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．

現在，世界で最も使われているロボットは工場などで見られる産業用のロボットアームである．このようなロボットを動かす際には環境認識や運動計画，制御等に関する様々な知識を統合する必要がある．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，これらのロボットの制御技術の一つである逆運動学に基づく手先位置制御法について学習する．

1.1 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 1.1 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．

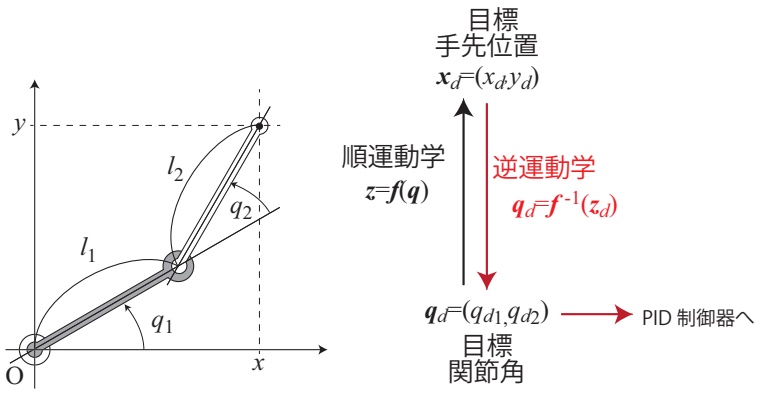


図 1.1: 順運動学と逆運動学による制御

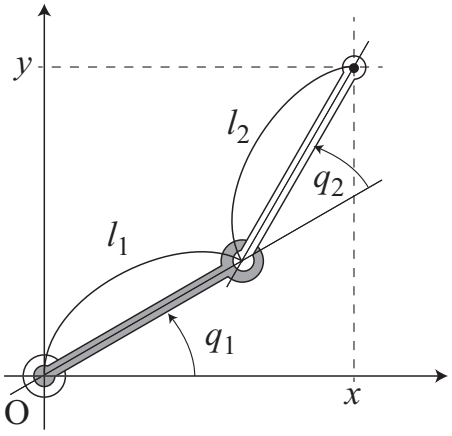


図 1.2: 平面を動く 2 自由度回転関節型ロボットアーム

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．

1.2 ロボットの運動学と可動範囲

1.2.1 順運動学と作業空間

ロボットアームの関節 $\boldsymbol{q}$ に対して，手先位置 $\boldsymbol{x}$ が一意に決定される．この関係を順運動学といい，次のように表すことができる．

$$\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{q}) \tag{1.1}$$

図 1.2 のように，リンクが一列に連なるシリアルマニピュレータの順運動学は解析的に求まることが知られている *．例えば，この 2 自由度回転関節型ロボットアームの場合，順運動学は次のように求まる．

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix} = \boldsymbol{f}(\boldsymbol{q}) = \begin{bmatrix} l_1 \cos q_1 + l_2 \cos(q_1 + q_2) \\ l_1 \sin q_1 + l_2 \sin(q_1 + q_2) \end{bmatrix}, \tag{1.2}$$

ここで，関節角 $\boldsymbol{q} = (q_1, q_2)$ であり，$l_i (i = 1, 2)$ はロボットアームのリンク長である．図 1.3 は，このアームの手先に描かせた円軌道 (青) とその手先軌道に対応する関節軌道 (黄) を重ねて描かせたものである．このように，手先から関節への写像はその位置が大きく変わり，形も歪むため，手先から関節を直感的には予

*一方，複数のリンクが遠位のリンクにつながったパラレルマニピュレータの場合，順運動学の解析解が存在しない場合がある．

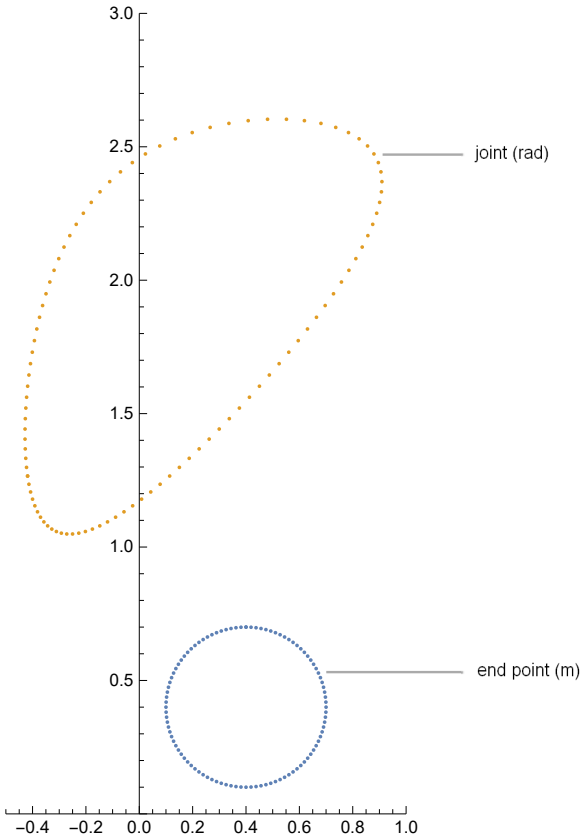


図 1.3: 関節軌道と手先軌道

測できない．そのため，順運動学により関節と手先の動きの幾何学的な対応関係を正確に捉えることが重要となる．

ロボットアームが目的の運動を確実に実現するためには，まずアームが到達しうる手先の「作業空間」を把握する必要がある．この作業空間は，取りうる全ての関節角 $\boldsymbol{q}$ に対する手先位置 $\boldsymbol{x}$ が明示できれば良い．図 1.4(a) は，取りうる関節角 $\boldsymbol{q}$ を乱数で発生させ (左図の点)，それに対応する手先位置を記録したもの (右図) である．このように，十分多くの数の $\boldsymbol{q}$ の点を発生させれば，それに対応する手先位置の取りうる領域の形を明示できる．

今，図 1.2 のロボットが，$l_1 > l_2$，かつ $0 \leq q_1 \leq 2\pi$ (rad), $0 \leq q_2 \leq \pi$ (rad) の範囲で動くとする．このロボットは回転関節を持つため，図 1.4(a) のように，一番手先を伸ばした状態と縮めた状態に囲まれた，原点周りのドーナツ型の領域に到達できることが分かる．

しかしながら，実際のロボットの場合，リンクとリンクの干渉やモータの能力などにより，関節の可動範囲が次のように制限される．

$$q_{i\min} \leq q_i \leq q_{i\max} \ (i = 1, 2) \tag{1.3}$$

ここで，$q_{i\min}(q_{i\max})$ は第 i 関節の最小 (最大) の関節角である．そのため，実際の作業空間はドーナツ型の領域の一部，例えば，関節と手先の作業空間は図 1.4(b) の赤い領域になる．

この関節角の領域と手先位置の領域の間の写像は，局所的には，位相構造 (角や辺の数が同じであったり，領域の中の点は必ず写像した後も領域の中の点である等) を保つ †．例えば，図 1.4(b) の赤い関節角の長方形領域の縁を動いた時にできる軌道は，赤い手先位置軌道の作業空間の縁を表す．つまり，ロボットアーム

†大域的には位相構造が保たれるとは限らない．

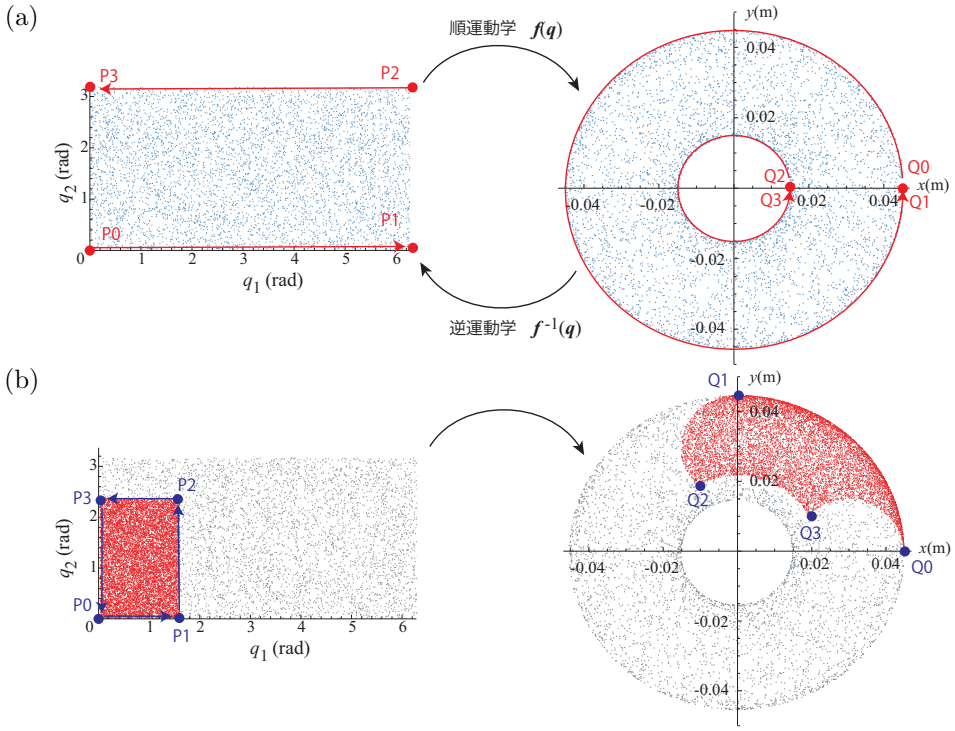


図 1.4: 2 自由度ロボットの関節の可動範囲と手先の作業空間: (a) 関節の全可動範囲からの写像. 左の関節空間の位置が点 Pi である時，右の手先位置は点 Qi に相当する．P0→P1 と P2→P3 を辿る線は，可動域のドーナツ型の領域の外側と内側の線に写像される．このように，関節空間の可動範囲の縁を辿ると，作業空間の全体像が見える．(b) 関節の可動範囲の制限を考慮した場合．左の赤の領域は制限された関節の可動範囲であり，右の赤い領域がそれに対応するロボットアームの作業空間である．

の関節角をある閉領域の縁に沿って動かしたとき，手先も対応する閉領域の縁を移動する．そのため，うまく閉領域を設定し，その縁を辿る動きを記録することができれば，作業空間の外形を把握できる．

1.2.2　一筆書き問題による作業空間の可視化

図 1.2 のロボットアームの作業空間を閉曲線によって囲むためには，一筆書きを利用すると良い．一筆書きは大きく二つの条件がある．

条件 1 筆記用具を平面から一度も離さず線図形を書くこと

条件 2 同じ線をなぞらないこと

広義の意味では**条件 1** のみであり，狭義の意味では**条件 1,2** の両方を満足する必要がある．

一筆書きに関する非常に有名な問題として「ケーニヒスベルクの橋問題」がある．ケーニヒスベルクは，18 世紀頃に存在したプロイセン王国の首都であり，図 1.5 のように，そこに流れていたプレーゲル川の川岸と中洲の間に 7 つの橋がかけられていた．この問題は，全ての橋を通り元の位置に戻れるかを問うものであった．但し，同じ橋を渡れるのは 1 度きりであり，どこから出発しても良いという条件がつく．

1786 年にレオンハルト・オイラーは，橋の集まる点を頂点，橋を頂点を結ぶ線とした連結グラフ ‡ で表し (図 1.5)，この問題を否定的に解決した．この時見つかった一筆書きの条件は以下のようになる．

‡グラフは，ノード (節点・頂点) の集合とエッジ (枝・辺) の集合で構成される．連結グラフは，任意の二点のノード間に道が存在するグラフである．

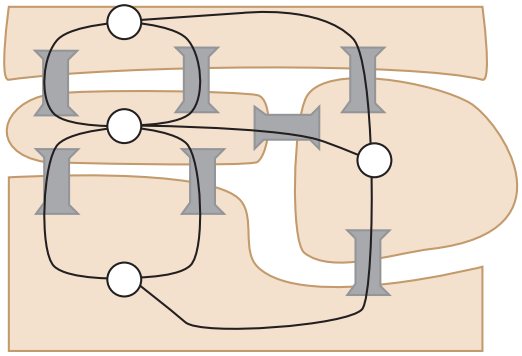


図 1.5: ケーニヒスベルグの橋問題

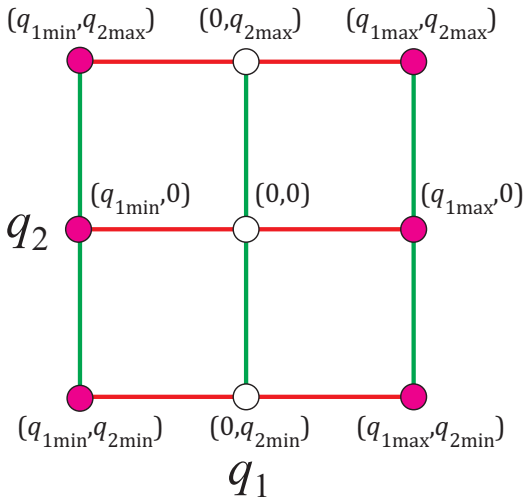


図 1.6: ロボットの作業空間を縁取るために重要な点と線から構成される連結グラフ

ある連結グラフが一筆書き可能となる必要十分条件は，以下のいずれか一方が成立することである．

1. 全ての頂点の次数が偶数
2. 次数が奇数である頂点が 2 点で，残り全ての頂点の次数が偶数

この方法をロボットアームの作業空間を描くのに利用する．ロボットアームの取る姿勢 $\boldsymbol{q} = (q_1, q_2)$ に対して，第 i 関節が最も腕を伸ばせる姿勢 0(rad) と最小角 $q_{i\min}(\leq 0(\text{rad}))$・最大角 $q_{i\max}(\geq 0(\text{rad}))$ をとる場合を頂点とする．$q_{i\min}$ から $q_{i\max}$ へは 0(deg) を経由しなければならないことに注意する．この時，各姿勢を結ぶ連結グラフは図 1.6 のようになる．関節でのノード間の境界線は手先の空間に射影した際には局所的に外部との境界を表すが，大域的には各境界線が領域の内部に含まれる場合もある．しかしながら，全ての境界線を辿るようにこの連結グラフをうまく一筆書きできれば，ロボットアームの作業空間を縁取ることができる．しかしながら，この連結グラフは，次数 4 である頂点が 1 つ，次数が 3 である頂点が 4 つ，次数が 2 である頂点が 4 つあり，一筆書きができないことが分かる．

一方，図 1.6 に対応する作業空間を表した図 1.7 を見ると，$q_1 = 0$ (rad) となる頂点 (0,0)，$(0, q_{2\min})$，$(0, q_{2\max})$ を結ぶ緑の線は作業空間の内部にあり，作業空間を縁どるには無用の点であることが分かる．これらの頂点を連結グラフから除くと，次数が 3 である頂点が 2 つ，次数が 2 である頂点が 4 つとなり，一筆書きが可能となる．そこで，次数 3 の頂点 $(q_{1\min}, 0)$, $(q_{1\max}, 0)$ のいずれかに最初に移動して，そこから

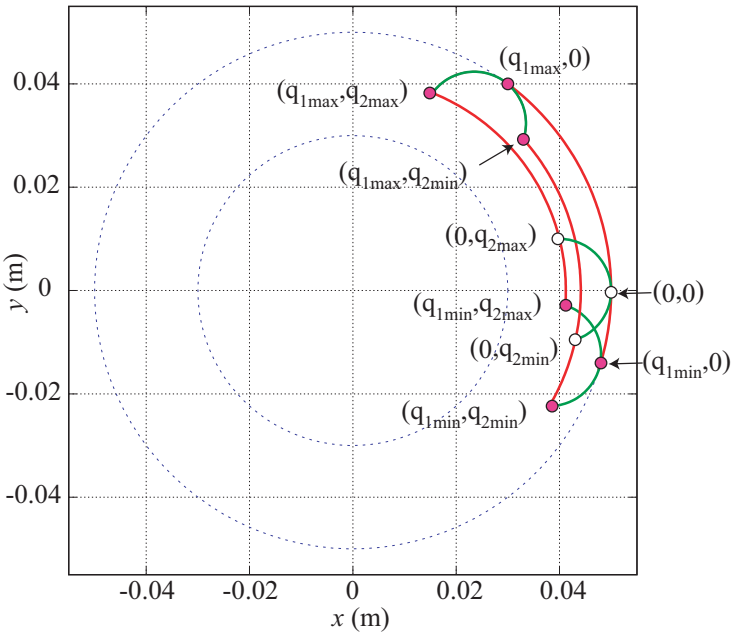


図 1.7: ロボットの作業空間の一例

一筆書きを始めると作業空間の外形を辿れることが分かる．

更に，第 2 関節の可動範囲が 0 (rad) をまたがない場合，図 1.6 の幾つかの頂点が消え，一筆書き問題が更に単純化できる．

問題 1.1 作業空間の一筆書き問題について考える．

1. $0 \leq q_{2\min} \leq q_2 \leq q_{2\max}$, または $q_{2\min} \leq q_2 \leq q_{2\max} \leq 0$ の場合，図 1.6 のどこの頂点が消えるか考え，図 1.6 の連結グラフを描きなおせ．
2. 上記の場合，どのように姿勢をたどれば一筆書きができるか．通過する頂点を順番通りに並べよ．

1.3 ロボットの逆運動学問題

1.3.1 2 自由度ロボットにおける厳密な解法

ロボットアームに目的の運動をさせるためには手先位置 $\boldsymbol{x}$ の運動計画を行う必要がある．一方，手先位置 $\boldsymbol{x}$ は通常ロボットの関節に取り付けたモータにより間接的に制御しなければならない．

手先位置 $\boldsymbol{x}$ に対応する関節角 $\boldsymbol{q}$ を求める方法は逆運動学と言われ，

$$\boldsymbol{q} = \boldsymbol{f}^{-1}(\boldsymbol{x}) \tag{1.4}$$

により表すことができる．順運動学とは違い，逆運動学の解析解は一般に存在しない．しかしながら，図 1.2 のような，平面 2 自由度アームの場合，特異点[§] を除き解析解が存在することが知られている．但し，2 自由度アームの手先位置 $\boldsymbol{x} = (x, y)$ に対して，肘が上，もしくは下につきだした二つの姿勢が存在する．

[§] 他と性質の異なる点．ロボットの場合，局所的にある方向には運動ができなくなる点を指す．

下に肘が出た姿勢の場合

図 1.8(a) のように，下側の姿勢を取る場合を考える．余弦定理から $\gamma(\angle\mathrm{ODB})$ は，

$$\gamma = \cos^{-1}\left(\frac{l_1^2 + l_2^2 - \left(x^2 + y^2\right)}{2l_1 l_2}\right) \text{ (rad)} \tag{1.5}$$

となる．この時，γ を用いることで，関節角 q_i は，

$$q_2 = \pi - \gamma \text{ (rad)} \tag{1.6}$$

$$q_1 = \tan^{-1}\left(\frac{y}{x}\right) - \tan^{-1}\left(\frac{l_2 \sin q_2}{l_1 + l_2 \cos q_2}\right) \text{ (rad)} \tag{1.7}$$

と求めることができる．q_1 は $\angle\mathrm{AOB}$ から $\angle\mathrm{COB}$ を引くことによって求めた．

上に肘が出た姿勢の場合

図 1.8(b) のように，上側の姿勢を取る時も γ' の値から，同様な方法を用いることによって求まる．この場合の内角 $\angle\mathrm{ODB}$ を γ' とすると，

$$\gamma' = \cos^{-1}\left(\frac{l_1^2 + l_2^2 - \left(x^2 + y^2\right)}{2l_1 l_2}\right) \text{ (rad)} \tag{1.8}$$

となる．また，

$$q_2 = -\pi + \gamma' (= \pi + \gamma') \text{ (rad)} \tag{1.9}$$

$$q_1 = \tan^{-1}\left(\frac{y}{x}\right) - \tan^{-1}\left(\frac{l_2 \sin q_2}{l_1 + l_2 \cos q_2}\right) \text{ (rad)} \tag{1.10}$$

である．

二つの姿勢を見ると非常に似ている．そこで，肘の姿勢の違いによる逆運動学の表現を統一化する方法を考えるために，二つの逆運動学を比較してみる．まず，式 (1.5) と (1.8) から $\gamma = \gamma'$ であり，式 (1.7) と (1.10) から，q_1 も等しい．一方，q_2 を求める際，式 (1.6) と式 (1.9) の γ と γ' の符号が逆である．このことから，上側の姿勢を取る際には式 (1.6) の方程式の $\gamma = -\gamma'$ と置き換えれば，上下どちらの姿勢でも式 (1.6) と (1.7) が共通に使えることが分かる．

1.3.2 逆運動学の計算上の工夫

関節の値域の変換

手先の全作業領域から 10,000 点の位置を発生させ，逆運動学を使って求めた関節角を表示させたものを図 1.9(a) である．q_2 の関節角は肘が下の姿勢 (上の姿勢) の場合，$0 \le q_2 \le \pi(\mathrm{rad})$ $(-\pi \le q_2 \le 0(\mathrm{rad}))$ の範囲にあり，理想通りの領域になっている．

一方，q_1 の方はどの q_2 に対しても可動域の幅が $2\pi(\mathrm{rad})$ あるが，その領域の位置が移動して使いづらい．そこで，式 (1.6)，もしくは式 (1.9) によって得られた解を

$$q_1 = \begin{cases} q_1 + 2\pi(\mathrm{rad}) & \text{if } q_1 < q_{1\min} \\ q_1 & \text{otherwise} \end{cases} \tag{1.11}$$

によって変換してやると，$q_{1\min} \le q_1 \le 2\pi + q_{1\min}$ (rad) にすることができる．図 1.9(b) は，$q_{1\min} = -\pi$ (rad) に設定した時の関節の可動域を表している．これから $-\pi \le q_1 \le \pi$ (rad) に設定できていることが分かる．

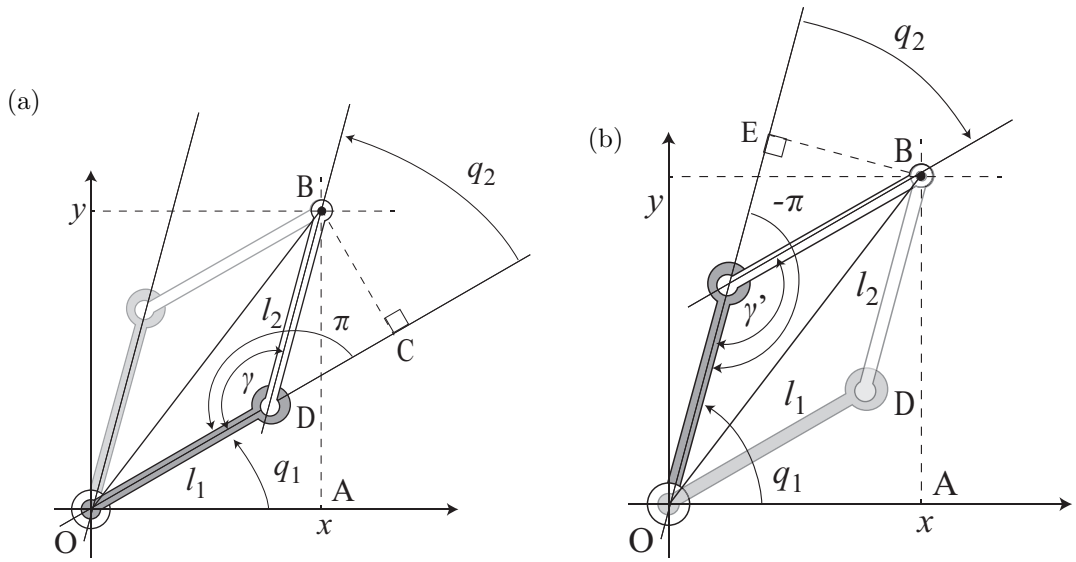


図 1.8: 2 自由度ロボットの逆運動学の解析解の導出 (a) 下側のアームの姿勢の場合．(b) 上側のアームの姿勢の場合．

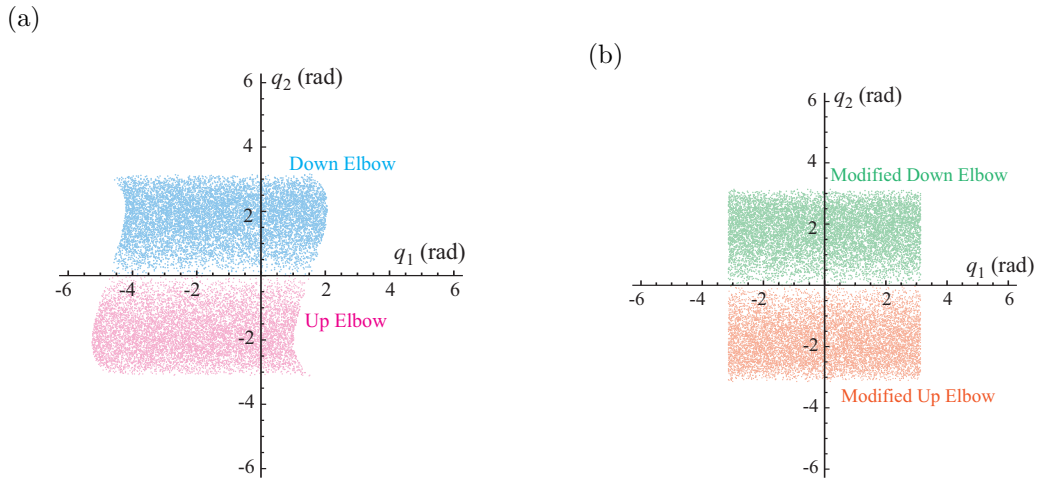


図 1.9: 2 自由度ロボットの逆運動学の解析解の分布 (a) そのままの場合．(b)q_1 に補正を加えた場合．

不可能解の判定

逆運動学は (x, y) が適切な範囲で与えられているときはこのような問題は考える必要はない．一方，任意の x, y に対して解が存在するわけではない．もし x, y が手先が届かない範囲に与えられた場合，それを実現する関節角は存在しないので，逆運動学が解けない．そこでこの判定を行う必要がある．

判定は 2 段階ある．一つは理想的なロボットが届く範囲の判定，もう一つは現在の可動域内での解の存在性の判定である．最初の判定は，1.3.1 節のような厳密解が存在するかの判定であり，不可能な場合，計算結果が解の不安定性を引き起こす．これは，逆運動学を解く前に行う．2 自由度アームの場合，図 1.9(a) のようなドーナツの内側にあるかの判定が必要となる．そのために，外の円形の内側に存在する条件として

$$x^2 + y^2 < l_1^2 + l_2^2 \tag{1.12}$$

が必要となる．$l_1 \geq l_2$ の場合，これでおわりである．一方，$l_1 < l_2$ の場合，ドーナツの穴に入っていない

ことも判定する必要がある．これは，

$$(l_1 - l_2)^2 < x^2 + y^2 \tag{1.13}$$

を満足すればよい．

もうひとつの条件は，逆運動学の解が可動範囲 (1.3) に存在するかどうかである．これは，上記の条件を満足した後に，逆運動学を解き，その上でその解が関節の可動範囲に収まっているかを判定すればよい．以上をまとめると次のようになる．

1. 式 (1.12) を満足するか判定．満足しない場合，解が存在しない．
2. $l_1 < l_2$ の場合，式 (1.13) を満足するか判定．満足しない場合，解が存在しない．
3. 条件 1,2 を満足した場合，逆運動学を解く．式 (1.11) による変換も行う．
4. 逆運動学の解が式 (1.3) を満足するか判定．満足する場合，それが実行可能解となる．満足しない場合は制御できない．

1.4 ロボットの制御システム

図 1.10 にロボットの制御システムを示す．典型的なモータの制御システムでは，コンピュータの中にモータの制御器を構成し，AD, DA 変換器を通してモータにから情報を受けとり，指令を与える．典型的には，制御器は，関節角の PID 制御系が用いられ，関節角の誤差値 $\Delta \boldsymbol{q} = \boldsymbol{q} - \boldsymbol{q}_d$ に対して，制御器の生成する目標トルクは

$$\boldsymbol{\tau} = -\boldsymbol{K}_P \Delta \boldsymbol{q} - \boldsymbol{K}_D \Delta \dot{\boldsymbol{q}} - \boldsymbol{K}_I \int_0^t \Delta \boldsymbol{q} \mathrm{d}t \tag{1.14}$$

となる．ここで $\boldsymbol{K}_P, \boldsymbol{K}_D, \boldsymbol{K}_I$ はそれぞれ比例，微分，積分誤差に対するフィードバックゲイン行列である．

今回実験で使うモータは，モータ内部にマイコンが搭載してあり，これが典型的な制御器で行う緑色の部分に相当する行う．そのため，実際に使う際には，モータの目標角を与えてあげれば，目標トルクや AD, DA 変換などは自動的に行ってくれる．但し，目標関節角をディジタル値として与える必要がある．

図 1.11 は，このモータと指令を出すコンピュータの実際の制御の状況を表している．モータとのデータのやりとりは，シリアルポートを通じたパケット通信にて行う．そのデータのやりとりを行うための初期設定は `void setup()` の関数に全て書かれている．

そのため，実際には，`getDiscreteAngle()`, `setDiscreteAngle()` の二つのコマンドを用いるとデータのやりとりができる．コンピュータが受け取ったデジタル値は，`Discrete2Rad()`, `Rad2Discrete()` という関数により，実際の関節角との間の変換をする．関節角と手先位置の変換には `DirectKinematics()`, `InverseKinematics()` という関数を用いる．

1.5 ロボットの組み立て

図 1.12 は本実験に用いるロボットアームである．このロボットアームは，二つのアクチュエータと制御用のボードおよびリンク部分から構成されている．アクチュエータは ROBOTIS 社製の XL430-W250T を使用しており，各アクチュエータと制御器をシリアルに接続することで制御が簡単に行える．アクチュエータにはデジタル制御器が搭載されており，位置制御モードと速度制御モードが利用できる．各モードでは，目標 (角度または速度) を与えるとそれに追従することができる．本実験では，目標関節角度 (モータ角度と一緒) を適切に与えることで，ロボットの手先位置を制御する．

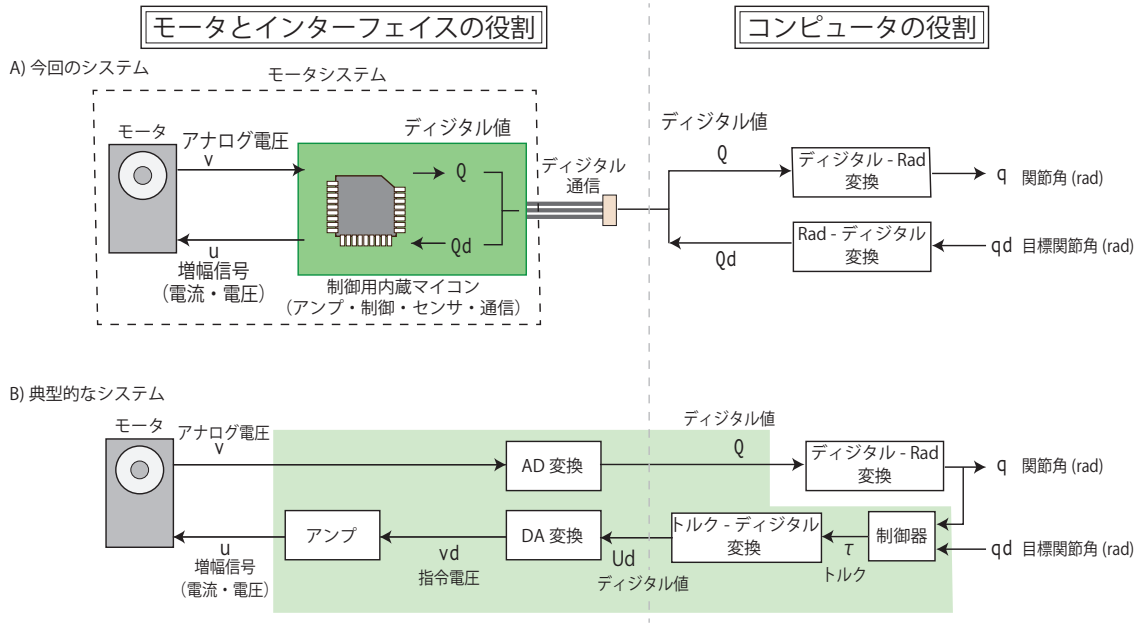


図 1.10: モータの制御システムの基本的な構造

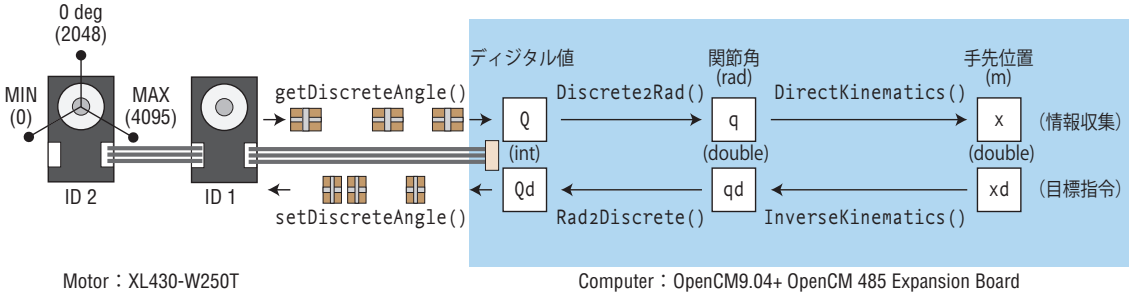


図 1.11: モータの通信とデータの変換

図 1.13 にあるように，位置制御を利用した場合，モータには 12 ビット (0〜4,095) のディジタル値で指令を送ることで角度を制御することができる．初期姿勢 0° の時のディジタル値は 2,048 であり，1 ディジタル値当たり 0.088° の大きさで角度が変わる．

問題 1.2 次の計算式を求めよ．

- 目標角度 q_d(rad) が与えられた時，これに相当するディジタル値 Q_d を求める方程式を求めよ．
- ディジタル値 Q が与えられた時，角度 q(rad) に変換する式を求めよ．

ロボットは，モータやリンクがバラバラの状態から各自が組み立てる．各リンクやモータの接続は日本建築に触発された継ぎ手構造によって構成されており，簡単に組み立てることができる．また，長さの違う複数のリンクが用意されており，実験者はそのリンクを変えることで，違う構造を持つロボットアームを構成することができる．

ここで使われている継ぎ手構造は尻挟み (しっぱさみ) 継ぎと言われ，古い木造建築物で使われる構造である．図 1.14 にあるように，尻挟み継ぎは二つの同じ構造をした構造を点対称に配置し，2 段階でスライドさせ，組み合わせた後にできる隙間に栓をすることで繋ぎ合わすことができる．分解する際には，栓を外

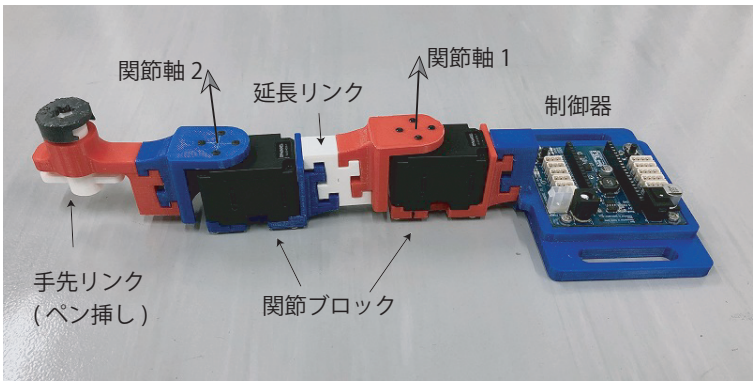


図 1.12: 実験用ロボットアーム

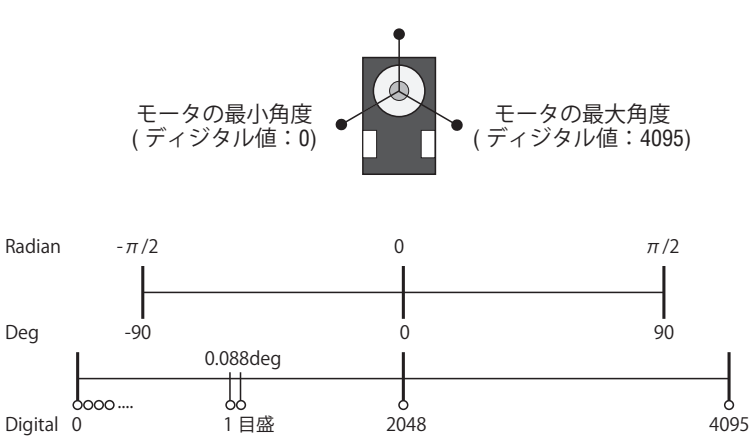


図 1.13: モータの角度変換

して逆の手順でスライドさせれば簡単に外れる.

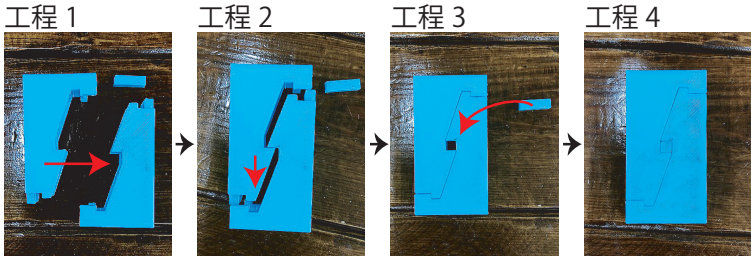
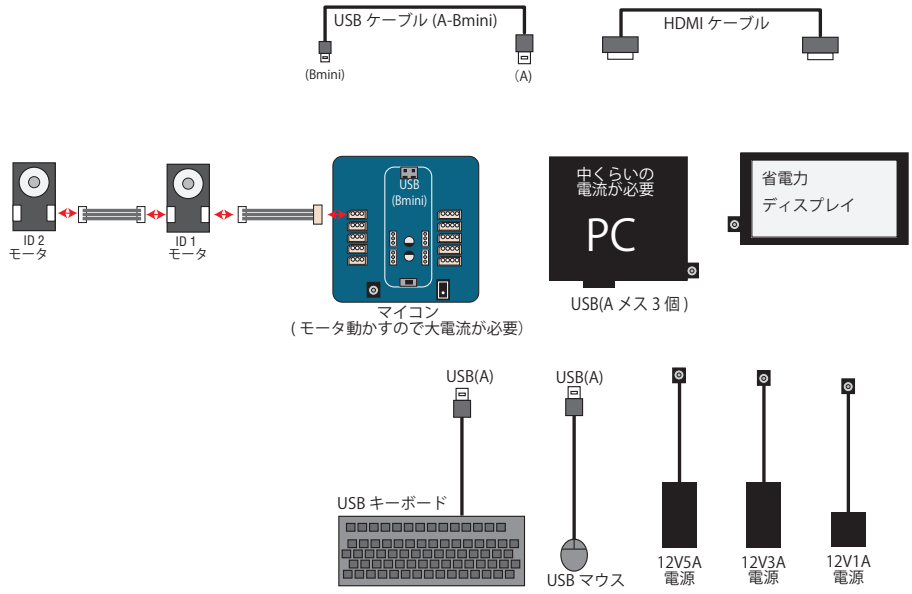


図 1.14: 尻挟み継ぎの構造

問題 1.3

- 下図の実験装置を適切に配線せよ.
- 教員の指示に従い, 適切なリンクを選べ.
- ロボットのリンクの長さ L1，L2 がどのくらいになるか計測せよ.



1.6 実験課題

本実験では，以下の課題を解くことでロボットの逆運動学に基づくロボットの制御方法を学んでいく.

課題 0

`case 't':`にモータの角度を制御するプログラムを書け．但し，以下の条件を満足するものとする.

- モータ 1,2 の目標をそれぞれ 3000,1500 にすること.
- モータ角度はあらかじめ用意した変数 `int Qd[DOF]` に与えること.

課題 1 `case 't':`

1. Radian をディジタル値に変換する計算式
   `void Discrete2Rad(const int Q[], double q[]);`
   のプログラムを書き，第 1,2 関節角 `q[i]`，`(i=0,1)` をそれぞれ, $\pi/3, -\pi/3$(rad) に制御せよ．プログラムを書く際は，変数の型変換に気をつけること．

2. ディジタル値を Radian に変換する計算式
   `void Rad2Discrete(const double q[], int Q[]);`
   のプログラムを書け. 上記の関数を使い，目標と実際の関節角 `qd[]`，`q[]` の比較結果をシリアルモニタに表示すること．但し，データの表示には,
   `void SerialPrintDataDouble(double x[])`
   もしくは,
   `void SerialPrintDataDouble(double x[], char ch[]);`
   を用いよ．

課題 2 `case 'w':`
関節の可動範囲を各個人の指定されたものに変え，作業空間を描かせよ．

- 順次辿る姿勢角は一筆書きの順番で指定すること．
- N 個の点を辿る場合，2 次元配列 `double qset[N][DOF]` を用いればよい.
- 例えば，`func (double q[])` に 0 番目の点を与える場合，`func(qset[0])` と書けばよい.
- 最後に原点に戻るようにせよ．
- うまく動作するようになったら，手先にラッションペンをつけて実際の作業空間を描かせること．

課題 3

1. ロボットの関節角と手先位置の関係を決める逆運動学の関数
   `void InverseKinematics(const double x[], dobule q[]);`
   のプログラムを書き，作業空間の中央近辺に適当な 2 点 `x[]` を設定し，その点にロボットの手先が辿るように制御せよ．

2. ロボットの関節角と手先位置の関係を決める順運動学の関数
   `void DirectKinematics(const double q[], dobule x[]);`
   を書き，目標と実際の手先位置 `xd[]`，`x[]` の比較結果をシリアルモニタに表示すること．

3. ロボットの手先を辿る複雑な図形 (5 点以上) を作業空間内に設定し，絵や字を描かせよ．

- 肘の形が Upper Configuration か，Lower Configuration か考えよ．
- 逆運動学の下位の範囲を $\pm\pi$(rad) にせよ．
- $\cos^{-1} x$ は `acos(x)`, $\tan^{-1}(y/x)$ は，`atan2(y,x)` という数学関数 (math.h) を使う．

付録 A　付録

A.1　ロボットプログラミング

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．下記のサンプルプログラムのように，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，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.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

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

//extern 宣言
void Discrete2Rad(int Q[], double q[]);
void Rad2Discrete(double q[], int Q[]);
void DirectKinematics(double q[], double x[]);
void InverseKinematics(double x[], double q[]);

// global variables
 int working_mode; //ロボットが制御中か、否か ;ON or OFF
/*************************************************************
 Initial setup
 *************************************************************/
void setup() {
  portHandler = dynamixel::PortHandler::getPortHandler(DEVICENAME);
  packetHandler = dynamixel::PacketHandler::getPacketHandler(PROTOCOL_VERSION);
  Initialization(portHandler, packetHandler);//ON if it is normal.

  working_mode = ON;
  pinMode((int) BOARD_LED_PIN_LEFT,OUTPUT);
  digitalWrite((int) BOARD_LED_PIN_LEFT,LED_ON);
  delay(500);
}
/*********************************************************************
```

```
37 loop 関数
38 ********************************************************************************/
39 void loop() {
40   int i,ch=0;
41   int Q[DOF]={2048},Qd[DOF];
42   double q[DOF]={0.0}, qd[DOF];
43   double qset[10][DOF]={{}};//joint trajectory
44   double xset[10][DOF]={{}};//endpoint trajectory
45
46   if(working_mode != OFF){
47     //MENU
48     Serial.print("************* Command *************** \n");
49     Serial.print("t: Test\n");
50     Serial.print("w: Workspace Drawing\n");
51     Serial.print("d: Drawing\n");
52     Serial.print("z: Zero Position\n");
53     Serial.print("q: Quit\n");
54     while(Serial.available()==0);
55     ch=Serial.read();
56
57     //Motion Selection
58     switch (ch){
59       /******* Workspace Drawing ******/
60       case 't':
61         //ここにプログラムを書く
62         break;
63
64       /******* Workspace Drawing ******/
65       case 'w':
66         //ここにプログラムを書く
67         break;
68
69      /******* Workspace Drawing ******/
70       case 'd'://Drawing
71         //ここにプログラムを書く
72         break;
73
74       /******* Zero Position ******/
75       case 'z':  //Go To Zero Position within 1500 ms.
76           GoToZeroPosition(1500, portHandler, packetHandler);
77         break;
78
79       /******* End motion *******/
80       case 'q':
81           GoToZeroPosition(1500, portHandler, packetHandler);
82           ClosePacketAndPort(portHandler, packetHandler);
83           digitalWrite((int)BOARD_LED_PIN_LEFT,LED_OFF);
84           working_mode = OFF;
85         break;
86       otherwise:
87         break;
```

```
88      }//end of switch()
89    }//end of if(working_mode)
90  }
91  /*********************************************************************
92    (ディジタル値 -> Rad)
93   *********************************************************************/
94  void Discrete2Rad(int Q[], double q[]){
95    int i;
96    //ここにプログラムを書く
97  }
98  /*********************************************************************
99  (Radian -> ディジタル値)
100  *********************************************************************/
101 void Rad2Discrete(double q[], int Q[]){
102   int i;
103   //ここにプログラムを書く
104 }
105 /*************************************************************
106  Analytical Solution of the direct kinematics
107  *************************************************************/
108 void DirectKinematics(double q[], double x[]){
109   //ここにプログラムを書く
110 }
111 /*************************************************************
112  Analytical Solution of the inverse kinematics
113  *************************************************************/
114 void InverseKinematics(double x[], double q[]){
115   double gamma;
116   //ここにプログラムを書く
117 }
```

working_mode=ON は，モータ角度が制御されている状態を表し，ボード上に青い LED が点灯する．このモードでは，モータを自由に動かすことが出来ない．一方，working_mode=OFF ではモータへの指令が終了となり，ボード上の青い LED が消灯する．モータは自由に動かせる状態になる．

49-53 行目は，モータの制御モードの選択を促すメニューである．これは，制御モードである時，毎回冒頭にシリアルモニタに表示される．このモードの選択は 54，55 行目にて行われる．上記のモードに相当するアルファベットをシリアルモニタから ch に入力すればよい．例えば，t を入力した場合，49 行目で表示するメニューは 60-62 行目のプログラムに相当する部分に対応している．以下，同じように動く．

A.2 関数

A.2.1 Arduino と math.h の標準関数

待ち時間

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．

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 getDiscreteAngle(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 setDiscreteAngle(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 $\boldsymbol{q} = (0.0, 0.0)(\mathrm{rad})$ 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．実数データと同様にラベルを替えたい場合，
`void SerialPrintDataInt2(int Z[], char ch[]);`
を代わりに用いればよい．

制御の終了

`void ClosePacketAndPort(PortHandler *port, PacketHandler *packet);`
通信に使うポートとパケットをクローズし，制御モードを終了させる．