RTミドルウェアによるロボットプログラミング技術

４．ロボットの運動学と制御の基礎（解答）

演習

ロボット制御に必要な以下のプログラムを示せ

課題１：2自由度のアームの逆運動学を計算する以下の仕様の関数のC++プログラムを作成し実行結果を示せ

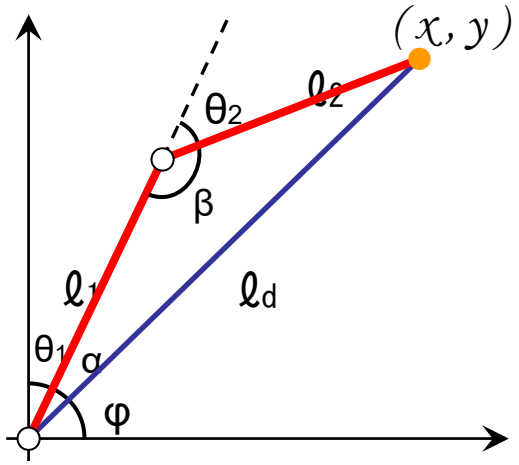
関数: th = invkinem(link, pos)
　　th: 2つの関節の角度[deg]
　　link: 2つのリンク長[m]
　　pos：手先位置[m]
　　引数、戻り値はいずれも要素数2の配列とする

課題２：ジョイスティックの現在値から移動ロボットの車輪角速度を出力するC++プログラムを作成し実行結果を示せ

関数: vl, vr = joy2vel (k, x, y)
　　k: 適当な係数
　　x, y: ジョイスティックの現在値
　　vl, vr：移動ロボットの左右車輪角速度 [rad/s]

課題1：2自由度のアームの逆運動学を計算する以下の仕様の関数のC++プログラムを作成し実行結果を示せ

関数: Angle = invkinem(link, pos)
　　th: 2つの関節の角度[deg]
　　link: 2つのリンク長[m]
　　pos：手先位置[m]
　　引数、戻り値はいずれも要素数2の配列とする



- ２自由度アームの制御に必要な計算式を考えてみよう

  最終的には…

  θ_1 ＝

  θ_2 ＝

  の形にしたい。

解答：逆運動学

～座標から関節の角度を求める～

- 余弦定理と逆関数を使って値を求めます。

$$\theta_1 = -\left(\frac{\pi}{2} - \alpha - \varphi\right)$$

$$\theta_2 = -(\pi - \beta)$$

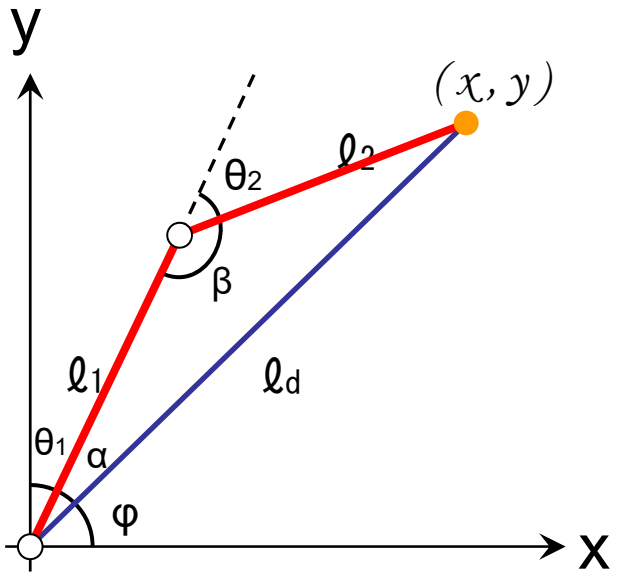
$$\cos\alpha = \left(\frac{l_1^2 + l_d^2 - l_2^2}{2 l_1 l_d}\right)$$

$$\cos\beta = \left(\frac{l_1^2 + l_2^2 - l_d^2}{2 l_1 l_2}\right)$$

$$\tan\varphi = \frac{y}{x}$$

$$\varphi = \tan^{-1}\frac{y}{x}$$

$$l_d = \sqrt{x^2 + y^2}$$



解答：逆運動学

～座標から関節の角度を求める～

- 二通りの解があります。

$$\theta_1 = -\left(\frac{\pi}{2} + \alpha - \varphi\right)$$

$$\theta_2 = -(-\pi + \beta)$$

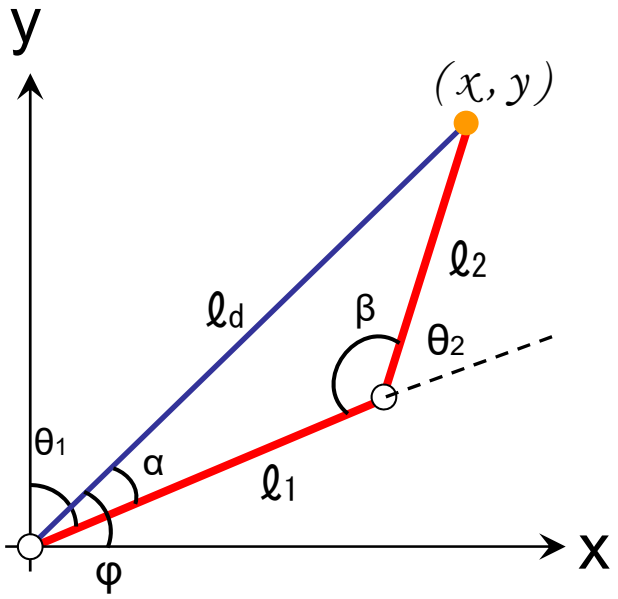
$$\cos\alpha = \left(\frac{l_1^2 + l_d^2 - l_2^2}{2 l_1 l_d}\right)$$

$$\cos\beta = \left(\frac{l_1^2 + l_2^2 - l_d^2}{2 l_1 l_2}\right)$$

$$\tan\varphi = \frac{y}{x}$$

$$\varphi = \tan^{-1}\frac{y}{x}$$

$$l_d = \sqrt{x^2 + y^2}$$



C++での実装

- #include <math.h>が必要
- πは M_PI (マクロ定義)
  - VC では#define _USE_MATH_DEFINES が必要
- $\sqrt{\ }$ は sqrt() 関数を利用
- 二乗は pow() 関数を利用
- sin, cos は sin(), cos() 関数を利用
- $\tan^{-1}$は atan2() 関数を利用
- $\cos^{-1}$は acos() 関数を利用

プログラム（C++）（1）

```
JointAngle invkinem(LinkLength link, Position pos)
{
  double ld(sqrt((pos.x * pos.x) + (pos.y * pos.y)));
  double l1(link.l0);
  double l2(link.l1);

  double a  = acos((pow(l1, 2) + pow(ld, 2) - pow(l2, 2)) / (2 * l1 * ld));
  double b  = acos((pow(l1, 2) + pow(l2, 2) - pow(ld, 2)) / (2 * l1 * l2));
  double phi = atan2(pos.y, pos.x);

  JointAngle j;
  j.th0 = - ((M_PI / 2) - a - phi) * 180 / M_PI;
  j.th1 = - ( M_PI - b) * 180 / M_PI;

  return j;
}
```

プログラム（C++）(2)

```
JointAngle invkinem2(LinkLength link, Position pos)
{
  double ld(sqrt((pos.x * pos.x) + (pos.y * pos.y)));
  double l1(link.l0);
  double l2(link.l1);

  double a  = acos((pow(l1, 2) + pow(ld, 2) - pow(l2, 2)) / (2 * l1 * ld));
  double b  = acos((pow(l1, 2) + pow(l2, 2) - pow(ld, 2)) / (2 * l1 * l2));
  double phi = atan2(pos.y, pos.x);

  JointAngle j;
  j.th0 = - ((M_PI / 2) + a - phi) * 180 / M_PI;
  j.th1 = - (-M_PI + b) * 180 / M_PI;

  return j;
}
```

参考：順運動学（2自由度）

点P1の位置は、

$$x_1 = l_1 \cos(-\theta_1)$$

$$y_1 = l_1 \sin(-\theta_2)$$

点P1から点P2の位置は、

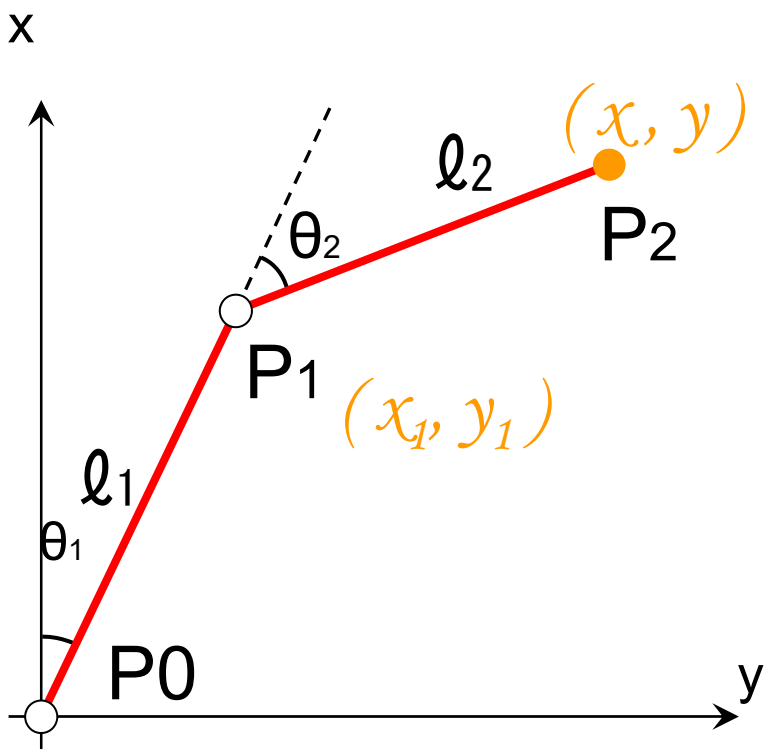
$$x_2 = l_2 \cos(-\theta_1 - \theta_2)$$

$$y_2 = l_2 \sin(-\theta_1 - \theta_2)$$

もとめる手先の位置は、

$$x = x_1 + x_2$$

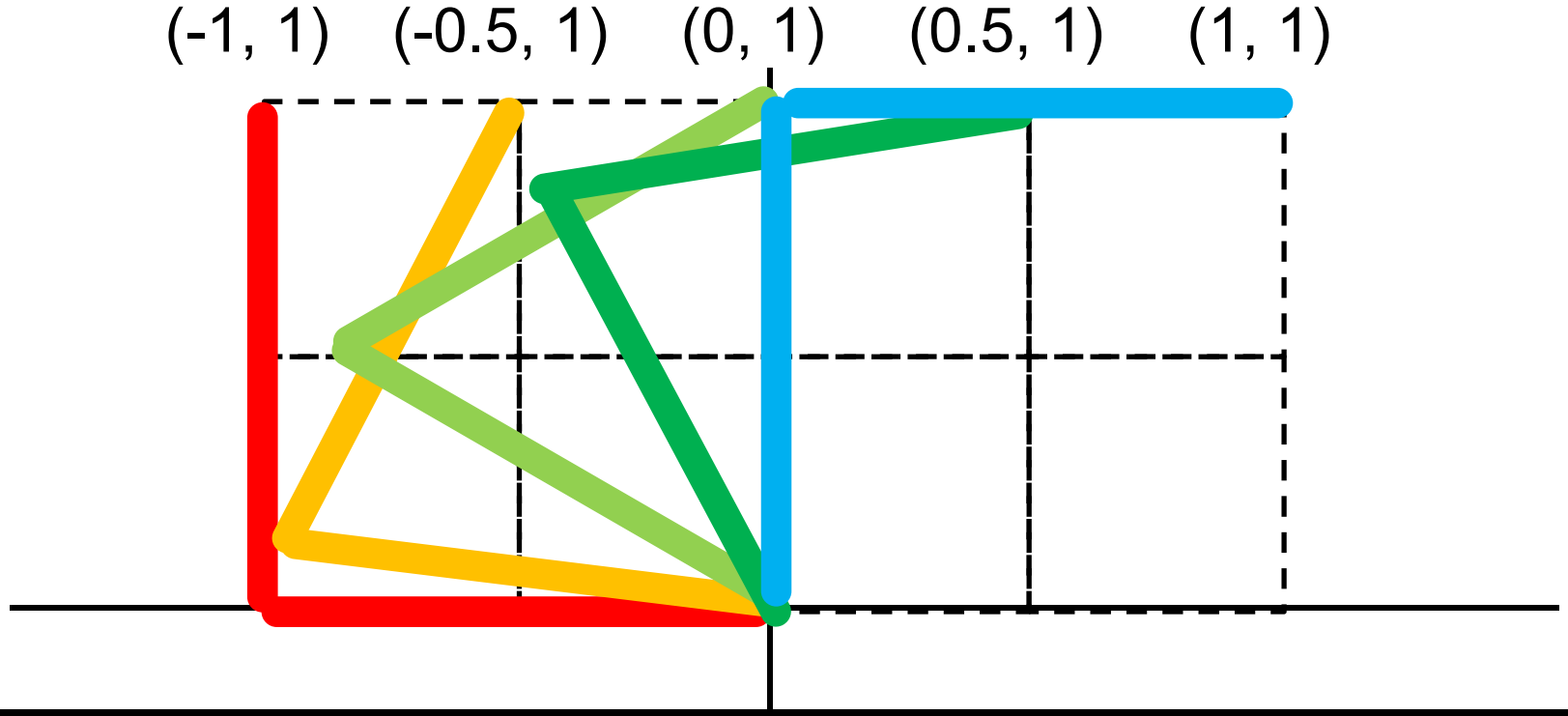
$$y = y_1 + y_2$$



解答

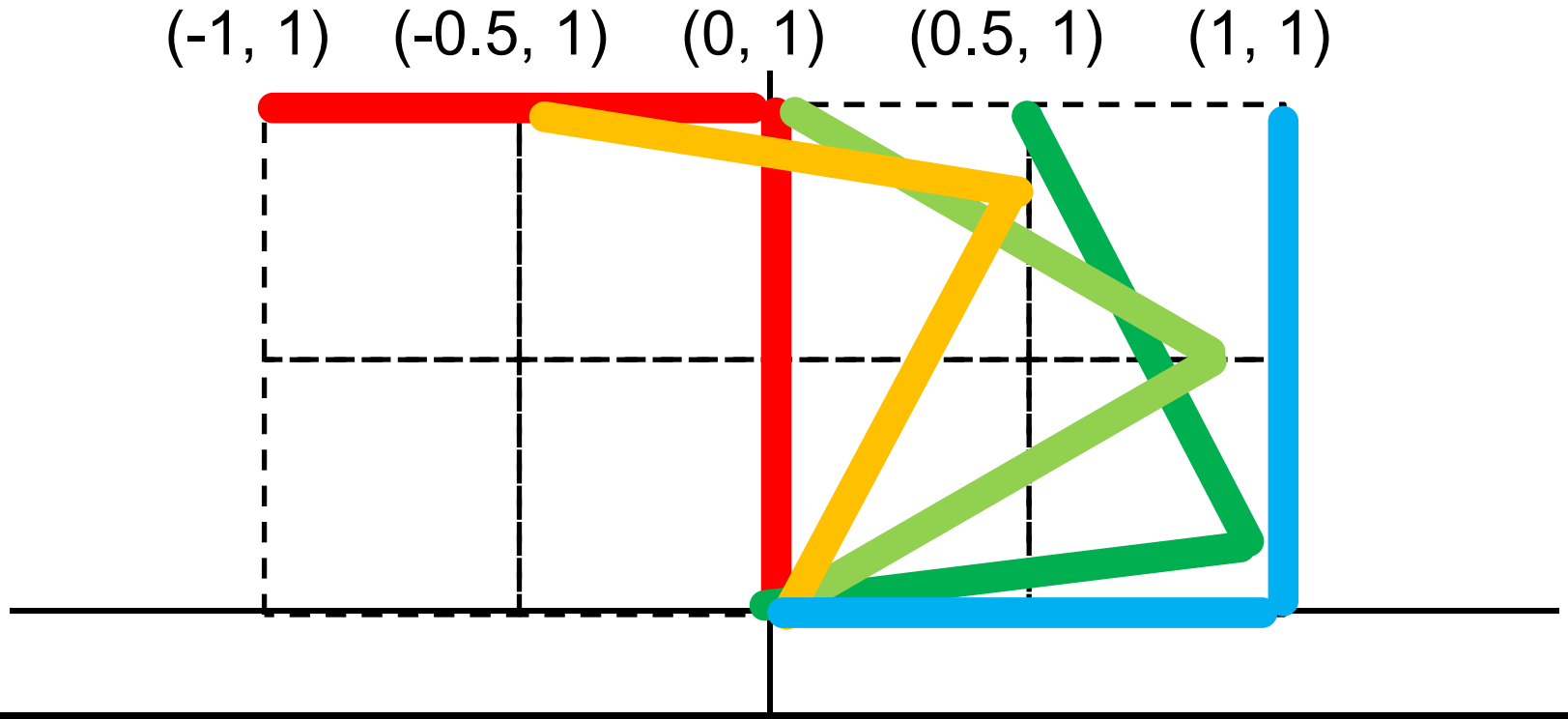
```
$ ./arm2dof
pos (x, y): -1, 1       ==> angle (th0, th1): 90, -90
pos (x, y): -0.5, 1     ==> angle (th0, th1): 82.5772, -112.024
pos (x, y): 0, 1        ==> angle (th0, th1): 60, -120
pos (x, y): 0.5, 1      ==> angle (th0, th1): 29.4471, -112.024
pos (x, y): 1, 1        ==> angle (th0, th1): -1.27222e-14, -90
または
$ ./arm2dof
pos (x, y): -1, 1       ==> angle (th0, th1): -0, 90
pos (x, y): -0.5, 1     ==> angle (th0, th1): -29.4471, 112.024
pos (x, y): 0, 1        ==> angle (th0, th1): -60, 120
pos (x, y): 0.5, 1      ==> angle (th0, th1): -82.5772, 112.024
pos (x, y): 1, 1        ==> angle (th0, th1): -90, 90
```

アームの姿勢




```
pos (x, y): -1, 1       ==> angle (th0, th1): 90, -90
pos (x, y): -0.5, 1     ==> angle (th0, th1): 82.5772, -112.024
pos (x, y): 0, 1        ==> angle (th0, th1): 60, -120
pos (x, y): 0.5, 1      ==> angle (th0, th1): 29.4471, -112.024
pos (x, y): 1, 1        ==> angle (th0, th1): -1.27222e-14, -90
```

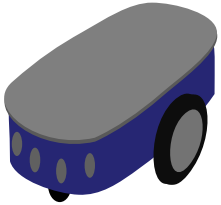
アームの姿勢



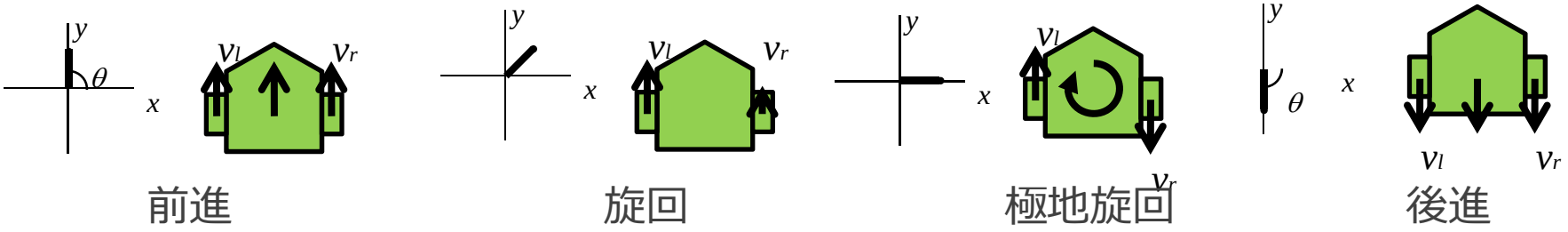

```
pos (x, y): -1, 1       ==> angle (th0, th1): -0, 90
pos (x, y): -0.5, 1      ==> angle (th0, th1): -29.4471, 112.024
pos (x, y): 0, 1        ==> angle (th0, th1): -60, 120
pos (x, y): 0.5, 1       ==> angle (th0, th1): -82.5772, 112.024
pos (x, y): 1, 1        ==> angle (th0, th1): -90, 90
```

課題２：ジョイスティックの現在値から移動ロボットの車輪角速度を出力するPythonプログラムを作成し実行結果を示せ

関数: vl, vr = joy2vel (k, x, y)
k: 適当な係数
x, y: ジョイスティックの現在値
vl, vr：移動ロボットの左右車輪角速度 [rad/s]

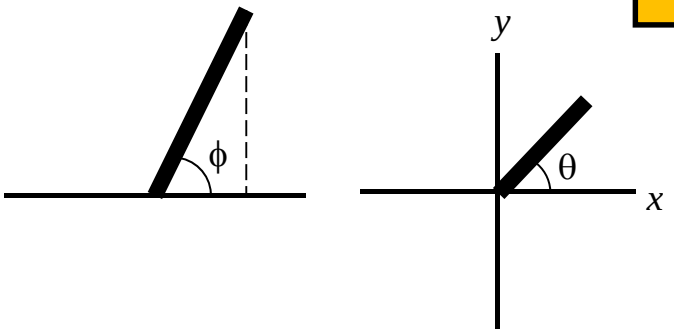


ジョイスティックを倒す角度と車輪の速度の大まかな関係は…

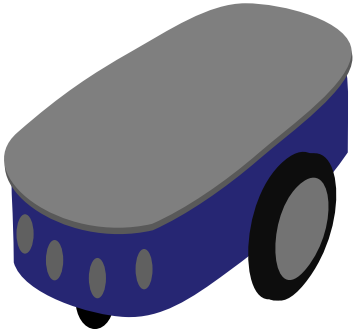


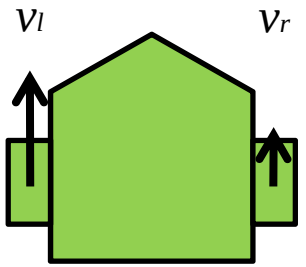
これらを滑らかにつなぐ関数を定義したい。

ジョイスティックによる操作



指令値
左右の車輪の（角）速度





横から見る
↓
どれだけ倒したか

上から見る
↓
どの方向へ倒したか

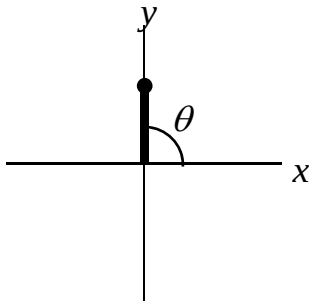
このタイプ（差動駆動型2輪移動ロボット）は
左右の車輪の速度を与えることで
ロボットを任意の場所へ移動させる

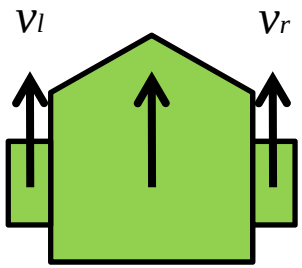
x, yの値が出力される

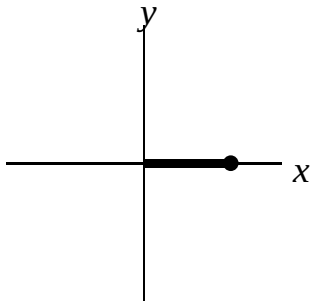
≠

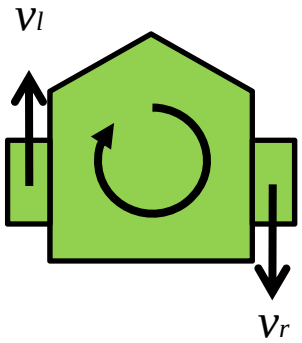
左右の車輪の速度を入力する

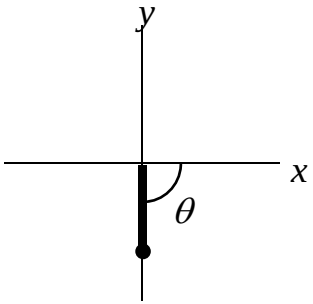
移動ロボットの制御

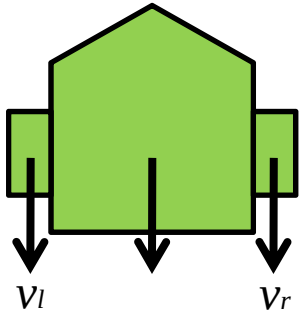


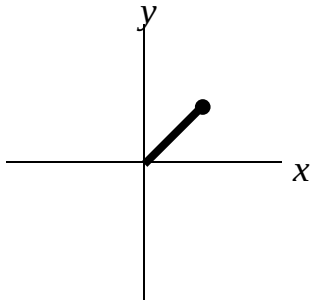


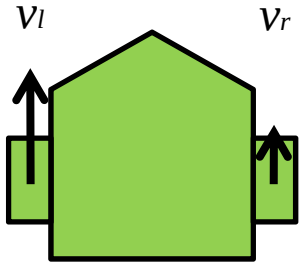




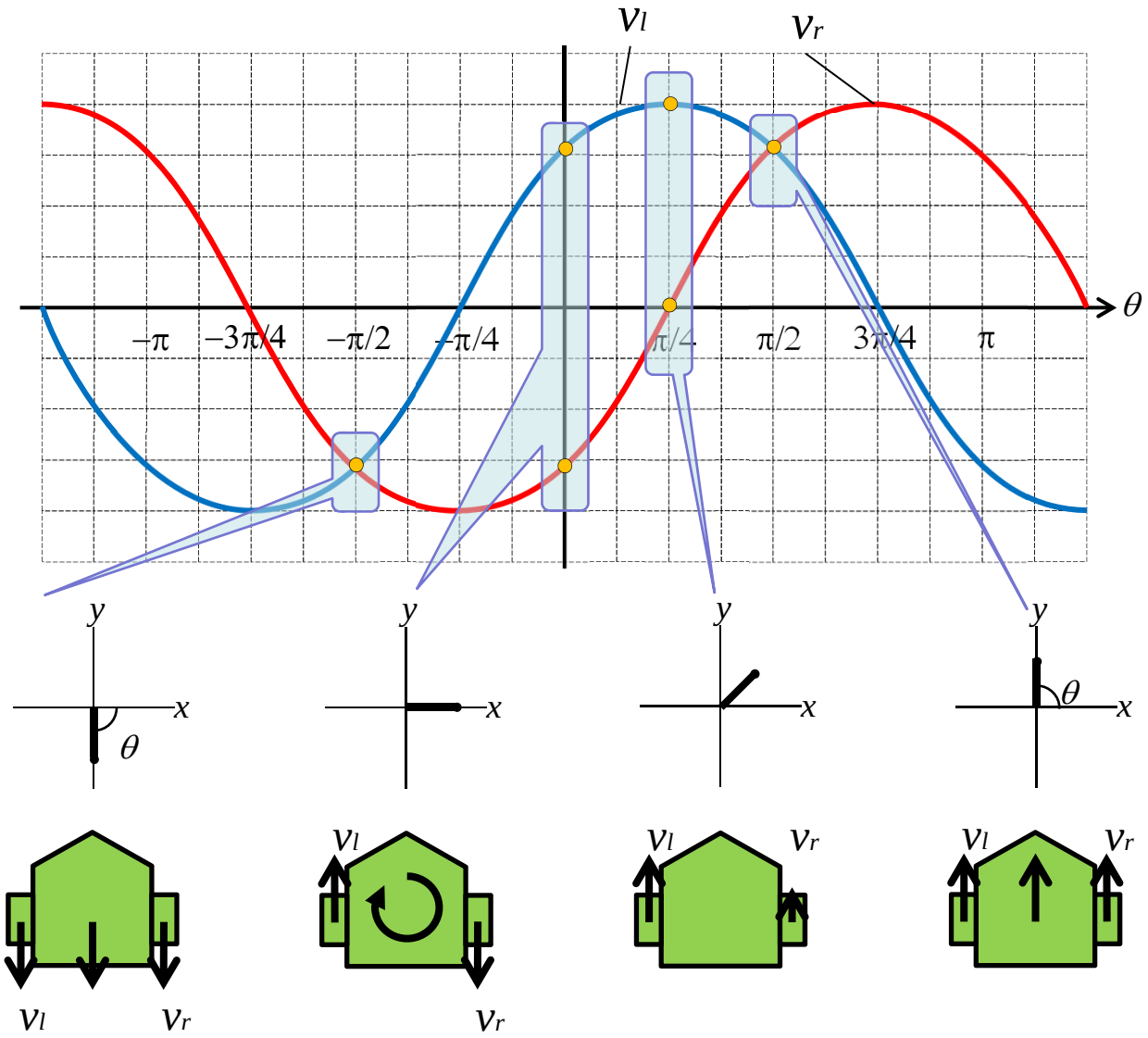




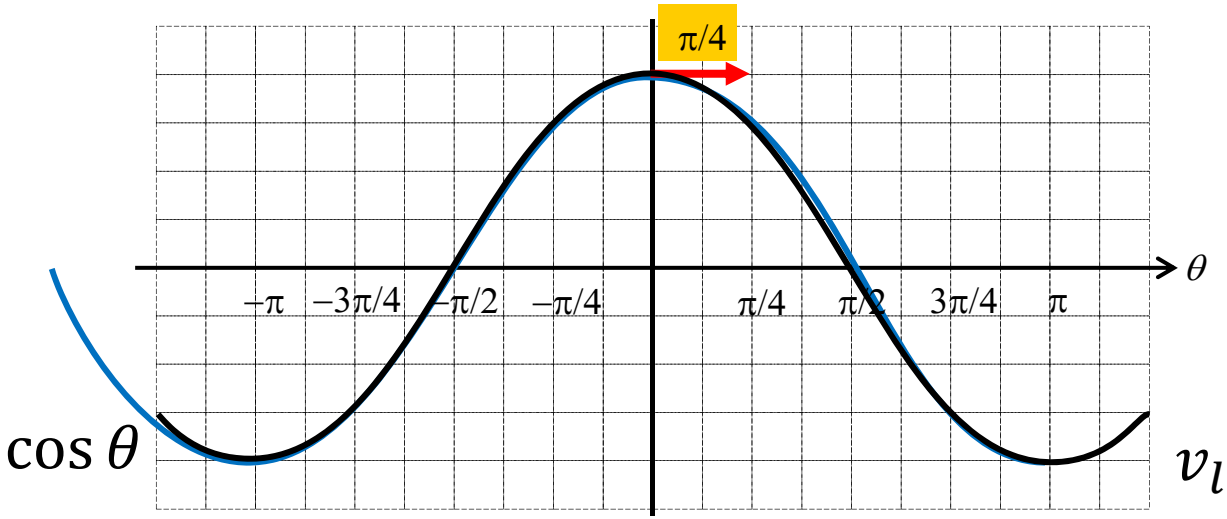




ジョイスティックと車輪の速度

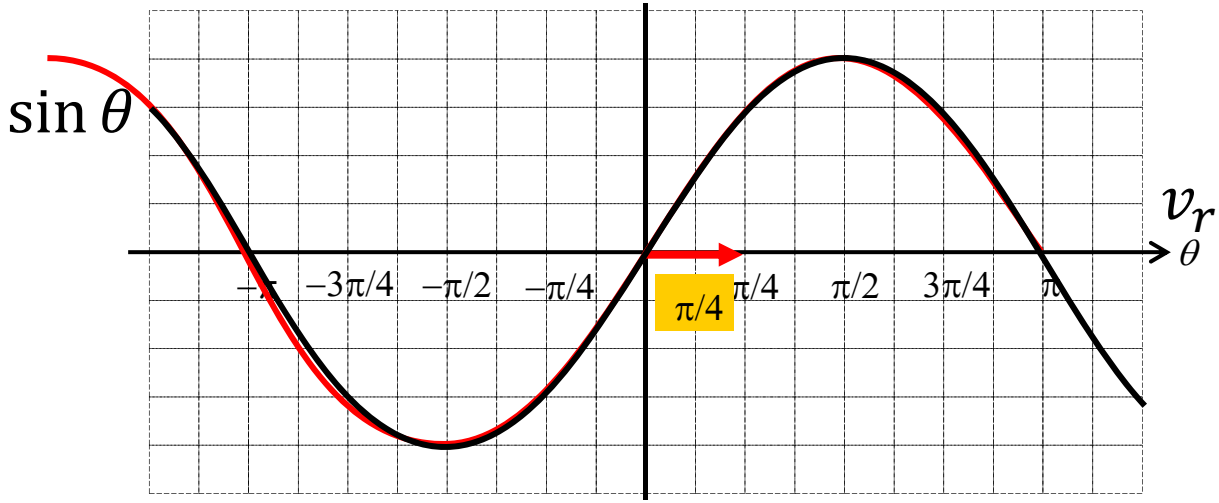


解答



$$v_l = \cos\left(\theta - \frac{\pi}{4}\right)$$

左車輪の速度



$$v_r = \sin\left(\theta - \frac{\pi}{4}\right)$$

右車輪の速度

プログラム（C++）

```
WheelVelocity joystick(double k, Position pos)
{
  double th = atan2(pos.y, pos.x);
  double v = k * hypot(pos.x, pos.y);
  WheelVelocity wv;
  wv.v_l = v * cos(th - M_PI / 4.0);
  wv.v_r = v * sin(th - M_PI / 4.0);

  return wv;
}
```

解答

```
$ ./joystick
pos (x, y): 0, 1        ==> velocity (v_l, v_r): 0.707107, 0.707107
pos (x, y): 1, 1        ==> velocity (v_l, v_r): 1.41421, 0
pos (x, y): 1, 0        ==> velocity (v_l, v_r): 0.707107, -0.707107
pos (x, y): 0, -1       ==> velocity (v_l, v_r): -0.707107, -0.707107
pos (x, y): -1, -1      ==> velocity (v_l, v_r): -1.41421, -1.73191e-16
```