t>te){//範囲外 (t>te)の時
    for(i=0;i<DOF;i++)
      x[i] = xe[i];
    }
  else{//範囲内 (ts<t<te)の時
    beta = 1.0;
    Dt;//ここに正しい式を書く。
    DeltaTn = Dt*Dt*Dt;
    for(i=0;i<3;i++){
      beta += a[i]*DeltaTn;
      DeltaTn *= Dt;
      }
    for(i=0;i<DOF;i++)
      x[i];//ここに正しい式を書く。
  }//else closed
}
```

A.1.4 手先位置の PTP 制御

num_point 個の手先軌道の集合 endpoint[num_point][2] と対応する通過時刻の集合 tset[num_point] が与えら得ているとする．この時，これらをつなぐ軌道を制御するための関数を作れ．但し，連続軌道は continuous_motion を使ってよい．

```
 void EndContinuousControl(double endpoint[][2], int num_points, double dt,dynamixel::
     PortHandler *port, dynamixel::PacketHandler *packet){
  int i, Ti;
  int Qd[DOF];
  double xd[DOF],qd[DOF];
  double t=0.0,ts=0.0, te=dt;
  Ti = millis();
  for(i=0;i<num_points-1;i++){
```

```
8     //Continuous Motion
9     while(t<te){
10      t= (double)(millis()-Ti)/1000.0;
11      continuous_motion(t, ts, te, endpoint[i], endpoint[i+1], xd);
12      InverseKinematics(xd,qd,LOWER_ELBOW);
13      Rad2Digital(qd, Qd);
14      setDigitalAngle(Qd, portHandler, packetHandler);
15      delay(10);
16      }
17      ts = te; te += dt;
18    }
19  }
20 }
```

A.2 関数

A.2.1 Arduino と math.h の標準関数

時間に関する関数

`delay(int t)`:
整数型で表される時間 `t`(ms) の間，次の命令を行うのを遅らせる．

`int millis()`:
時間を整数型で (ms) 単位で返す．

シリアルモニタへの送信

`Serial.print(a);//`改行無し
`Serial.println(a);//`改行有り
シリアルモニタへ a を送信して表示させる．a には，文字列や数字などが送れる．`println()` と `print()` の違いは a の後に改行の有るか，無いかである．

シリアルモニタからの受信

`int Serial.read()`:
送られてきたシリアルモニタのデータの最初の 1byte を取得する．

DIO ピンのモードの設定

`pinMode(int pin, int mode)`:
`pin` 番の DIO(digital Input Output) ピンの動作モード `mode` を設定する．`INPUT`，または `OUTPUT` に設定される．

DIO への値の書き込み

`digitalWrite(int pin, int value)`:
`OUTPUT` モードにあるピン `pin` に対して `value` の値を書き込む．値は `HIGH`, または `LOW` がある．

DIO の値の読み込み

`digitalRead(int pin)`:
`INPUT` モードにあるピン `pin` の値を関数の返り値として読む．返値は `HIGH`, または `LOW` である．

A.2.2 三角関数

`double sin(double x)`:
$\sin x$ の値を返す．

`double cos(double x)`:
$\cos x$ の値を返す．

`double acos(double x)`:
$\theta = \cos^{-1} x$ の値を返す．返り値の範囲は $0 \leq \theta \leq \pi$(rad).

`atan2(double y, double x);`
$\tan^{-1}(y/x)$ の値を返す．$\tan\theta$ は定義域 $-\pi/2 \leq \theta \leq \pi/2$ (rad) に対する値を求める関数である．そのため，$\tan\theta$ の逆関数である $\tan^{-1} z$ の値域は $-\pi/2 \leq \tan^{-1} z \leq \pi/2$(rad) として与えられる．この返値の値域が狭いため，`atan2(y,x)` として二つの引数を持つ関数とし，$y = a\sin\theta, x = a\cos\theta$ という性質を利用して値域を $-\pi \leq$`atan2(y,x)`$\leq \pi$(rad) と広げる．

A.2.3 実験用の独自関数

ポートやパケット等の初期化

`void Initialization(PortHandler *port, PacketHandler *packet);`
この関数は，モータと通信を行うために必要な PortHandler, PacketHandler, Serial 等の初期化を行う．

モータからの関節値取得

`void getDigitalAngle(int Qd[DOF], PortHandler *port, PacketHandler *packet);`
目標関節角度を表すディジタル値を Qd[DOF] をモータに与え，制御させる．

モータへの関節値を指令

`void setDigitalAngle(int Q[DOF], PortHandler *port, PacketHandler *packet);`
モータの現在の関節角度を表すディジタル値を Q[DOF] に取得する．

初期姿勢への復帰

`void GoToZeroPosition(int Ts, PortHandler *port, PacketHandler *packet);`
ロボットの初期姿勢 $\boldsymbol{q} = (0.0, 0.0)$(rad) に `Ts`(ms) 以内に移動する動作である．`Ts`(ms) は，初期姿勢への復帰命令後の待ち時間であり，実際に初期姿勢に戻るまでの時間と一致しないことに注意する．

実数データの表示

`void SerialPrintDataDouble(double z[]);`
`z[DOF]` は double 型の `DOF`(2) 個の要素を持つ 1 次元配列である．この 2 つの要素をシリアルモニタに次のように表示させる．

```
x[0]=0.1340; x[1]=4.5643;
```

もし先頭ラベルの `x` を変えた時は次のような関数を使えばよい．
`void SerialPrintDataDouble2(double z[], char ch[]);`
ここで `ch[]` はラベルの文字列であり，例えば，

```
qd[0]=0.1340; qd[1]=4.5643;
```

と表示させたい場合，`ch[]=''qd''` とすればよい．

ディジタル値の表示

`void SerialPrintDataInt(int Z[]);`
`Z[DOF]` は int 型の `DOF`(2) 個の要素を持つ 1 次元配列である．目標と実際のディジタル値 `Zd[DOF]` この 2 つの要素をシリアルモニタに表示させる．に表示させる．
実数データの時と同様にラベルをすげ替えたい場合，
`void SerialPrintDataInt2(int Z[], char ch[]);`
を代わりに用いればよい．

角度とディジタル値の変換

`int Rad2Digital(const double q[DOF], int Q[DOF]);`
関節角 `q[DOF]`(Rad) からモータへの命令で使うディジタル値 `Q[DOF]` を計算する．

`int Digital2Rad(const int Q[DOF], double q[DOF])`:
モータへの命令で使うディジタル値 `Q[DOF]` から関節角 `q[DOF]`(Rad) を計算する．