1 ソフトウェア

1.1 アルゴリズム

まず，次のように 8 つの二次元単位ベクトルを定義する．

$$\boldsymbol{u}_0 = \begin{bmatrix} -0.707 \\ -0.707 \end{bmatrix}, \quad \boldsymbol{u}_1 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \quad \boldsymbol{u}_2 = \begin{bmatrix} 0.707 \\ -0.707 \end{bmatrix}, \quad \boldsymbol{u}_3 = \begin{bmatrix} 1 \\ 0 \end{bmatrix},$$
$$\boldsymbol{u}_4 = \begin{bmatrix} 0.707 \\ 0.707 \end{bmatrix}, \quad \boldsymbol{u}_5 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \boldsymbol{u}_6 = \begin{bmatrix} -0.707 \\ 0707 \end{bmatrix}, \quad \boldsymbol{u}_7 = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

これらは，機体周囲に放射状に備え付けた距離センサにより得られる物体との距離の値に応じて，斥力に相当するベクトルを算出し，機体に対して物体との距離を適切に保つよう進行方向を表すベクトルに合成する際に，方向として利用する．

距離センサのセンサ値 v より距離 d [cm] を求める式は次の 2 式である．

$$d_{\text{long}} = 45.514 v^{-0.822} \tag{1.1}$$
$$d_{\text{short}} = 0.09999 v + 0.4477 \tag{1.2}$$

障害物との衝突を避けるため，センサにより得られた距離が小さいほど大きな斥力ベクトルが機体の進行方向ベクトルに作用するようにしなければならない．また，同時に離れすぎることも，競技エリア外に出てしまうことが懸念されるため，避けなければならない．そのため，距離が大きい時には大きな負の斥力ベクトル，すなわち障害物から引力を受けるように，次の式を利用する．

$$\tanh^{-1}(x-1) \tag{1.3}$$

1.2 画像認識

炎上ポールの認識には配布された Rapspberry Pi NoIR Camera V2（以下，カメラモジュール）を使用する．画像撮影から最も近い（と思われる）ポールへのベクトルを出力する一連の手順を，簡単に以下に示す．

画像取得

カメラモジュールへのアクセスには既成ライブラリ [1] を利用した．撮影によりピクセル値の二次元配列（cv::Mat）が出力として得られる．画像サイズの大小が及ぼす処理時間への影響は極めて大きいが，それが画像処理結果の精度に与える影響は未検証であるため，現在は適当

な値が設定されている．

カラーモデル変換

得られた画像のカラーモデルを RGB から HSV に変更する．

2 値画像化

HSV 画像データに対し赤色マスクをかけて二値画像に変換する．赤色マスクの範囲は適当．

ノイズ除去

適当に膨張・縮小を繰り返してノイズを除去する．ちゃんとやってる処理じゃねえから現状ほとんど意味がねえ．

構造解析

2 値画像中の輪郭線を検出した後，それを矩形で囲む．囲んだ矩形を縦横比で解析し，ポールの縦横比に対して ±20% 以上の差があるものを除外する．残った矩形の重心点を求めて，適当にベクトルを作る．

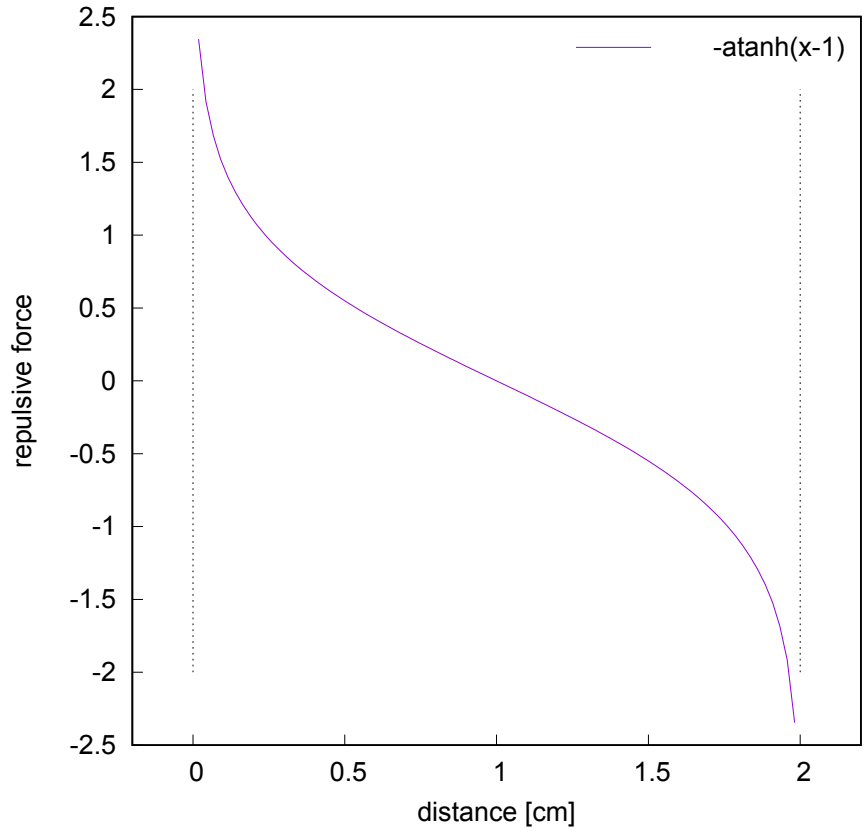

図 1.1　ほげほげ

参考文献

[1] "RaspiCam: C++ API for using Raspberry camera with/without OpenCV",
"https://www.uco.es/investiga/grupos/ava/node/40", 2017 年 5 月 31 日最終確認.