2023 年度 卒業論文
自律走行型ロボットと人間のコミュニケーション発展に向けて

奈良女子大学 生活環境学部 情報衣環境学科
生活情報通信科学コース 4 回生 新出研究室
20480293 久田美海

2024 年 2 月

目次

図 1　人数カウント用コントローラ RSX-A100

1　はじめに

近年、様々な場面でロボットが人間に代わって行動をする機会が増えている。そのため、ロボットと人間のコミュニケーションを豊かにすることで、これからの時代、ロボットと人間がより上手く社会で共存する必要があると考えられる。現在、新出研究室で扱っている自律走行型ロボット Arno は、センサによって周囲の地図や障害物を認識することで、障害物の前では止まったり、進む角度を変えたりしながら、自律走行を行う。この際に、ロボットが通行人の動きをリアルタイムで認識することができれば、通行人とのコミュニケーションによって互いの進路支障を解決したり、それ以外にもロボットの応用を拡げることもできると考える。そこで本研究では、Arno に新たに人数計測用のセンサ RSX-A100 を追加することで、周囲の人間の動きを察知し、その動きに応じて Arno が何らかの反応を行う機構の実現を目指す。

2　センサと位置データの取得

2.1　人数計測用センサ RSX-A100

RSX-A100 は、北陽電機が開発した製品で、測域センサ用コントローラ「RS コントローラ」を利用したサーバー式人数カウントシステムである。最大 20m、270 ° の広範囲を検出し、レーザ式の精度の高い入場・退場者数のカウントが可能である。また無線 LAN が内蔵されており、PC やスマートフォンを利用してブラウザ上から来場者の確認も行える (図 1, 図 2) [1]。

本研究では、このセンサに入ってくる情報を利用して、周囲の人間の動きを認識するプログラムを作成する。

2.2　Arno との通信

Arno は ROS で動いているため、センサを Arno で利用するには、センサから送られてくる情報を ROS のメッセージに変換して ROS トピックに配信する必要がある。そこで本研究ではまず、RSX-A100 が検出した周囲の人物（オブジェクト）の位置を検出してそのデータを取得し、それを ROS トピックに配信するプログ

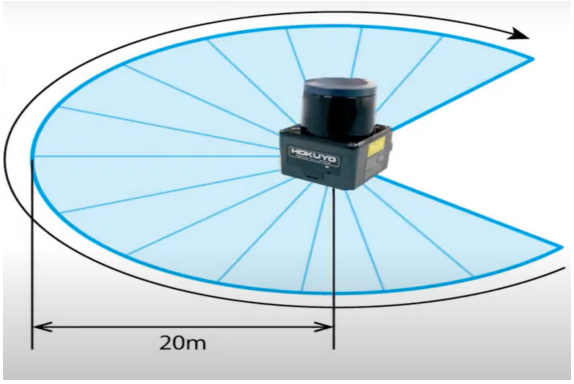

図 2　RSX-A100 の検出範囲

(CLIENT→SERVER)

'M'	'A'	'G'	'P'	':'	CRC	RT

(SERVER→CLIENT)

'M'	'A'	'G'	'P'	':'	status	LF
Number Of	'~'	Timestamp	LF			
Object ID	'~'	X-Coodinate	'~'	Y-Coodinate	'~'	Object Size	LF
CRC	LF					
LF						

図 3　検出位置データ取得マニュアル

ラムを作成した。

2.3　検出位置データ取得プログラム

検出位置データ取得プログラムは、RSX-A100 にとってはクライアントとなってデータを受け取るプログラムである。まず RSX-A100 と TCP で通信し、検出範囲にいる各オブジェクトに割り当てられた ID とそれぞれのオブジェクトの X 座標と Y 座標をセンサからバイト列として受け取る (図 3)。この時の X 座標と Y 座標は、センサの位置を原点として検出しているものである。そして、その各 ID と X 座標と Y 座標をオブジェクトごとにまとめて ROS トピックに配信する関数を作成し、配信されたメッセージから各オブジェクトの位置データを読み取れるようにした (図 4) [2]。

実行結果は図 5 のとおりである。この場合、センサの周りにオブジェクトが 3 つあり、一番上がオブジェクト ID5、2 番目がオブジェクト ID3、3 番目が 0 というようにそれぞれに ID が割り当てられているとわかる。そして検出位置も X-Coordinate、Y-Coordinate という形で座標データが表示されていることが読み取れる。

```
# マニュアル4.3「検出位置データ取得」を行う
# (センサ番号、タイムスタンプ、1スキャン分の距離データのリスト)の3要素か
# なるタプルをセンサの台数分持つリストを返す
def getPosition(self, req):
    if not 0 <= req.sensorNo <= 4:
        __class__.__error('sensorNo', sensorNo, 'getPosition')
    if req.raw not in (True, False):
        __class__.__error('raw', raw, 'getPosition')

    reply = self.getreply(b'MAGP:')

    status = reply[0][5:7]
    if status != b'00':
        raise StatusError('Status code is %s' % status)
    del reply[0:2] # 先頭のstatus部分とセンサ台数の部分を除去
    # この段階でreplyにはコントローラから返された
    # センサの台数分のバイト列からなる情報が入っている
    print(reply)
    retlist = list(map(self._data2Position, reply))
    return Positions(positionList = retlist)
def _data2Position(self, data):
    l = data.split(b'~')
    o = int(l[0])
    X = int(l[1])
    Y = int(l[2])
    return Position(objectID = o, X_Coordinate = X, Y_Coordinate = Y)
```

図４　検出位置データ取得プログラム

```
positionList:
  -
    sensorNo: 0
    stamp: 0
    objectID: 5
    X_Coordinate: 1603
    Y_Coordinate: 2189
    ObiectSize: []
  -
    sensorNo: 0
    stamp: 0
    objectID: 3
    X_Coordinate: -1336
    Y_Coordinate: 2690
    ObiectSize: []
  -
    sensorNo: 0
    stamp: 0
    objectID: 0
    X_Coordinate: 1181
    Y_Coordinate: 1411
    ObiectSize: []
---
positionList:
  -
    sensorNo: 0
    stamp: 0
    objectID: 5
    X_Coordinate: 1585
    Y_Coordinate: 2170
    ObiectSize: []
  -
    sensorNo: 0
    stamp: 0
    objectID: 3
    X_Coordinate: -1333
    Y_Coordinate: 2690
    ObiectSize: []
  -
    sensorNo: 0
    stamp: 0
    objectID: 0
    X_Coordinate: 1181
    Y_Coordinate: 1410
    ObiectSize: []
---
```

図５　実行結果

```
#!/usr/bin/python3
# -*- coding: euc-jp -*-

import roslib
roslib.load_manifest('rsxA100ichi')

import rospy
from rsxA100ichi.msg import Position, Positions

import math
import array as a

objects = {}

def traceCallback(msg):
    rospy.loginfo('callback function called')
    for pos in msg.positionList:
        id = pos.objectID
        if id in objects:
            rospy.loginfo('existing object! id {0}'.format(id))
        else:
            rospy.loginfo('new object! id {0}'.format(id))
        objects[id] = pos

        X = pos.X_Coordinate
        if X in objects:
            rospy.loginfo('existing object! X {0}'.format(X))
        else:
            rospy.loginfo('new object! X {0}'.format(X))
        objects[X] = pos

        Y = pos.Y_Coordinate
        if Y in objects:
            rospy.loginfo('existing object! Y {0}'.format(Y))
        else:
            rospy.loginfo('new object! Y {0}'.format(Y))
        objects[Y] = pos

        dis = math.sqrt(X**2 + Y**2)
        if dis in objects:
            rospy.loginfo('existing distance! dis {0}'.format(dis))
        else:
            rospy.loginfo('new distance! dis {0}'.format(dis))
        objects[dis] = pos

rospy.init_node('example1_f')
rospy.Subscriber('rsxA100_Position', Positions, traceCallback)
rospy.spin()
```

図 6　プログラム

3　オブジェクトの動きの把握

前節で述べたプログラムにより、位置データを取得し、トピックに配信することができた。しかし、このままではその時々でのオブジェクトの位置がわかるだけであり、それぞれのオブジェクトがどのように動いているかは不明である。そこで次に、この検出位置データを利用して、各オブジェクトの位置の変化、つまりどのように動いているかがわかるプログラムを作成した。

作成したプログラムと実行結果を図 6 と図 7 に示す。トピックからメッセージを受け取った時の Callback 関数を使用して、各オブジェクトの ID、X 座標、Y 座標が変化していれば “new object” 変化していなければ “existing object” と出力されるようにし、各オブジェクトの動きがわかるようにした。そして X 座標と Y 座標から、三平方の定理を使ってオブジェクトとセンサとの距離を計算し、出力されるようにプログラムを作成した。

この状況を図を使って表すと図 8、図 9 のようになる。同じ ID のオブジェクトに注目して、座標や距離が変化していれば動いている、変化していなければ静止しているということである。この図の場合、座標と距離から赤のオブジェクトは静止しており、黄色のオブジェクトは動いていると分かる。

また黄色のオブジェクトは、オブジェクトとセンサの距離が小さくなっており、オブジェクトが Arno に近づいてくることがわかる。この近づいてきているという動きがあった時に Arno が周囲に対して何か反応できれば、周囲とコミュニケーションを取ることにつながる。

```
5789177.707475]: callback function called
5789177.711565]: existing object! id 5
5789177.713092]: new object! X 1585
5789177.714819]: new object! Y 2170
5789177.716789]: new distance! dis 2687.2151011781693
5789177.718519]: existing object! id 3
5789177.720367]: existing object! X -1333
5789177.722050]: existing object! Y 2690
5789177.723868]: new distance! dis 3002.164052812571
5789177.725713]: existing object! id 0
5789177.727545]: existing object! X 1181
5789177.729445]: existing object! Y 1410
5789177.731341]: existing distance! dis 1839.2555559247335
```

図 7　実行結果

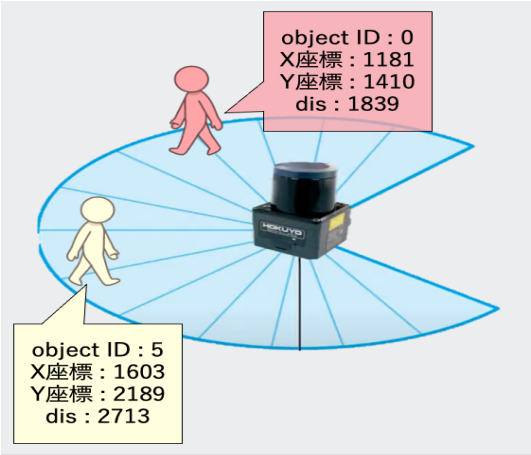

図 8　イメージ図

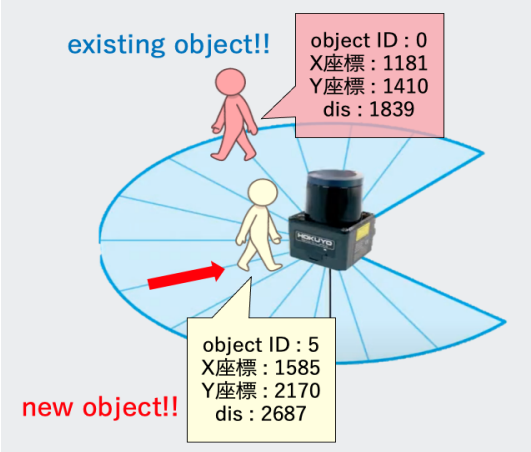

図 9　イメージ図

```
def traceCallback(msg):
    rospy.loginfo('callback function called')
    for pos in msg.positionList:
        id = pos.objectID
        if id in objects:
            rospy.loginfo('existing object! id {0}'.format(id))
        else:
            rospy.loginfo('new object! id {0}'.format(id))
        objects[id] = pos
        continue

        X = pos.X_Coordinate
        if X in objects:
            rospy.loginfo('existing object! X {0}'.format(X))
        else:
            rospy.loginfo('new object! X {0}'.format(X))
        objects[X] = pos

        Y = pos.Y_Coordinate
        if Y in objects:
            rospy.loginfo('existing object! Y {0}'.format(Y))
        else:
            rospy.loginfo('new object! Y {0}'.format(Y))
        objects[Y] = pos

        if len(objects[id]) < 5:
            objects[id].append(pos)
        else:
            del(objects[id][0])
            objects[id].append(pos)

        if len(objects[id]) < 5:
            continue

        if objects[id][0].X_Coordinate**2 + objects[id][0].Y_Coordinate**2 < obj
ects[id][1].X_Coordinate**2 + objects[id][1].Y_Coordinate**2 < objects[id][2].X_
Coordinate**2 + objects[id][2].Y_Coordinate**2 < objects[id][3].X_Coordinate**2 
+ objects[id][3].Y_Coordinate**2 < objects[id][4].X_Coordinate**2 + objects[id][
4].Y_Coordinate**2:
            print("!!!coming object!!!")
            subprocess.call("aplay z.wav",shell=True)
```

図 10　プログラム

4　周囲に対して反応させる

オブジェクトの接近に反応する一例として、本研究では、周囲のオブジェクトが近づいてきたら、Arno が何か音を発生させるというプログラムを作成した。プログラムと実行結果はそれぞれ図 10 と図 11 のとおりである。「オブジェクトが近づいてくる」という部分に関しては、同じオブジェクトのセンサとの距離が 5 回連続で小さくなっていれば、オブジェクトが近づいてくるという認識になるようにプログラムを作成した。センサに入ってくるデータをオブジェクト ID ごとのリストに入れ、5 つ溜まれば 6 つ目からは 1 番古いデータを撤廃し新しいデータを後ろに加える、という方法で最新の 5 つのデータを読むことができるようにし、この 5 つのデータの比較で接近を判定した。実行結果から、オブジェクトが近づいてきたら音声ファイルが再生されていることがわかる。実際に著者がセンサに近づいても音声ファイルを再生することを確認した。

5　おわりに

人数カウント用センサからのデータを ROS を介してロボットが受け取ることで、自律走行型ロボットが周囲の動きに対して反応する機構を実現することができた。今後、さらにセンサのもつ機能を活用することで、周囲の動きに対して反応するだけでなく、ロボットから人間に対して道を開ける交渉をしたり、会話ができるようになれば、自律走行型ロボットと周囲の人間のコミュニケーションをさらに豊かにすることが可能であると考える。

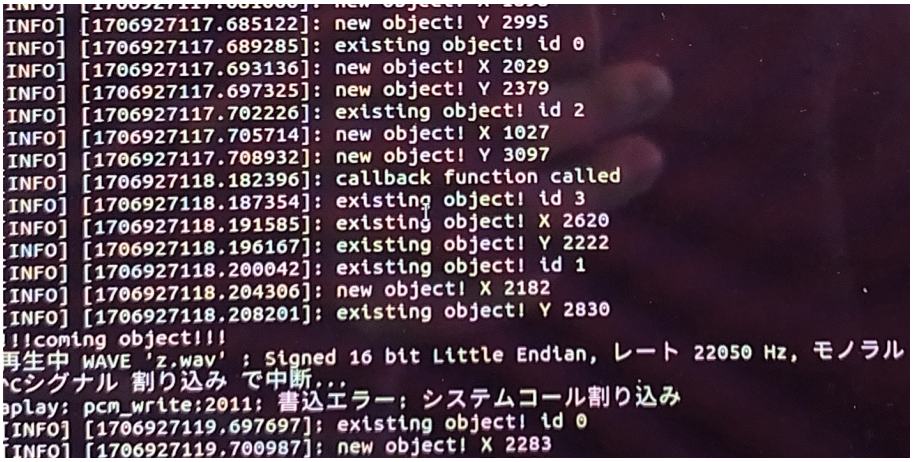

図 11　実行結果

6　謝辞

本研究及び本論文の執筆にあたり、最後まで丁寧かつ細やかな指導をしてくださった新出尚之准教授、論文にご助言くださった鴨浩靖准教授に深く感謝の意を表します。また、ロボット・センサの提供、中之島ロボットチャレンジにご協力頂いた北陽電機の皆様、並びにロボットの使い方やプログラムの書き方についてご指導してくださった新出研究室の先輩方に心から感謝申し上げます。

参考文献

[1] 北陽電機. Rsx-a100 人数カウント用 rs コントローラ—応用システム. https://www.hokuyo-aut.co.jp/search/single.php?serial=178.

[2] 北陽電機. 人数カウント用 rs コントローラ [rsx-a100] 通信仕様書, 2018.