卒業論文 2019 年度 (令和元年度)

QR コードを用いた経路情報に関してステートレスなロボットナビゲーションシステムの設計と実装

慶應義塾大学 環境情報学部環境情報学科
井手田悠希

卒業論文要旨 - 2019 年度 (令和元年度)

QR コードを用いた経路情報に関してステートレスなロボットナビゲーションシステムの設計と実装

人間が物品管理を行っている倉庫で物品管理の自動化を行おうとする場合二通りの方法が考えられる．一つ目が物品棚自体に機械を搭載したり，物品棚自体をロボットで移動させながら物品管理を行う方法．二つ目は倉庫内は何も手を加えずに巡回ロボットを用いて物品管理を行う方法である．本研究では 2 つ目の方法で物品管理に利用されることを想定したロボットのナビゲーションシステムの開発を目指す．ロボットには移動の自由度が高いドローン (小型無人航空機) を採用している．ロボットを自律的に倉庫内を移動させる場合ロボットの経路情報の管理は非常に重要であり，現状ロボットのナビゲーションシステムでは経路情報はソフトウェアによる生成及び管理が主である．しかしながら，倉庫での巡回という目的においては現状のナビゲーションシステムは経路情報の扱いを含めて少々冗長な部分が存在している．そこで今回は特に経路情報の扱いに注目し，経路情報に関してステートレスなナビゲーションシステムを考案し利用可能かを検証した．本研究では QR コードを用いて経路情報に関してステートレスなナビゲーションシステムの実現をする．

キーワード:
1. ドローン, 2. ナビゲーションシステム, 3. 倉庫, 4. QR コード,

慶應義塾大学 環境情報学部環境情報学科
井手田悠希

Abstract of Bachelor's Thesis - Academic Year 2019 - 2020

Development of a robot navigation system with QR code

There are 2 ways which realize automate item management system in the warehouse. The first way is using an automated rack or using robots to move a rack. The second way is using patrol robots without any warehouse reform. This research focuses on the robot which is used in a second way. We use a drone as a robot because the drone has many degrees of freedom. It is important that is route information and managed by software mainly. However, It is verbose a little for patrol in the warehouse that already exists methods. Thus, this research targeted the route control system and developed a stateless route control system. In this system, we use QRCode to develop a stateless route control system.

Keywords :
1. Drone, 2. Navigation System, 3. WareHouse, 4. QRCode,

Keio University Bachelor of Arts in Environment and Information Studies
Yuki Ideta

目 次

図 目 次

表 目 次

第1章　序論

本章では本研究の背景，課題及び手法を提示し，本研究の概要を示す．

1.1　本研究の背景

1.1.1　自己位置推定手法の発展

モバイルデバイスの高度化やセンサの小型化，計算能力の向上により高度な処理が可能となった．これにより自己位置推定の様々な手法が提案され，その精度も向上を続けている．しかしながら実運用を考慮すると運用に必要な金銭的費用や機体性能，環境による制約が多く要件に応じた適切な自己位置推定手法を取る必要がある．

1.1.2　倉庫内での物品管理方法

倉庫内の物品管理[1]の方法は大別すると以下の二通りが挙げられる．一つ目が物品棚自体に機械を搭載したり，物品棚自体をロボットで移動させながら物品管理を行う方法．二つ目は倉庫内は何も手を加えずに巡回ロボットを用いて物品管理を行う方法である．自動型の物であれば物品棚自体に自動化用の機械を取り付ける場合や物品棚自体をロボットで移動させながら物品管理及び取得・移動を行う物が存在している．一方で人間が読み取り機を用いて物品管理や物品の配置/取得をする形式の倉庫も多く存在している．現在人間が物品管理を行っている倉庫で新たに形で自動化を行おうとすると倉庫自体の大きな改修が必要となる．ここで大規模な改修が不要な物品管理の自動化の方法として自律航行するロボットを巡回させる形での物品管理の自動化が考えられる．また，この際用いられるロボットとしては高さのある棚に対しても対応が容易な自由度の高いドローンを用いることが想定される．そこで本研究では将来的に物品管理に利用されることを想定したドローンのナビゲーションシステム[2]の開発を目指す．

1.2　本研究の問題と仮説

自律航行するロボットの制御において最も重要な要素のうちの一つが自己位置推定である．自己位置推定とは，任意の系においてロボット自身がどこにいるのかを推定する事で

[1]本論文では定期的に物品の位置の記録を行うことを指す
[2]本論文ではロボットの自己位置推定と経路選択を行うシステムを指す

ある．

現在自己位置推定には GPS(Global Positioning System) を用いる手法が非常に有力であり頻繁に用いられる．しかし，屋内等の理由により直上方向に遮蔽物がある場合にはその精度が著しく低下することや利用できないという状況が発生する．その為屋内では後述する Visual SLAM や LiDAR SLAM，TDOA 等様々な自己位置推定手法が考案され利用されている．これらの手法を利用したナビゲーションシステムで倉庫の巡回タスクを想定した実運用を考えると, いくつかの冗長な点と問題点が浮上する．

1.2.1 倉庫の巡回タスクにおける特徴

まず倉庫の巡回というタスクを考えた際に挙げられる特徴を以下にまとめる．

- 移動経路がほぼ固定化する
- 障害物の位置が既知である
- 機器の故障やメンテナンス等により機体の入れ替えが頻繁に発生する
- 収容率を上げる為にも通路は狭ければ狭い程良いとされる

1.2.2 倉庫の巡回タスクにおける課題

これらの特徴を踏まえた上で既存のナビゲーションシステムでは冗長であると考えられる部分や課題を以下にまとめる．詳細については 3 章にて説明する．

- 経路情報の管理 3.2.1
- 倉庫内での自己位置推定 3.2.2
- 専用機材の用意 3.2.3

1.2.3 本研究の仮説

そこで本研究では QR コードを利用する事で倉庫の巡回タスクの特徴を踏まえて既存の自己位置推定方法を利用せず，また経路情報のソフトウェア側での管理も行わず，比較的容易に用意できる機材を用いて巡回タスクを達成する事ができるのではないかと考えた．本研究では自己位置推定の為のランドマークと経路情報の管理をまとめて QR コードが担っている．

1.3 本論文の構成

本論文における以降の構成は次の通りである．

2 章では，本研究の要素技術となる PnP 問題と SLAM と QR コードに着いて整理する．

3 章では，本研究における問題の定義と，解決するための要件の整理を行う．

4 章では，本研究の提案手法を述べる．

5 章では， 4 章で述べたシステムの実装について述べる．

6 章では， 3 章で求められた課題に対しての評価を行い，考察する．

7 章では，本研究のまとめと今後の課題についてまとめる．

第2章　本研究の要素技術

本章では本研究の要素技術となる PnP 問題と GraphBasedSLAM と QR コードについて各々整理する．

2.1　PnP 問題

PnP(Perspective-n-Point) 問題は 3 次元空間上の n 個の点の情報が与えられた際にカメラ座標系に投影された 2 次元平面状の同じ点をを用いてカメラの外部パラメータ[1]を求める．という問題を指す．

本論文上の実装では QR コードの左上の角を世界座標系原点として推定を行う．QR コードの四角を利用し P4P 問題として帰着させる．

2.2　QR コード

自動車メーカーのデンソーに発明された．汚れによる耐障害性が非常に高く，非対称なので，向きの検出も可能．英数字で最大 4296 文字まで記録が可能

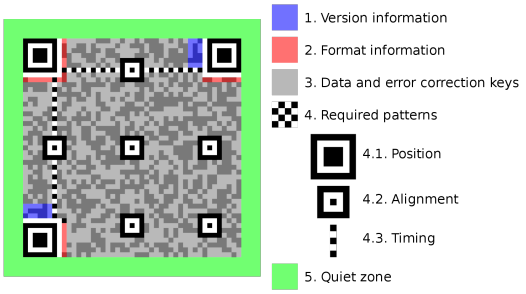

図 2.1: QR コードの仕様: https://en.wikipedia.org/wiki/QR_code

コンピュータ上で検出する際は以下の手順で検出を行う．

1. 水平方向に探索をし，白黒の比が 1:1:3:1:1 になる箇所を探索しファインパターンの候補として記録する．

[1]世界座標系におけるカメラ座標系の座標を求める為の回転と変換のパラメータ

2. 垂直方向に探索をし，ファイパターン候補の中心から上下にそれぞれ白黒の比が3:2:2になるか検証する．

3. 3箇所のファインパターンが求まれば3点で構成される平行四辺形の頂点から残りの1点を推定し四角を推定します．

データ部分のデコードは右下から順に黒を1白を0として右下のマスを基点として左隣のマス，上のマス，左上のマス，と順に読み取っていく．

第3章　本研究における問題定義と仮説

本章では 1 章で述べた背景より，現状の自律航行システムを倉庫内で物品管理に用いる場合の問題点を列挙し，整理する．また，どのように解決すれば良いのかを定義する，

3.1　本研究における問題定義

現状の自律飛行システムを倉庫内での物品管理に用いる場合の問題点を列挙し，整理する

3.2　倉庫の巡回タスクで利用する場合の課題

3.2.1　経路情報の管理

既存の方法による倉庫を巡回タスクでの経路情報の管理はいくつか方法が考えられる．

はじめに考えられるのは機体それぞれに飛行プログラムの一部として経路情報を書き込む方法である (以下経路埋め込み型と表記する)．外部との通信を必要とせず自律飛行が可能である事に加えて，倉庫の巡回タスクにおいては基本的に経路情報が変化しない為適していると考えられる．しかし飛行前に経路情報のチェックやアップデートの有無の確認，経路情報のバージョン管理はやはり必要であると考えられる．また，機器の故障やメンテナンスによる機体の入れ替えの度に飛行経路に応じた書き換えが必要となる．同じ経路のプログラムを書き込んだ予備機体を用意することも考えられるが飛行させたい経路が増えるにつれてその予備台数も増加する事になってしまう．

次に考えられるのがグラウンドコントロールシステムによる中央集権的な管理である (以下中央管理型と表記する)．基本の飛行システムさえ書き込んでおけば経路情報と誘導はグラウンドコントールシステム側で一元管理が可能になり飛行前のアップデートの確認の手間やバージョン管理は不要になると考えられる．機体側は外部通信機構が必須になる上グラウンドコントロールシステム側での複数機体の管理，グラウンドコントロールシステムと各機体とのコミュニケーションが必要となる上に機体数が増えた場合にそのネットワークの帯域も問題となる可能性があるというデメリットがある．

3.2.2　倉庫内での自己位置推定

今回対象としている倉庫の巡回タスクにおいて自己位置推定は非常に重要な要素となる．

まず，現状の自己位置推定の手法とその問題点を以下にまとめる．

GPS を用いた自己位置推定

現在世界座標系における自己位置推定を行う上で頻繁に用いられるが，その利用可能環境には制限がある．それは屋内や遮蔽物の多い環境で衛星電波を受信出来ず利用できないという事である．加えて屋外の見通しの良い環境であっても誤差が平均して 1 2m 程あり[1]細かい制御には向かない．

一方で現在 GPS の精度向上や補助をする為のシステムが複数開発されている．その中の一つが A-GPS[1] と呼ばれ主に携帯電話やスマートフォンに搭載されているシステムである．A-GPS は GPS の信号強度が弱い場合や，建造物などの影響によるマルチパスが発生している際に利用可能なネットワークを利用して測位者の概ねの位置を取得する方式である．これにより GPS 測位時間の短縮や測位可能なエリアの拡大を実現している．

他に JAXA が開発した IMES(Indoor MEssaging System)[2] という屋内に GPS と同じ電波を利用した送信機を設置し細かい位置測位を行うという物もある．しかしこれらの方法は専用の送信機や通信機構を用意する必要がある．

SLAM を用いた自己位置推定

屋内で自己位置推定を行う方法として SLAM が挙げられる．

Visual SLAM

Visual SLAM はその特性上比較的多くの計算リソースを必要とする．加えて対象の輪郭などが確認できるくらいの明るさや色彩差がないと利用することが出来ない．映像が理想的な状態で得られたとしても手のひらサイズ程度の小型のロボット上で利用するには少々パワー不足となることが多い．更に今回想定しているようなリアルタイムでの計算が必要な状況においては計算リソースが非常に重要になる．また，この問題を避ける為計算サーバのようなものを用意する事を考えると複数機体利用する場合にネットワーク帯域や計算リソースが問題となり，解消できるような環境を用意する場合倉庫の巡回タスクにおいては費用面で実用には向かない．加えて Visual SLAM による環境地図作成と自己位置推定の精度には周辺の映像から抽出できる特徴量によって大きく左右される．例えば単色の背景が続くような環境や，ガラスなどの光透過性の高い環境では正確に環境地図の作成が行えない事が想定される．

[1]同研究室の水野史暁氏と竹瀬浩行氏と他数名によって慶應義塾大学湘南藤沢キャンパス WEST 地区にて計測

LiDAR SLAM

LiDAR SLAM は Visual SLAM に比べて計算リソースは不要なものの，そのセンササイズがカメラよりも大きい物が多い．これは小型のロボットに積載するには向かず，ドローンのような積載重量によって航行性能に大きく影響が出るようなロボットでは尚のこと利用が難しい．役割が異なるので純粋に価格で比較することは難しいが，センサ 1 個あたりの費用も SLAM に必要なスペックのものという前提で選出するとカメラモジュールよりも高くなる．加えて LiDAR SLAM においてもガラスなどの光透過性が高い環境では正確に環境地図の作成が行えない事が想定される．

TDOA を用いた自己位置推定

他に自己位置推定を行う方法として TDOA(Time Difference Of Arrival) が挙げられる．これは既知の波長の電波を複数箇所から射出し，ロボットではそれぞれの電波の到達時間の差によって自己位置推定を行う物である．

この方法では電波を射出する設備が必要な事に加えて電波ノイズに対しては脆弱であるとも考えられる．また，直接到達した電波なのか，反射して到達した電波なのか等も考慮して計算をしなければならない．

また GPS も衛星からの電波を利用した TDOA の一種であると言える．

3.2.3 専用機材の用意

倉庫の巡回タスクを実行するにあたって既存の手法を用いる場合それぞれ専用のセンサーを積載したロボットや機材を必要とする．専用センサーを積載したロボットを作成しようとするとロボットの機体サイズが大きくなってしまう．ドローンである程度自由にセンサーや制御用のボードを積載したものを現在流通しているパーツで自作するとなるとおよそ 1 辺 40cm 程の大きさになってしまい 3.1 狭い通路を飛行させる事を考慮すると目的に適さないと考えられる．

3.3 本研究における仮説

3.3.1 提案手法

今回上述した課題を解決する為に QR コードを用いた経路情報に対してステートレスなナビゲーションシステムを考案した．この方法には以下の利点が存在する

- RGB カメラが積載されていれば利用可能
- QR コードを用意するのは他の手法に比べ容易
- QR コードは非対称図形のため QR コードに対する相対的な位置推定が可能

上述した課題をどのように解決するかをそれぞれ以下にまとめる．

図 3.1: 自作機の写真

経路情報の管理

今回経路情報の管理をソフトウェアでは行わないという方法で課題となっていた部分を解決する．QR コードを一定間隔で棚に貼り付け，ドローンに積載されているカメラで読み取り経路情報を取得するという形をとる事にした．QR コードには次に向かうべき QR コードの座標データが相対座標系で記録されており，ドローンはこの情報を元に飛行計画を作成する事となる．

倉庫内での自己位置推定

本手法では QR コードに対する自身の相対的な位置を PnP 問題として計算して推定する．倉庫の巡回タスクにおいては倉庫全体絡みた自己位置推定はせずともタスクを遂行することが可能である．経路情報の項で述べたように QR コードに次に向かうべき座標データがその時点で観測している QR コードを原点とする座標系で記載されている．その為，QR コードに対する相対的な自己位置推定が出来れば想定される飛行経路に沿った飛行が可能となる．しかし，次の QR コードに移動する途中で QR コードを認識出来ない区間は

ドローン自体のオドメトリに頼った自己位置推定となり使用する機材に大きく依存した精度での制御となる．今回使用する機材ではモーターの回転数，ドローン下部に搭載された単眼カメラ，赤外線センサによるオドメトリでの自己位置推定を行っている．

専用機材の用意

既存手法では LiDAR SLAM では LiDAR センサとそれを積載可能なサイズのドローン，得られた点群データを処理できるだけの計算リソースが必要であった．また Visual SLAM ではカメラ映像より抽出される特徴量とそこからさらに計算される点群データを処理できるだけの計算リソースが必要であった．他に TDOA などの手法を用いる場合には電波の送信機と受信機などの機材が必要であった．さらに Visual/LiDAR SLAM を利用する場合は利用可能な環境に制約が発生する．3.2.2

一方で提案手法ではトイドローンにも積載されている単眼カメラと QR コードの検出と QR コードの姿勢推定さえ可能な計算リソースがあれば QR コードを印刷するだけで最低限の明るさが確保出来れば環境による制約もなく利用可能である．

3.3.2 各アプローチとの比較

以下に既存手法との比較を表にしたものを示す．

表 3.1: 各自己位置推定手法との比較

手法	推定に必要な機材	環境地図	拡大時の計算量	利用不可環境
Visual SLAM	RGB カメラ・イメージセンサ	有	増加	色変化の少ない環境
LiDAR SLAM	LiDAR	有	増加	光透過性の高い環境
TDOA	電波送信機/受信機	無	変化なし	電波ノイズの高い環境
提案手法	RGB カメラ/QR コード	無	変化なし	障害物の位置が変化する環境

表 3.2: 経路情報の管理方法の比較

手法	外部通信の有無	経路変更方法
埋め込み型	無	ソフトウェアアップデート
中央管理型	有	特になし
提案手法	無	QR コードの変更・貼り替え

第4章　提案手法

本章では提案手法について述べる．

4.1　概要

3 章で述べた仮説を検証するために，本研究の手法について解説する．

4.1.1　問題解決の為のアプローチ

本手法では QR コードの中心を原点とした座標系でドローンは自己位置推定を行う．この時，QR コードのデータ部をデコードし，同時に経路情報を取得する．4.1

データ部には以下のフォーマットでデータが格納されており，観測している QR コード座標系での次に向かうべき QR コードの座標データが記載されている．また，本システムで扱う距離単位は全て cm(センチメートル) である．

プログラム 4.1: data format

```
{
  "x"?: number;
  "y"?: number;
  "z"?: number;
  "rotate"?: number;
  "command"?: string;
}
```

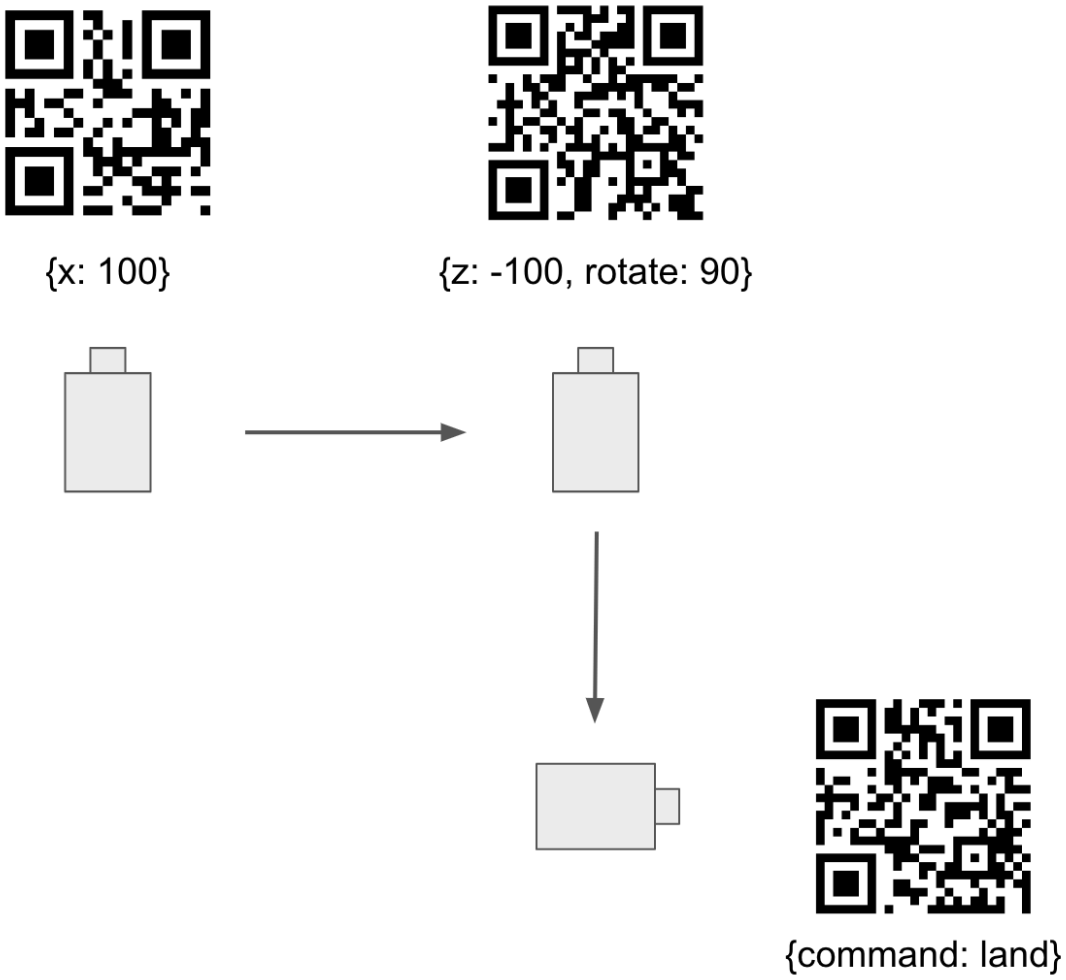

図 4.1: ナビゲーション時の簡略図

第5章　実装

本章では提案手法の実装について述べる．

5.1　実装環境

本研究では以下の環境で実装及び実行を行う．

- DJI Tello edu
- Ubuntu 18.04
- python 2.7

5.2　カメラキャリブレーション

はじめにカメラの内部パラメータ (焦点距離，歪み係数) を計算するためにカメラキャリブレーションを行う．ここで得られるパラメータはカメラ毎に固有の値となる為新しいカメラを利用する場合は毎回行う必要がある．

今回はOpenCV のフィンdChessboardCorners 関数を用いてチェスボードの格子点を検出し，そのデータを用いて calibrateCamera 関数を用いてキャリブレーションを行った．キャリブレーションには一マス 2.40cm の正方形が縦 7 マス横 10 マスに並ぶチェスボードと呼ばれる格子模様の画像 100 枚を利用してキャリブレーションを行った．

プログラム 5.1: calibration code

```
1  def calib(self, frame):
2      image = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
3      square_size = 2.4
4      pattern_size = (7, 10)
5      reference_img = 100
6
7      pattern_points = np.zeros((np.prod(pattern_size), 3), np.
           float32)
8      pattern_points[:, :2] = np.indices(pattern_size).T.reshape
           (-1, 2)
9      pattern_points *= square_size
10
11     if len(self.objpoints) < reference_img:
12         gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

```
13
14          ret, corner = cv2.findChessboardCorners(gray,
                pattern_size)
15          print(corner)
16          if ret is True:
17              print(str(len(self.objpoints)+1) + "/" + str(
                    reference_img))
18              term = (cv2.TERM_CRITERIA_EPS + cv2.
                    TERM_CRITERIA_COUNT, 30, 0.1)
19              cv2.cornerSubPix(gray, corner, (5, 5), (-1, -1),
                    term)
20              self.imgpoints.append(corner.reshape(-1, 2))
21              self.objpoints.append(pattern_points)
22
23          cv2.imshow('image', image)
24          if cv2.waitKey(200) & 0xFF == ord('q'):
25              pass
26      else:
27          gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
28          print("calculating␣camera␣parameter...")
29          ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(
30              self.objpoints, self.imgpoints, gray.shape[::-1],
                    None, None)
31
32          np.save("tello_mtx", mtx)
33          np.save("tello_dist", dist.ravel())
34          print("RMS␣=␣", ret)
35          print("mtx␣=␣\n", mtx)
36          print("dist␣=␣", dist.ravel())
```

5.3 映像部

今回はSDK[3] として用意されていたlibh264decoderというC++のプログラムをpythonから呼び出す形で利用した．また，そこから得られた画像を利用し易い形に以下プログラム 5.2 を用いて変換した．

ここではself.droneという変数で参照が可能となっている

プログラム 5.2: video loop method

```
1  def video_loop(self):
2      try:
3          time.sleep(0.5)
4          while not self.stopEvent.is_set():
5              frame = self.drone.read()
6              if frame is None or frame.size ==0:
7                  continue
8              self.pnp_qr(frame)
9      except( KeyboardInterrupt, SystemExit):
10         self.stopEvent.set()
11         print('exit␣video␣loop')
```

5.4 トラッカー部

トラッカー部の実装には OpenCV を用いる．

QR コードの検出とデコードには OpenCV に用意されている DetectQR クラスを利用し，そこから得られた QR コードの四角に対して PnP による 3 次元再構成を行った．PnP には OpenCV に用意されている solvePnPRansac 関数を利用した．

これにより QR コードよりデコードされた次の QR コードの位置情報と QR コードに対するカメラの回転ベクトル，並進ベクトルが得られるのでこれらの情報をナビゲーション部??にて処理をする

実際にスマートフォン上に表示した QR コードに対して姿勢推定結果を描画したものを以下の図 5.1 に示す

図 5.1: QR コードの姿勢推定結果

ソースコードの該当部分をそれぞれ以下のプログラム 5.3 に示す．以下のプログラムでは映像から QR コードの検出，デコード，姿勢推定を行いバッファに推定値を追加している．

プログラム 5.3: pnp method

```
1  def pnp_qr(self, frame):
2      img = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
3
4      qr = cv2.QRCodeDetector()
5      data, points, _ = qr.detectAndDecode(img)
6      if data:
7          self.qr_data = self._qr_validator(json.loads(data))
8          _, origin_rvec, origin_tvec, inliers = cv2.
               solvePnPRansac(
9              self.objp, points, self.mtx, self.dist)
10         if not self.is_moving:
11             self.target_pos = self.qr_data
12             tvec = origin_tvec * self.UNIT_SIZE
13             rvec = origin_rvec * self.RAD_UNIT
```

```
14              self.drone_pos_list.append(tvec)
15              self.drone_rotate_list.append(rvec)
16              if len(self.drone_pos_list) > 20:
17                  self.drone_pos_list.pop(0)
18                  self.drone_rotate_list.pop(0)
19              drone_avg_pos = np.mean(self.drone_pos_list, axis
                    =(0))
20              drone_avg_rotate = np.mean(self.drone_rotate_list,
                    axis=(0))
21              self.drone_pos = {
22                  'x': drone_avg_pos[0],
23                  'y': drone_avg_pos[1],
24                  'z': drone_avg_pos[2],
25                  'r': drone_avg_rotate[1]
26              }
27          imgpts, jac = cv2.projectPoints(
28              self.axis, origin_rvec, origin_tvec, self.mtx, self
                    .dist)
29          img = self.draw(img, points, imgpts)
30
31          cv2.imshow('img', img)
32          cv2.waitKey(10)
33      else:
34          cv2.imshow('img', img)
35          cv2.waitKey(10)
36      if not self.count in self.log_data:
37          self.log_data[self.count] = time.time()
38      self.out.write(img)
```

5.5 コントローラ部

制御の流れとしては以下の流れをとった.

1. QR コードが画角中央且つ距離 85cm の位置に来るように推定値を用いて制御する,
2. QR コードが上下左右距離が誤差± 10cm になっていれば次点へ移動する. なっていなければ 1 の処理を行う
3. QR コードの中の情報を利用して次の地点まで誘導する.
4. 以下 1,2,3 の繰り返し

この際推定誤差によって大きく外れた値が推定される事があり, 毎フレーム推定と制御を行うと外れ値を制御に利用してしまうこととなる. その為, 今回は 20 フレーム分バッファを作り, その平均値を推定値として扱うようにした. 実際に移動する際には以下の_move 関数 5.4 を用いており, 次の check_standby 関数 5.5 が True を返すようになるまで QR コードに対して誤差± 10cm になるように位置の補正を行う

プログラム 5.4: move method code

```
def _move(self, data):
    self.is_moving = True
    # move x
    if data["x"] >= 10:
        self.drone.move_right(data["x"])
        time.sleep(data["x"]/self.SPEED + self.
           COMMAND_WAIT_BUFFER)
        print('sleep␣x:␣', data["x"]/self.SPEED +
            self.COMMAND_WAIT_BUFFER)
    elif data["x"] <= -10:
        self.drone.move_left(-data["x"])
        time.sleep(-data["x"]/self.SPEED + self.
           COMMAND_WAIT_BUFFER)
        print('sleep␣x:␣', -data["x"]/self.SPEED +
            self.COMMAND_WAIT_BUFFER)
    # move y
    if data["y"] >= 10:
        self.drone.move_down(data["y"])
        time.sleep(data["y"]/self.SPEED + self.
           COMMAND_WAIT_BUFFER)
        print('sleep␣y:␣', data["y"]/self.SPEED +
            self.COMMAND_WAIT_BUFFER)
    elif data["y"] <= -10:
        self.drone.move_up(-data["y"])
        time.sleep(-data["y"]/self.SPEED + self.
           COMMAND_WAIT_BUFFER)
        print('sleep␣y:␣', -data["y"]/self.SPEED +
            self.COMMAND_WAIT_BUFFER)
    # move z
    if data["z"] >= 10:
        self.drone.move_forward(data["z"])
        time.sleep(data["z"]/self.SPEED + self.
           COMMAND_WAIT_BUFFER)
        print('sleep␣z:␣', data["z"]/self.SPEED +
            self.COMMAND_WAIT_BUFFER)
    elif data["z"] <= -10:
        self.drone.move_backward(-data["z"])
        time.sleep(-data["z"]/self.SPEED + self.
           COMMAND_WAIT_BUFFER)
        print('sleep␣z:␣', -data["z"]/self.SPEED +
            self.COMMAND_WAIT_BUFFER)
    # rotate r
    if data["r"] > 0:
        self.drone.rotate_cw(data["r"])
    else:
        self.drone.rotate_ccw(-data["r"])
    time.sleep(self.COMMAND_WAIT_BUFFER)
    # initialize values
    self.is_moving = False
    self.is_standby = False
    self.drone_pos_list = []
    self.drone_rotate_list = []
    return
```

プログラム 5.5: check standby method

```
def check_standby(self):
```

```
2       x_val = int(self.drone_pos["x"] - (self.QR_SIZE/2))
3       y_val = int(self.drone_pos["y"] - (self.QR_SIZE/2))
4       z_val = int(self.drone_pos["z"] - self.Z_SPACE)
5       rotate_val = int(self.drone_pos["r"])
6       self.is_standby = -self.permit_noise < x_val and self.
           permit_noise > x_val and \
7           -self.permit_noise < y_val and self.permit_noise >
              y_val and \
8           -self.permit_noise < z_val and self.permit_noise >
              z_val and \
9           -self.permit_noise < rotate_val and self.permit_noise >
               rotate_val
10      print('check:␣', x_val, y_val, z_val, rotate_val)
11      print('is_standby:␣', self.is_standby)
```

第6章　評価

本章では，提案システムの評価について述べる．

6.1　評価方法

倉庫棚を模した棚にQRコードを貼り付け，実際に棚の内容物の検品ができるかを確かめた．今回は実際に棚の内容物の映像を撮影する形で検品したと見なすこととした．

実運用ではRFIDリーダ等を装着し棚の中身を読み取る形や，商品にQRコードを貼り付けた形での検品を想定している．

第7章　結論

本章では，本研究のまとめと今後の課題を示す．

7.1　本研究のまとめ

今回のこのシステム実際に使う上では間違って他の QR コードを認識してしまったり，悪意ある人間によって QR コードがすり替えられていないかの検証システムが必要になってくる．その場合には jwt などの方式で QR コード内のデータに署名をする．という事で対応できるが，悪意ある人間がその環境に存在した場合 QR コードの場所が置き換えられる可能性や複製されてしまうと言う可能性は残る．

また，QR コードを LED パネルで表示する方法 [4] も考案されている為この方法を用いて QR コードを容易に変更・配置の記録をする事も可能になる．一方でどこにどの QR コードが配置されているかはシステム全体として保持しておきたい情報でもあり，LED パネルで QR コードを表示する手法は本研究との親和性が非常に高いと考えられる．

7.2　本研究の課題

付 録A　付録

A.1　付録内容

謝辞

参考文献

[1] 鈴木 喬, 青木 智治, 髙橋 誠, and 緒方 進. スマートフォン向け位置測位方式の高度化. NTT DOCOMO Technical Journal, 2014.

[2] 吉冨 進. 日本発の屋内外シームレス測位の実現へ. http://www.jaxa.jp/article/special/michibiki/yoshitomi_j.html, 2003.

[3] DJI. Tello sdk2.0. https://dl-cdn.ryzerobotics.com/downloads/Tello/Tello%20SDK%202.0%20User%20Guide.pdf.

[4] Ukida Hiroyuki, Miwa Masafumi, Tanimoto Yoshio, Sano Tetsuya, and Hideki Yamamoto. Visual uav control system using led panel and on-board camera. IEEE International Instrumentation and Measurement Technology Conference (I2MTC), 2013.