2024 年度卒業研究レポート

羽田空港における地上走行

明治大学　総合数理学部　現象数理学科

4 年 1 組 38 番　2610210013

松林茜里

2025 年　　2 月　27 日

目次

1. はじめに

現在羽田空港は、一日に約 1300 回の離着陸が行われている世界でもトップクラスの忙しさを誇ると言われている。そのため、現在でも羽田空港とその付近の上空は混雑している。しかし、インバウンド需要の増加から羽田空港の発着数をさらに増やすそうと考えられている。([4])また、航空会社も新たな海外の就航地を開拓し、より世界中と繋がろうとしている。したがって、羽田空港はより効率的で最適な滑走路や誘導路の使い方をしなければならない。

飛行機に搭乗した際、ドアが閉まってから離陸するまでの時間、そして、着陸してからドアが開くまでの時間が長いことがある。その時の飛行機によっても、長い場合と短い場合の両方がある。長いと感じる場合、大抵は、離陸時は誘導路の走行時間と滑走路の前での待機時間が長く、着陸時は誘導路の走行時間と誘導路からスポットまでの時間が長いと感じることが多い。

これらのタキシング時間の差はどこで変化をするのかを確認したいと考える。今後、羽田空港における発着便数が増える可能性が高い。つまり、離着陸の間隔が短くなるということである。今回は、離着陸の間隔を変化することで地上走行の時間にどのような変化があるのかシミュレーションをし、確認したい。

2. 用語

本論文を読むにあたり必要な用語の意味を記載する。

滑走路：　飛行機が離着陸を行う路

誘導路：　ターミナルと滑走路をつなぐ路

ターミナル：　飛行機が人や荷物を載せる・降ろす場所

スポット(エリア)：　飛行機を各々駐機するために定められた場所でターミナルを細分化したもの

タキシング：　駐機場から滑走路までの誘導路を車輪でゆっくりと走行させること

駐機：　スポットで飛行機が止まること

3. 羽田空港の滑走路の説明

3-1 滑走路のアルファベットと数字について

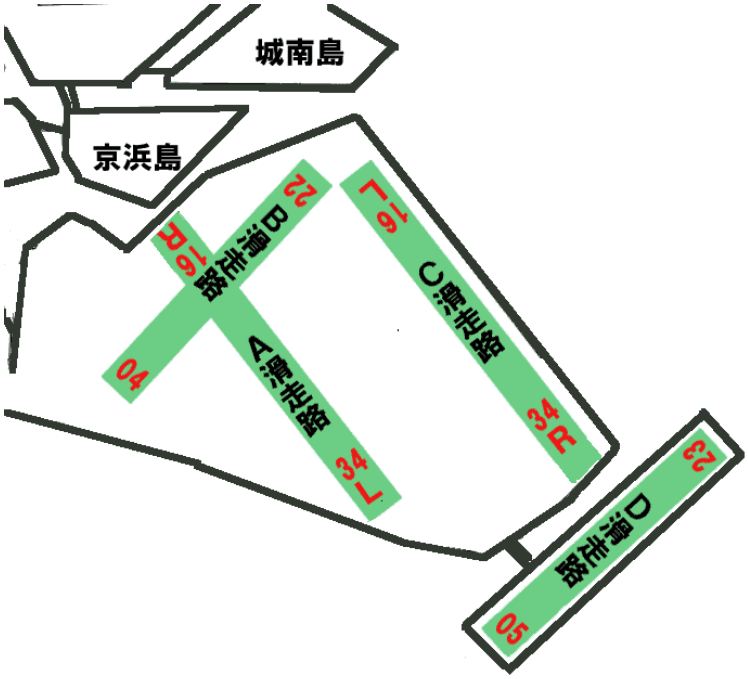
羽田空港は、以下[図 1]のように 4 本の滑走路がある。本論文では、34L, 34R のように説明をしていく。

16R または 34L： A 滑走路のこと

16L または 34R： C 滑走路のこと

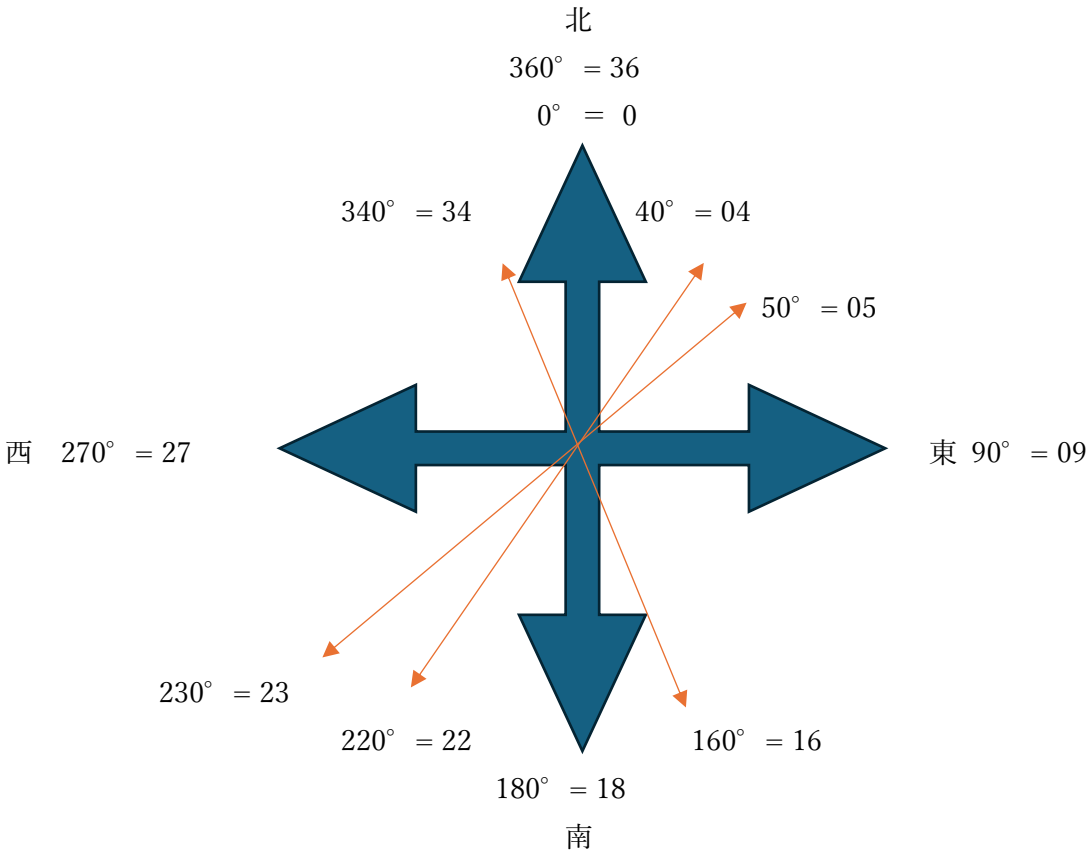
04 または 22： B 滑走路のこと

05 または 23： D 滑走路のこと



[図 1] 羽田空港の滑走路と名称([3])

滑走路は角度によって数字が定められている。[図 2]のように、北を 0° または 360° として時計回りに角度が増えていくようになっている。



［図 2］ 滑走路の数字と方角

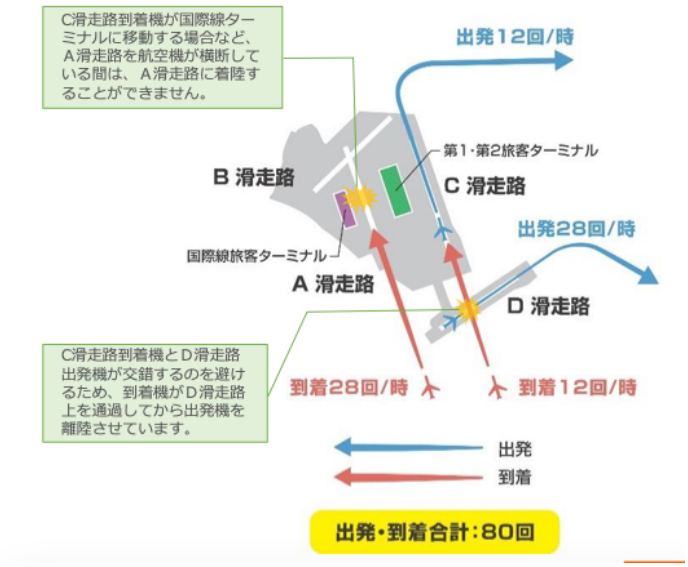
飛行機の管制などでは、滑走路名ではなく離着陸する方向を表した数字とアルファベットで離着陸の滑走路の指示をする。羽田空港のように方角が同じ滑走路の場合右と左で R と L がつけられる。

3-2　羽田空港での風向きの変化による滑走路の使い分け

飛行機は向かい風を受けて離着陸をする。その方が、離陸時には向かい風が強いほど早く浮かび上がり、着陸時には向かい風が強いほど早く減速をすることができるからである。そのため、羽田空港では風の方向によって使う滑走路や離着陸の方向を変えている。その使い方として大きく北風時と南風時の 2 つある。

北風時 昼に多い
34L　A 滑走路　着陸のみ

34R　C 滑走路　離着陸両方
05　　D 滑走路　離陸のみ

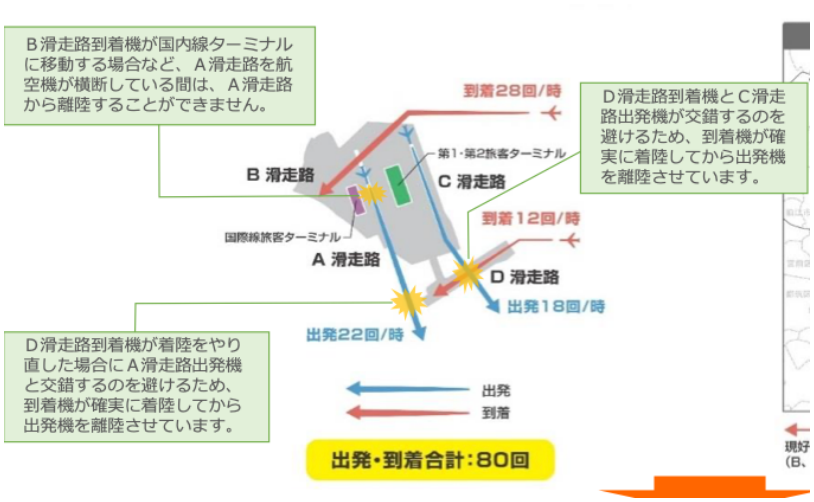


[図 3]　北風運用時の滑走路の使用方法([1])

北風時の運用の疑問点
「なぜ B 滑走路を使わないのか。B 滑走路と D 滑走路の方角はほぼ同じである。そのため、離陸で使うことは可能であると考える。D 滑走路を活用すれば C 滑走路の離陸を減らすことができ、その分着陸させることができるのではないのか。」という疑問が発生する。

この疑問の回答として滑走路と誘導路の交差を考える必要がある。
B 滑走路は A 滑走路と C 滑走路と交わっていてそこで着陸と離陸の間隔を考えなくてはならない。さらに誘導路のところで A 滑走路を横断するため、そこでも着陸との間隔を考えないといけない。そのため、B 滑走路を使うよりも C・D 滑走路を使った方が楽なのである。

南風時　夜に多い
16R　　A 滑走路　離陸
16L　　C 滑走路　離陸
22　　B 滑走路　着陸
23　　D 滑走路　着陸



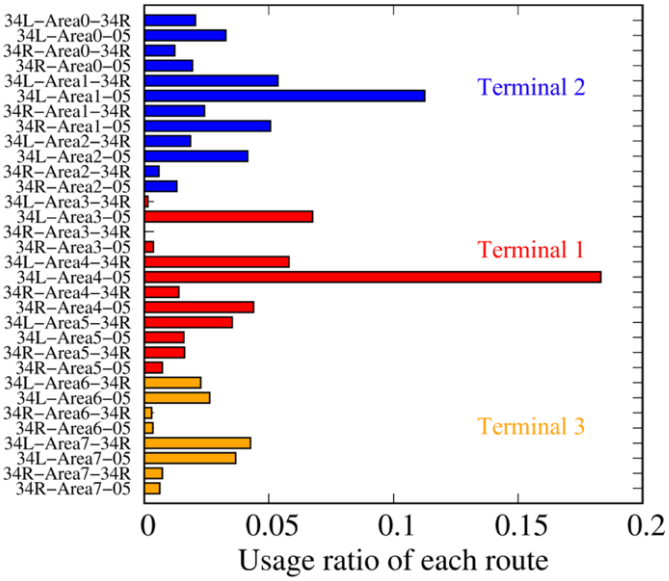
[図 4]　南風運用時の滑走路の使用方法([1])

今回のシミュレーションでは、着陸する滑走路の決定やその後の進むターミナル、離陸する滑走路の決定を以下の使用割合を使用し、確率的に扱うことにする。この使用割合は北風時のものしかないので、今回は南風時のことは考えずに北風時の時のみを考えることにする。

4.　滑走路の使用状況

4-1　使用割合（北風時）

羽田空港での北風運航では、着陸滑走路 2 本、エリア 8 つ、離陸滑走路 2 本で考えられ、32 の標準ルートに分割することができる。各路線の使用率が以下の[図 5]である。([6])



[図 5]　北風運用時の使用割合([6])

[図5]の利用率をシミュレーションで使用するため、数値をグラフから読み取ったものが[表1]となる。今回は、自分でグラフの読み取りをした。そのため、数値は全て大まかにあっていれば良いとし、合わせた数が 1 になっていないがそのままとする。シミュレーション時には合計が 1 になるように比率を計算し、分岐を決定する。

着陸滑走路	エリア	離陸滑走路	使用割合
34L	0	34R	0.021
34L	0	05	0.033
34R	0	34R	0.013
34R	0	05	0.019
34L	1	34R	0.053
34L	1	05	0.112
34R	1	34R	0.025
34R	1	05	0.051
34L	2	34R	0.018
34L	2	05	0.042
34R	2	34R	0.006
34R	2	05	0.014
34L	3	34R	0.002
34L	3	05	0.067
34R	3	34R	0
34R	3	05	0.004
34L	4	34R	0.057
34L	4	05	0.184
34R	4	34R	0.014
34R	4	05	0.045
34L	5	34R	0.036
34L	5	05	0.016
34R	5	34R	0.017
34R	5	05	0.007
34L	6	34R	0.023
34L	6	05	0.027
34R	6	34R	0.003
34R	6	05	0.004
34L	7	34R	0.043
34L	7	05	0.036
34R	7	34R	0.008
34R	7	05	0.007

[表 1] 使用割合([図 5]のグラフの値)

[表 1]ではターミナルの中にも駐機位置によってエリアで細分化されているが、今回はそれについては考慮せずに大きくターミナルごとで分けることにする。

着陸滑走路	ターミナル	離陸滑走路
34L	1	34R
34R	2	05
	3	

2 × 3 × 2

より 12 通りで表すことが可能になる。

[表 1]をまとめたものが[表 2]である。エリア 0,1,2 をターミナル 2 に、エリア 3,4,5 をターミナル 1 に、エリア 6,7 をターミナル 3 にまとめる。

着陸滑走路	ターミナル	離陸滑走路	使用割合
34L	2	34R	0.092
34L	2	05	0.187
34R	2	34R	0.044
34R	2	05	0.084
34L	1	34R	0.095
34L	1	05	0.267
34R	1	34R	0.031
34R	1	05	0.056
34L	3	34R	0.066
34L	3	05	0.063
34R	3	34R	0.011
34R	3	05	0.011

[表 2] ターミナルごとにまとめた使用割合

着陸滑走路から各ターミナルへの移動確率

	ターミナル 1 へ	ターミナル 2 へ	ターミナル 3 へ
R34L に着陸	0.362	0.279	0.129
R34R に着陸	0.087	0.128	0.022

[表 3] 着陸滑走路からターミナルへの移動確率

ターミナルから離陸滑走路への移動確率

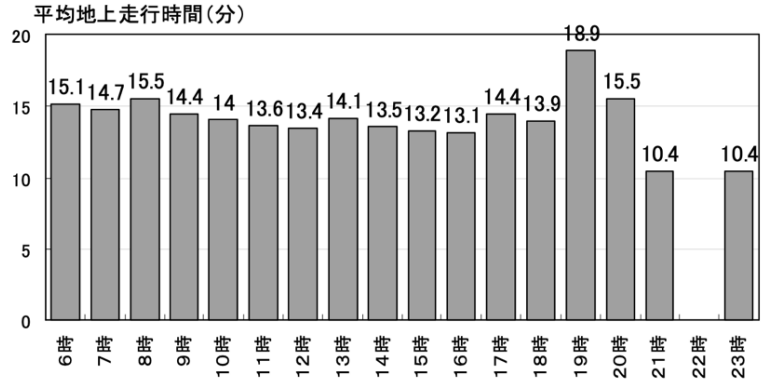
	ターミナル 1 から	ターミナル 2 から	ターミナル 3 から
R34R で離陸	0.126	0.136	0.077
R05 で離陸	0.323	0.271	0.074

[表 4] ターミナルから離陸滑走路への移動確率

[表 3],[表 4]の値を元に、プログラミングでの飛行機の挙動を確率で決める。

4-2　滑走路走行時間

羽田空港における地上走行時間を調べたが、最新のデータは得ることができなかった。[図 6]の平均地上走行時間は 2008 年 10 月 1 日~2009 年 3 月 31 日までの記録で、この記録の際には羽田空港の D 滑走路はまだできていない。そのため、現在とはタキシング距離や滑走路の運用方法が異なる可能がある。しかし、羽田空港のタキシング時間は 10 分から 20 分だと言われているため、とても大きな差異があるというわけではないのかもしれない。



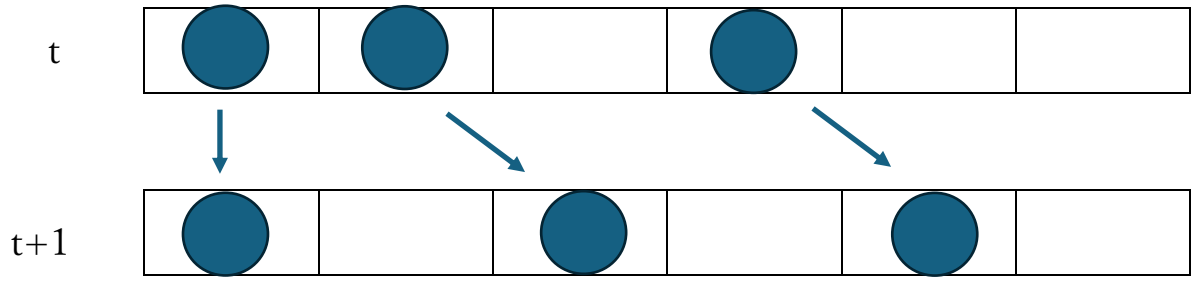
[図 6] 羽田空港における平均地上走行時間([2])

D 滑走路が新たにできたため、離着陸の便数が増やすことはできているのかもしれない。しかし、D 滑走路の位置はどのターミナルからも遠くなっている。そのため、タキシング時間は少なくても 10 分はかかってしまうのではないかと考えられる。

5. シミュレーション

5-1　セルオートマトン法について

セルオートマトン法とは空間に格子状に敷き詰められた多数のセルが、近隣のセルと相互作用をする中で自らの状態を変化させていくものである。今回は[図 7]のように前のセルが空いていたら進む、空いていなければその場で待機をするというルールで行なっていく。



[図 7] セルオートマトン法

5-2　セルオートマトンの設定

[6]でのセルオートマトンの設定の中で、今回のシミュレーションで使うことができる値を抜き出したものを以下に書く。

・セルの大きさ
空港全体を 2 次元で表した時の x 軸と y 軸が 1408×1126 セル
1 セルの大きさ 4.73m
飛行機が他の飛行機と衝突しないために前後左右に余裕を持った空間を設定する。それを衝突防止のアンテナという。
衝突防止のアンテナは前後に 24 セル 113.52m の長さ

・速度
タキシングの速度をエリア内、タキシングエリア、高速区間に分けている。

速度は$v_{tx} = \begin{cases} 5kt & エリア内 \\ 14kt & タキシングエリア \\ 20kt & 高速区間 \\ 140kt & 滑走路 \end{cases}$

kt はノットを表している。これを km/h と m/s にすると以下のようになる。

1kt = 1.852km/h = 0.5144m/s

5kt=9.26km/h=2.572m/s

14kt=25.928km/h=7.2016m/s

20kt=37.04km/h=10.288m/s

140kt=259.28km/h=72.016m/s

・滑走路占有時間(ROT)

滑走路占有時間とは、滑走路において前の飛行機が離着陸を行ってから次の飛行機が離着陸を行うことができるようになるまでの時間である。

離着陸間隔　120~130 秒

[6]のセルオートマトン法では滑走路の侵入から出口まで 40~50 秒がかかる。

したがって 120~130 秒の合計 ROT に対して 80 秒の追加時間が設定。つまり、飛行機が滑走路からいなくなったとしてもすぐに次の飛行機の離着陸を行うわけではなく、80 秒程度の時間を空けてから離着陸を行うことができる。

・着陸許可

着陸許可は飛行機が着陸する際に、滑走路内に飛行機がいないようにするために設ける時間である。

着陸許可　26 秒前

着陸許可発令後、他の飛行機は離陸・着陸・滑走路横断ができなくなる。

着陸許可は滑走路到着閾値から 1NM に達する前に発令される。NM はノーティカル・マイルで 1NM=1852m である。

最終侵入時の飛行速度が v_{tx} の滑走路の速度と考え 140kt とする。1NM 飛行する時間は、$\frac{1NM}{140kt} \cong 26s$となるので、26 秒前に発令される。

5-3　今回のシミュレーションでの設定

今回は先行研究の設定をそのまま使ったものと変更した点がある。

そのまま使用した点
・着陸許可　　26 秒前
飛行機が着陸する際に滑走路に侵入する 26 秒前から他の飛行機がその滑走路を使用した離陸・着陸・滑走路の横断ができなくなる。

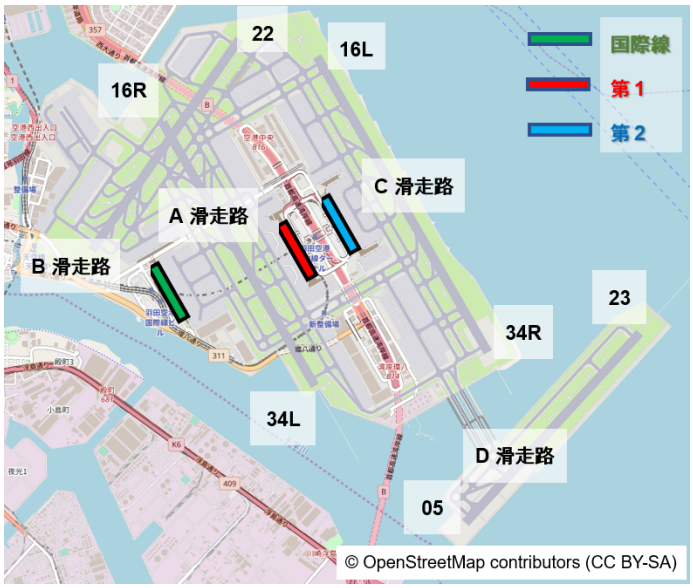
変更した点
・セルの大きさ　200m 程度
飛行機の大きさを 70m 程度、それにプラスで飛行機が他の飛行機と衝突しないために前後左右に余裕を持った空間を設定する。それを衝突防止のアンテナといい、113.52 メートルとする。そこから、1 セルの大きさを 200 メートル程度とした。

・タキシング速度　　14kt と 140kt 程度と想定
タキシングの速度は、滑走路と誘導路と大きく 2 つに分類した。誘導路では滑走路の 1/10 の速度で進む。滑走路での速度を 140kt と仮定して考えているので 1/10 で 14kt とした。滑走路では 1 ステップで 1 セル進み、誘導路では 10 ステップで 1 セル進む。1 ステップを現実での 3 秒程度と仮定している。

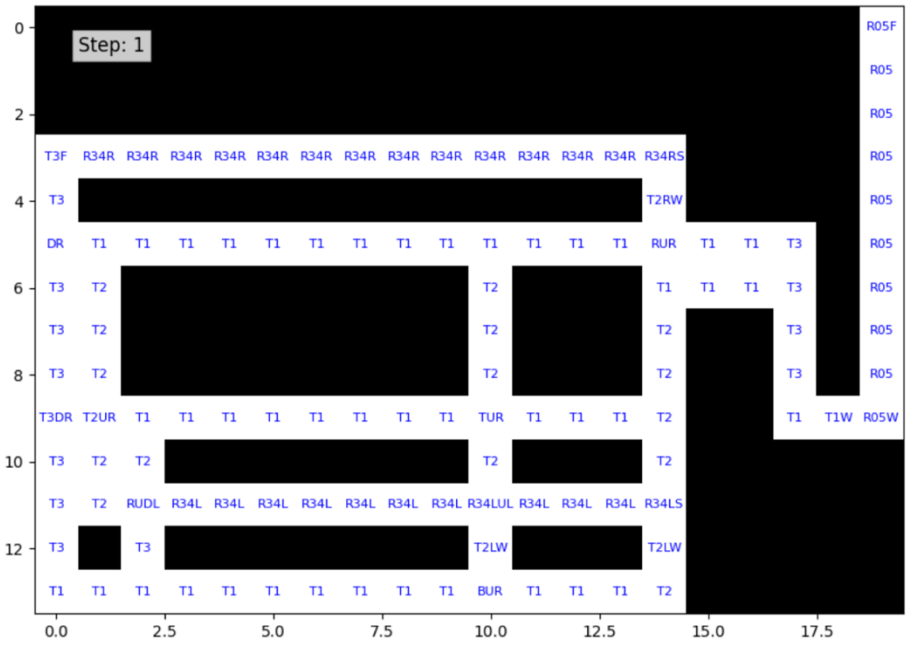
・滑走路占有時間　　変化させる
滑走路占有時間は、滑走路の侵入から出口までを 40~50 秒と計算し、着陸許可は 26 秒程度とした。追加時間を変化させるという設定で、連続着陸や連続離陸できる時間を変えて比較をした。

・その他の設定
今回は北風時の滑走路の使用を仮定してシミュレーションを行なった。
北風時なので、離陸専用の滑走路が 1 本、着陸専用の滑走路が 1 本、離着陸両方行う滑走路が 1 本となっている。
以下の図のように羽田空港の滑走路と誘導路を簡略化してシミュレーションを行う。

羽田空港の滑走路・誘導路は[図 8]のようになっている。これらの滑走路・誘導路を[図 9]のような略図にしてシミュレーションを行った。



[図 8] 羽田空港の滑走路と誘導路([5])



[図 9] シミュレーションを行う羽田空港の略図

[図 9]において R34RS-R34R-T3F となっている路が離着陸をする滑走路 34R。R34LS-R34L-R34LUL-R34L-RUDL となっている路が着陸滑走路 34L。R05W-R05-R05F となっている路が離陸滑走路 05。その他 T1,T2,T3, DR,T3DR,RUDL,BUR,T1W,T2RW,T2LW,T2UR, RUR,TUR は全て誘導路となっている。

R34RS と R34LS で飛行機は生成される。(着陸をイメージ)

T3F と R05F で飛行機は削除される。(離陸をイメージ)

5-4　各セルでのルール

図 9 におけるセルのルールを以下に示す。

T1: 右に進む

T2: 上に進む

T3: 下に進む

R34R: 左に進む

R34L: 左に進む

R05: 上に進む

R05F: 飛行機を削除する

DR,T3DR,RUDL,BUR: [表 3],[表 4]の確率によって分岐する

R34LUL:

　　R34LS で着陸した飛行機

　　　　左に進む

　　ターミナルを通過してきた離陸したい飛行機

　　　　上に進む

R34LS:

　　着陸機の場合

　　　　着陸許可の時間として 7 ステップ停止させる。その後左に進む。

　　T2LW から来た飛行機の場合

　　　　上に進む

R34RS:

　　着陸機の場合

　　　　着陸許可の時間として 7 ステップ停止させる。その後左に進む。

　　離陸機の場合

　　　　左に進む

T3F:

　　着陸機の場合

　　　　下に進む

　　離陸機の場合

　　　　飛行機を削除する

R05W:

R34RS に着陸許可を出された飛行機が存在する(着陸する飛行機がいる)

R34RS からいなくなるまで待機

R34RS に着陸許可を出された飛行機が存在しない(着陸する飛行機がいない)

上に進む

T1W:

R05W-R05-R05F に飛行機がいない

前の飛行機が R05W からいなくなって離陸間隔以上の時間が経過している

右に進む

前の飛行機が R05W からいなくなって離陸間隔未満しか経過していない

待機する

R05W-R05-R05F に飛行機がいる

待機する

T2RW:

R34RS-R34R-T3F に飛行機がいる

待機する

R34RS-R34R-T3F に飛行機がいない

前の飛行機が着陸か離陸をしてから離陸間隔以上の時間が経過している

上に進む

前の飛行機が着陸か離陸をしてから離陸間隔未満しか経過していない

待機する

T2LW:

R34LS-R34L-R34LUL-R34L-RUDL に飛行機がいる

待機する

R34LS-R34L-R34LUL-R34L-RUDL に飛行機がいない

上に進む

T2UR:

R34LS に着陸した飛行機の場合

上に進む

R34RS に着陸した飛行機の場合

右に進む

RUR:

TUR を通過してきた飛行機の場合

上に進む

TUR を通過していない飛行機の場合

[表 3],[表 4]の確率によって分岐する

TUR:

BUR を通過してきた飛行機の場合

上に進む

BUR を通過していない飛行機の場合

[表 3],[表 4]の確率によって分岐する

離陸点、交差点、合流点、分岐点の説明

離陸点

R05F と T3F が離陸点となる。R05F はその地点に飛行機が到達したら削除するようになっている。T3F は離陸機と着陸機があるため、その飛行機がどちらなのかを判別して処理を変更している。飛行機が離陸機の場合、必ず T2RW を通過している。そのため、T2RW の通過の有無で判別し、通過している場合は離陸機として飛行機を削除する。T2RW を通過していない着陸機の場合、T3F では必ず下方向に進む。

交差点

滑走路を横断する場合、着陸の飛行機がいないかつ、滑走路を走行している飛行機がいない場合横断をすることができる。誘導路の場合、進みたい交差するセルに飛行機がいないかつ、もう一方の前のセルに飛行機がいない場合はそのまま進むことができる。交差するセルの前のセルに異なる 2 方向から来ている場合、先に 10 ステップ経過した方が進むことができる。ステップ数も同じ場合は、着陸した時間が早い飛行機が先に進むことができる。

合流点

合流点に飛行機がいないかつ、もう一方の前のセルに飛行機がいなければそのまま進むことができる。もう 1 方の合流点の 1 個前のセルに飛行機がいる場合、先に 10 ステップを経過した方が進むことができる。ステップ数も同じ場合は、着陸した時間が早い飛行機が先に進むことができる。

分岐点

北風時の使用割合の値([表 3],[表 4])をもとに確率を求めている。その確率に従って、どの方向に進むのか決定している。

前提条件

飛行機の離着陸間隔を変更するため、3 つに分けた。

1. R34R での飛行機の着陸間隔が 70 ステップに 1 機で、R34L での着陸と、R34R,R05 での離陸間隔が 35 ステップに 1 機のシミュレーション
2. R34R での飛行機の着陸間隔が 80 ステップに 1 機で、R34L での着陸と、R34R,R05 での離陸間隔が 45 ステップに 1 機のシミュレーション
3. R34R での飛行機の着陸間隔が 75 ステップに 1 機で、R34L での着陸と、R34R,R05 での離陸間隔が 40 ステップに 1 機のシミュレーション

5-5　シミュレーションの結果

以下の[表 5]は離着陸の滑走路とターミナルごとに 12 通りの飛行機の挙動にかかったステップ数を確認するために、通過した時のステップ数を書いたものである。今回はターミナルを設定していないので、ターミナル付近のセルをターミナルと見立て、そのセルへの到着ステップ数を T1,T2,T3 に記録している。T3F,R05W,RUDL,R34RS,T1,T2,T3 に到着した時の経過ステップ数を記録し、それらの値から着陸滑走路からターミナルまでの経過ステップ数と、ターミナルから離陸滑走路までの経過ステップ数を計算している。
R34RS-1-05 の場合、着陸滑走路からターミナルまでのステップ数を T1-T3F で計算している。ターミナルから離陸滑走路までのステップ数を R05W-T1 で計算している。
具体的な例として、前提条件 1 の離着陸間隔が 35,70 の部分で見ると、着陸してからターミナルまでの経過は 217-93=124 ステップ、ターミナルから離陸するまでの経過は 409-217=192 ステップである。同様に、前提条件 2 では離着陸間隔が 45,80 の部分で見ると、着陸してからターミナルまでの経過は 2222-103=119 ステップ、ターミナルから離陸するまでの経過は 574-222=352 ステップである。これらの値を一番右の列にまとめている。

着陸滑走路 -ターミナル -離陸滑走路	前提条件 1 の 離着陸間隔が 35,70 の時の ステップ	前提条件 2 の 離着陸間隔が 45,80 の時の ステップ	前提条件 3 の 離着陸間隔が 40,75 の時の ステップ	前提条件 1~3 の値 をまとめたもの
R34RS – 1 – 05	T3F 93 T1 217 R05W 409	T3F 103 T1 222 R05W 574	T3F 173 T1 309 R05W 479	ターミナルまで 124,119,136 滑走路まで 192,352,170
R34RS – 2 – 05	T3F 23 T2 102 R05W 374	T3F 23 T2 102 R05W 384	T3F 23 T2 102 R05W 389	ターミナルまで 全て 79 滑走路まで 272,282,287
R34RS – 3 – 05	T3F 93 T3 273 R05W 551	T3F 263 T3 425 R05W 834	T3F 23 T3 199 R05W 569	ターミナルまで 180,162,176 滑走路まで 278,409,370
R34LS – 1 - 05	RUDL 91 T1 155 R05W 344	RUDL 156 T1 228 R05W 569	RUDL 21 T1 79 R05W 349	ターミナルまで 64,72,58 滑走路まで 189,341,270
R34LS – 2 – 05	RUDL 56 T2 184 R05W 479	RUDL 66 T2 192 R05W 564	RUDL 21 T2 142 R05W 429	ターミナルまで 128,126,121 滑走路まで 295,372,287
R34LS – 3 – 05	RUDL 21 T3 94 R05W 309	RUDL 202 T3 268 R05W 654	RUDL 141 T3 215 R05W 429	ターミナルまで 73,66,74 滑走路まで 215,386,214
R34RS – 1 – R34R	T3F 93 T1 239 R34RS 448	T3F 23 T1 142 R34RS 299	T3F 98 T1 217 R34RS 379	ターミナルまで 146,119,119 滑走路まで 209,157,162
R34RS – 2 – R34R	T3F 175 T2 252 R34RS 359	T3F 183 T2 265 R34RS 393	T3F 248 T2 332 R34RS 538	ターミナルまで 77,82,84 滑走路まで 107,128,206
R34RS – 3 – R34R	T3F 23 T3 189 R34RS 350	T3F 156 T3 334 R34RS 558	T3F 23 T3 199 R34RS 461	ターミナルまで 166,178,176 滑走路まで 161,224,262
R34LS – 1 – R34R	RUDL 91 T1 155 R34RS 334	RUDL 156 T1 212 R34RS 334	RUDL 61 T1 125 R34RS 286	ターミナルまで 64,56,64 滑走路まで 179,122,161
R34LS – 2 – R34R	RUDL 56 T2 182 R34RS 274	RUDL 66 T2 205 R34RS 377	RUDL 61 T2 209 R34RS 294	ターミナルまで 126,139,148 滑走路まで 92,172,85
R34LS – 3 – R34R	RUDL 337 T3 415 R34RS 586	RUDL 202 T3 268 R34RS 493	RUDL 385 T3 439 R34RS 620	ターミナルまで 78,66,54 滑走路まで 171,225,181

[表 5] 前提条件ごとのステップ数とそのまとめ

着陸からターミナルまでの平均時間

	ステップ数	現実の時間(分)
全ての	107	6~7
R34R 着陸	126.72	7~8
R34L 着陸	87.61	5~6

ターミナルから離陸までの平均時間

	ステップ数	現実の時間(分)
全て	227	13~14
R34R 離陸	166.88	9~10
R05 離陸	287.83	16~17

[表 6]離着陸における平均タキシング時間

離着陸の間隔を気にせず、全ての結果についてまとめたものが[表 6]となる。この結果から、ターミナルから離陸するまでの移動時間は、着陸してからターミナルまでの移動時間の約 2 倍という結果となった。今回はターミナルの設定はしていないので、ここからさらに誘導路を移動し、スポットまで移動する時間がかかると考えると 10 分~20 分程度かかると考えられる。そのように考えると、4-2 の地上走行時間とほとんど同じと言う結果となっている。[図 6]は 2008 年なので、現在の D 滑走路はまだできていないと言うことを考えると、D 滑走路の新設により羽田空港の離着陸できる飛行機数は増加した。しかし、D 滑走路は海の上にあり、誘導路の距離も大幅に増加した。その結果、タキシング時間はあまり変わっていないのではないかと考えられる。

	35,70(離着陸回数多)		45,80(離着陸回数少)		40,75	
	ステップ	時間(分)	ステップ	時間(分)	ステップ	時間(分)
着陸全体	108.75	6.217	105.33	6.0742	107.4166	6.19436
R 着陸	128.66	7.4197	123.166	7.1026	128.33	7.4005
L 着陸	88.83	5.1227	87.5	5.0458	86.5	4.98816
離陸全体	196.66	11.341	264.166	15.2336	221.25	12.75875
R 離陸	153.166	8.8326	171.33	9.8802	176.166	10.15894
05 離陸	240.16	13.8496	357	20.587	266.33	15.3585

[表 7]　離着陸間隔ごとの平均タキシング時間

今回の目的である離着陸の時間間隔の変化による結果を表したのが上表である。[表 7]より、着陸に関しては、離着陸の間隔の増減に伴っての時間変化に、大きな差はなかった。しかし、離陸に関しては、離着陸の間隔が短い方がタキシング時間は短い結果となった。

5-6　結果からの考察

今回の結果については、初期の飛行機数が少ないため、以上のような結果となったと考える。今回はステップ数をそこまで大きくせずにシミュレーションを行った。そのため、離着陸の間隔の短縮による、タキシングする飛行機数の増加がそこまで大きく反映されなかったのではないかと考える。実際に、シミュレーション内の最後の方のステップになった時、R05 で離陸したい飛行機の行列は離着陸の間隔を短くしたシミュレーションの方が長いことが多かった。そのため、今回のシミュレーションはタキシングしている飛行機が少ない時。つまり、朝早い時間などの場合と考えるのがいいのではないかと考える。
今回のようなタキシングをしている飛行機が少ない場合、さらに離着陸の間隔を短くすることでよりタキシング時間は短縮することができるかもしれない。しかし、飛行機は後方乱気流の影響などを考え、連続離着陸の時間間隔の短縮には限界がある。そのため、R05 離陸において、現在よりも飛行機を増やしつつ、安全性を守ることは難しいのではないかと考える。その結果、タキシング時間はある程度までは減少しても、その後双曲線を描くようにまた増加してしまうのではないかと考える。

6.　今後

6-1　今後行いたい比較

今回は、離着陸の間隔のみを変化させ、それに応じて、タキシングの時間の増減を確認した。しかし、空港における飛行機の離着陸の間隔の減少には限度があると考える。そのため、離着陸の間隔を同じにしたまま、飛行機の台数を増やすなどし、それによるタキシング時間の変化などを観察してみたいと考える。
さらに、初期のタキシングをしている飛行機の数を増加させることで、より忙しい時間の空港の再現をすることができると考える。そこで、離着陸の間隔を変化させることで、今回とは異なった結果を得ることができるのではないかと考える。

6-2　より現実に近づけるための設定

今回のシミュレーションでは飛行機の挙動は 8 割程度がうまく動いているが、たまに意図していない挙動をする飛行機が発生してしまっている。主に、分岐・交差の部分である。飛行機の離着陸の時間調整のために遠回りをすることがないとも限らないが、あまり現実的ではないと考えるため、変更をしたい。
さらに、今回はシミュレーションに組み込むことができなかったが、実際に羽田空港を模すために行いたかった設定を以下に示しておく。

・空港の設定
今回は誘導路の本数を減らしてシミュレーションを行った。そのため、実際には渋滞にならないところで渋滞が発生してしまった。そのため、少なくても後 1 本、R34R の下に誘導路を増やしたい。

・ターミナルの設定
今回はターミナルを設定していない。しかし、現実にはターミナルがあり、さらにはそのターミナルの中にもエリアがあり、その中にスポットがある。空きスポットがある場合は、飛行機はそのスポットに駐機する。空きがない場合にはその手前の誘導路で待機するように設定をしたい。ターミナルでの駐機時間は国内線では 45 分以下、国際線では 90 分以下に設定したい。そのため、ターミナル 1,2 の飛行機は 45 分以下、ターミナル 3 では 95 分以下に設定をし、その時間が経過しないと飛行機は動き出さないように設定したい。

・タキシング速度
タキシング速度においては、[6]のように誘導路の場所に応じて、変化をさせたい。

・交差、合流する場所での優先順位
今回は 10 ステップにつき 1 セル進むので、先に 10 ステップ進んだ方が動き、そのステップも同じ場合には先に着陸していた飛行機が優先されるようになっている。しかし、その結果、何機か待つことになり、なかなか進むことができない飛行機が発生した。そのため、明確な基準を定義し、1 機は待つことができるが 2 機は待たないなどの設定を行いたい。

7. まとめ

今回羽田空港の地上滑走についてシミュレーションを行った。飛行機の離着陸の間隔を変化させ、それによってタキシング時間がどのように変化をするのかを調べた。初期の飛行機数が少ない場合(現実での朝などの想定)では、飛行機の離着陸の間隔が短い方がタキシング時間は短かった。しかし、ステップ数が増加するにつれて、離陸したい飛行機の行列が伸びていった。そのため、ステップ数を増加させた場合、今回とは異なる結果となる可能性も十分にあり得ると考える。

8. 付録　プログラム

```
import matplotlib.pyplot as plt
import numpy as np
import random
import time
from matplotlib.animation import FuncAnimation
from IPython.display import HTML

# 初期のセルオートマトン
rectangle = np.array([
    [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
    [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
    [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
    [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,1],
    [1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1],
    [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1],
    [1,1,0,0,0,0,0,0,0,0,1,0,0,0,1,1,1,1,0,1],
    [1,1,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,1,0,1],
    [1,1,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,1,0,1],
    [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,1,1,1],
    [1,1,1,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0],
    [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0],
    [1,0,1,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0],
    [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0],

])

# 滑走路と誘導路に名前をつけた新しい配列
label_grid = np.array([
    [" ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "R05F"],
    [" ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "R05"],
    [" ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "R05"],
    ["T3F", "R34R", "R34R", "R34R", "R34R", "R34R", "R34R", "R34R", "R34R", "R34R", "R34R", "R34R",
"R34R", "R34R", "R34RS", " ", " ", " ", " ", " ", "R05"],
    ["T3", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "T2RW", " ", " ", " ", " ", " ", "R05"],
    ["DR", "T1", "T1", "T1", "T1", "T1", "T1", "T1", "T1", "T1", "T1", "T1", "T1", "T1", "T1", "T1", "RUR", "T1", "T1", "T3",
" ", "R05"],
    ["T3", "T2", " ", " ", " ", " ", " ", " ", " ", " ", " ", "T2", " ", " ", " ", "T1", "T1", "T1", "T3", " ", "R05"],
    ["T3", "T2", " ", " ", " ", " ", " ", " ", " ", " ", " ", "T2", " ", " ","," , "T2", " ", " ", "T3", " ", "R05"],
    ["T3", "T2", " ", " ", " ", " ", " ", " ", " ", " ", " ", "T2", " ", " ","," , "T2", " ", " ", "T3", " ", "R05"],
    ["T3DR",        "T2UR",        "T1","T1","T1","T1","T1","T1","T1","T1","TUR","T1","T1","T1","T2","         ","
","T1","T1W","R05W"],
    ["T3","T2", "T2", " ", " ", " ", " ", " ", " ", " ", " ", "T2", " ", " ", " ", "T2", " ", " ", " ", " ", " ", " "],
    ["T3","T2", "RUDL", "R34L",   "R34L", "R34L", "R34L", "R34L", "R34L", "R34L", "R34LUL", "R34L",
"R34L", "R34L", "R34LS", " ", " ", " ", " ", " ", " "],
    ["T3"," ", "T3", " ", " ", " ", " ", " ", " ", " ", " ", "T2LW", " ", " ", " ", " ", "T2LW", " ", " ", " ", " ", " ", " "],
    ["T1","T1", "T1","T1","T1","T1","T1","T1","T1","T1","T1","BUR","T1","T1","T1","T2", " ", " ", " ", " ", " ", " "]
])

particles = [(0, 10), (3, 5), (16, 5), (17, 6), (4, 9), (14, 10), (9, 13), (18, 9), (14, 6), (12, 5), (17, 8),(11, 9)]
```

```
# アニメーション用の初期設定
fig, ax = plt.subplots(figsize=(10, 8))
ax.imshow(rectangle, cmap='gray')
scatter = ax.scatter([], [], c='red')

# ステップ数表示用のテキストオブジェクトを作成
step_text = ax.text(0.05, 0.95, '', transform=ax.transAxes, fontsize=12,
                    verticalalignment='top', bbox=dict(facecolor='white', alpha=0.8))

# 保存用のリスト
particle_history = []

# R34LS で新たに生成された飛行機のリスト
newly_generated_particles = []

# passed_positions をループ外で定義
passed_positions = {}  # {飛行機 ID: {位置名: 通過回数}}
# 飛行機ごとの移動カウンタを格納する辞書
move_counters = {}  # {飛行機 ID: 移動カウンタ}

# 飛行機の ID を格納するリスト
plane_ids = []

# 飛行機が最後に生成されたステップ
last_generated_step = -40  # 最初に飛行機を生成するために-40 で初期化
last_generated_step1 = -75

# R05W から飛行機がいなくなったステップを記録する変数
last_r05w_exit_step = -40  # 初期値は-40 (最初の飛行機がすぐに生成されるように)
# R34RS から飛行機がいなくなったステップを記録する変数
last_r34rs_exit_step = -40  # 初期値は-40 (最初の飛行機がすぐに生成されるように)

# R34RS で飛行機が生成されてからの経過ステップ数を追跡
r34rs_generation_step = {}
# R34LS で飛行機が生成されてからの経過ステップ数を追跡
r34ls_generation_step = {}


def is_cell_occupied(x, y):
    return any((px, py) == (x, y) for px, py in particles)

# アニメーションの更新関数
for step in range(525):
    particle_history.append(np.zeros_like(rectangle))
    for x, y in particles:
        particle_history[-1][y, x] = 1
    new_particles = []

    # パーティクルの移動ロジック
    for index, (x, y) in enumerate(particles):
        particle_id = index  # 各飛行機に一意の ID を付与

        # 通過地点ごとの処理
        current_position = label_grid[y, x]  # 現在の位置を取得
```

```
            if particle_id not in passed_positions:
                passed_positions[particle_id] = {}  # 初めて通過する飛行機のために辞書を初期化

            # 移動カウンタの初期化
            if particle_id not in move_counters:
                move_counters[particle_id] = 0

            if current_position in ["T3F","R34LS", "R34L", "R34R", "R34RS", "T1", "R05", "T2","T3","RUR",
"RUDL", "DR","BUR","TUR", "T3DR","T2UR","R34LUL","T2RW","T2LW","T1W","R05W"]:
                if current_position not in passed_positions[particle_id]:
                    passed_positions[particle_id][current_position] = 1  # 初めて通過した場合
                else:
                    passed_positions[particle_id][current_position] += 1  # 通過回数を増加

            # T3F の処理（通過回数に応じて行動を変更）
            if current_position == "T3F":
                if "T2RW" in passed_positions[particle_id] or "RUR" in passed_positions[particle_id]:
                    # 2 回目の通過ならその飛行機は消す
                    continue  # 飛行機を削除（リストから除外）
                elif "T2RW" not in passed_positions[particle_id] and "RUR" not in
passed_positions[particle_id]:
                    # 1 回目の通過なら下方向に進む
                    neighbors = [(x, y+1)]  # 下方向のみ

            elif label_grid[y, x] == "R34LS":
                # 新しく生成された飛行機の処理
                if (x, y) in newly_generated_particles:
                    if (x, y) not in r34ls_generation_step:
                        r34ls_generation_step[(x, y)] = step  # 生成ステップを記録
                    steps_since_generation = step - r34ls_generation_step[(x, y)]

                    if steps_since_generation < 7:
                        # 7 ステップ未満の場合は待機
                        neighbors = [(x, y)]
                    else:
                        # 7 ステップ以上経過したら左方向に移動
                        neighbors = [(x-1, y)]
                        del r34ls_generation_step[(x, y)]  # 移動後、生成ステップの記録を削除
                        newly_generated_particles.remove((x, y))
                else:
                    # 既に存在する飛行機は上方向に動く
                    neighbors = [(x, y-1)]  # 上方向のみ

            # 他の位置での動作を処理
            elif current_position == "R34L":
                neighbors = [(x-1, y)]  # 左方向のみ
            elif current_position == "R34R":
                neighbors = [(x-1, y)]  # 左方向のみ
            elif current_position == "R34RS":
                # 新しく生成された飛行機の処理
                if (x, y) in newly_generated_particles:
                    if (x, y) not in r34rs_generation_step:
                        r34rs_generation_step[(x, y)] = step  # 生成ステップを記録
```

```
            steps_since_generation1 = step - r34rs_generation_step[(x, y)]

            if steps_since_generation1 < 7:
                # 7 ステップ未満の場合は待機
                neighbors = [(x, y)]
            else:
                # 7 ステップ以上経過したら左方向に移動
                neighbors = [(x-1, y)]
                last_r34rs_exit_step = step
                # 移動後、生成ステップの記録を削除
                del r34rs_generation_step[(x, y)]
                newly_generated_particles.remove((x, y))

        else:
            neighbors = [(x-1, y)]  # 左方向のみ


    elif current_position == "R05":
        neighbors = [(x, y-1)]  # 上方向のみ
    elif current_position == "T1":
        if move_counters[particle_id] % 10 == 0:
            if not is_cell_occupied(x+1, y):  # 前のセルが空いているかチェック:
                neighbors = [(x+1, y)]  # 右方向のみ
            else:
                neighbors = [(x, y)]
        else:
            neighbors = [(x, y)]
        move_counters[particle_id] += 1
    elif current_position == "T2":
        if move_counters[particle_id] % 10 == 0:
            if not is_cell_occupied(x, y-1):  # 前のセルが空いているかチェック:
                neighbors = [(x, y-1)]  # 上方向のみ
            else:
                neighbors = [(x, y)]
        else:
            neighbors = [(x, y)]
        move_counters[particle_id] += 1
    elif current_position == "T3":
        if move_counters[particle_id] % 10 == 0:
            if not is_cell_occupied(x, y+1):  # 前のセルが空いているかチェック:
                neighbors = [(x, y+1)]  # 下方向のみ
            else:
                neighbors = [(x, y)]
        else:
            neighbors = [(x, y)]
        move_counters[particle_id] += 1
    elif current_position == "RUR":
        if move_counters[particle_id] % 10 == 0:
            if "TUR" in passed_positions[particle_id] or "T2" in passed_positions[particle_id]:
                if not is_cell_occupied(x, y-1):  # 前のセルが空いているかチェック
                    neighbors = [(x,y-1)] # 上方向のみ
                else:
                    neighbors = [(x, y)]
            elif "TUR" not in passed_positions[particle_id] and "BUR" not in
```

```
passed_positions[particle_id]:
                        #2 タミから R 離陸が 0.136,05 離陸が 0.271
                        if random.random() < 0.33:
                            if not is_cell_occupied(x, y-1):  # 前のセルが空いているかチェック
                                neighbors = [(x, y-1)]  # 上方向のみ
                            else:
                                neighbors = [(x, y)]
                        else:
                            if not is_cell_occupied(x+1, y):  # 前のセルが空いているかチェック
                                neighbors = [(x+1, y)]  # 右方向のみ
                else:
                    neighbors = [(x, y)]
                move_counters[particle_id] += 1
            elif current_position == "RUDL":
                if random.random() < 0.47:
                    if not is_cell_occupied(x, y-1):  # 前のセルが空いているかチェック
                        neighbors = [(x, y-1)]  # 上に移動
                elif random.random() < 0.47 + 0.36:  # 0.47 + 0.36 = 0.83
                    if not is_cell_occupied(x-1, y):  # 前のセルが空いているかチェック
                        neighbors = [(x-1, y)]  # 左に移動
                else:
                    if not is_cell_occupied(x, y+1):  # 前のセルが空いているかチェック
                        neighbors = [(x, y+1)]  # 下に移動 (残りの 0.17)

            elif current_position == "DR":
                if move_counters[particle_id] % 10 == 0:
                    if random.random() < 0.54:
                        if not is_cell_occupied(x, y+1):  # 前のセルが空いているかチェック
                            neighbors = [(x, y+1)]
                    else:
                        if not is_cell_occupied(x+1, y):  # 前のセルが空いているかチェック
                            neighbors = [(x+1, y)]  # 下右方向のみ
                else:
                    neighbors = [(x, y)]
                move_counters[particle_id] += 1
            elif current_position == "BUR":
                if move_counters[particle_id] % 10 == 0:
                    #3 タミから R 離陸が 0.077,05 離陸が 0.074
                    if random.random() < 0.51:
                        if not is_cell_occupied(x, y-1):  # 前のセルが空いているかチェック
                            neighbors = [(x, y-1)] # 上方向のみ
                    else:
                        if not is_cell_occupied(x+1, y):  # 前のセルが空いているかチェック
                            neighbors = [(x+1, y)]  # 右方向のみ
                else:
                    neighbors = [(x, y)]
                move_counters[particle_id] += 1
            elif current_position == "TUR":
                if move_counters[particle_id] % 10 == 0:
                    if "BUR" in passed_positions[particle_id] and "T2LW" in passed_positions[particle_id]:
                        neighbors = [(x, y-1)]
                    elif "BUR" not in passed_positions[particle_id] or "T2LW" not in
passed_positions[particle_id]:
                        #1 タミから R 離陸が 0.126,05 離陸が 0.323
```

```
                    if random.random() < 0.28:
                        if not is_cell_occupied(x, y-1):  # 前のセルが空いているかチェック
                            neighbors = [(x, y-1)] # 上方向のみ
                    else:
                        if not is_cell_occupied(x+1, y):  # 前のセルが空いているかチェック
                            neighbors = [(x+1, y)]  # 右方向のみ
                #else:
                    #neighbors = [(x, y-1)]
            else:
                neighbors = [(x, y)]
            move_counters[particle_id] += 1

        elif current_position == "T3DR":
            if move_counters[particle_id] % 10 == 0:
                if random.random() < 0.8:
                    if not is_cell_occupied(x+1, y):  # 前のセルが空いているかチェック
                        neighbors = [(x+1, y)]
                else:
                    if not is_cell_occupied(x, y+1):  # 前のセルが空いているかチェック
                        neighbors = [(x, y+1)]
            else:
                neighbors = [(x, y)]
            move_counters[particle_id] += 1
        elif current_position == "T2UR":
            if move_counters[particle_id] % 10 == 0:
                if "T3DR" in passed_positions[particle_id]: #and passed_positions[particle_id]["T3DR"] ==
1:
                    # R34R に着陸した機体なら右方向に進む
                    if not is_cell_occupied(x+1, y):  # 前のセルが空いているかチェック
                        neighbors = [(x+1, y)]  # 右方向のみ
                    else:
                        neighbors = [(x, y)]
                elif "RUDL" in passed_positions[particle_id] or "T2" in passed_positions[particle_id]:
                    if not is_cell_occupied(x, y-1):  # 前のセルが空いているかチェック
                        neighbors = [(x, y-1)]  # 上方向のみ
                    else:
                        neighbors = [(x, y)]
                else:
                    neighbors = [(x+1, y)]
            else:
                neighbors = [(x, y)]
            move_counters[particle_id] += 1
        elif current_position == "R34LUL":
            #if "BUR" not in passed_positions[particle_id] and "T2LW" not in passed_positions[particle_id]:
             #    neighbors = [(x-1, y)]  # 左方向のみ
            if "BUR" in passed_positions[particle_id] or "T2LW" in passed_positions[particle_id]:
                if not is_cell_occupied(x, y-1):  # 前のセルが空いているかチェック
                    neighbors = [(x, y-1)]  # 上方向のみ
                    # R34LUL に到達した飛行機を newly_generated_particles リストから削除
                    if (x, y) in newly_generated_particles:
                        newly_generated_particles.remove((x, y))
                else:
                    neighbors = [(x, y)]
            else:
```

```
                neighbors = [(x-1, y)]  # 左方向のみ

            elif current_position == "T2RW":
                if move_counters[particle_id] % 10 == 0:
                    # R34RS、R34R、T3F に飛行機がいない場合、かつ次に R34RS で飛行機が生成されない場合
                    r34rs_cells = [(x, y) for y in range(label_grid.shape[0]) for x in range(label_grid.shape[1]) if label_grid[y, x] == "R34RS"]
                    r34rs_occupied = any((x, y) in particles for (x, y) in r34rs_cells)  # R34RS に飛行機がいるかどうか判定
                    r34r_cells = [(x, y) for y in range(label_grid.shape[0]) for x in range(label_grid.shape[1]) if label_grid[y, x] == "R34R"]
                    r34r_occupied = any((x, y) in particles for (x, y) in r34r_cells)  # R34R に飛行機がいるかどうか判定
                    t3f_cells = [(x, y) for y in range(label_grid.shape[0]) for x in range(label_grid.shape[1]) if label_grid[y, x] == "T3F"]
                    t3f_occupied = any((x, y) in particles for (x, y) in t3f_cells)  # T3F に飛行機がいるかどうか判定

                    # R34RS に飛行機がいない、かつ R34R に飛行機がいない、かつ T3F に飛行機がいない、かつ次に R34RS で飛行機が生成されない場合
                    if not r34rs_occupied and not r34r_occupied and not t3f_occupied and not any((x, y) in newly_generated_particles for (x, y) in r34rs_cells):
                        if step - last_r34rs_exit_step > 40:
                            neighbors = [(x, y-1)]  # 上方向に進む
                        else:
                            neighbors = [(x, y)]  # その場で待機
                else:
                    neighbors = [(x, y)]  # その場で待機
                move_counters[particle_id] += 1

            elif current_position == "T2LW":
                if move_counters[particle_id] % 10 == 0:
                    # R34LS、R34L、R34LUL、RUDL に飛行機がいない場合、かつ次に R34RS で飛行機が生成されない場合
                    r34ls_cells = [(x, y) for y in range(14) for x in range(20) if label_grid[y, x] == "R34LS"]
                    r34l_cells = [(x, y) for y in range(14) for x in range(20) if label_grid[y, x] == "R34L"]
                    r34lul_cells = [(x, y) for y in range(14) for x in range(20) if label_grid[y, x] == "R34LUL"]
                    rudl_cells = [(x, y) for y in range(14) for x in range(20) if label_grid[y, x] == "RUDL"]

                    r34ls_occupied = any((x, y) in particles for (x, y) in r34ls_cells)
                    r34l_occupied = any((x, y) in particles for (x, y) in r34l_cells)
                    r34lul_occupied = any((x, y) in particles for (x, y) in r34lul_cells)
                    rudl_occupied = any((x, y) in particles for (x, y) in rudl_cells)

                    # R34LS に飛行機がいない、かつ R34RL に飛行機がいない、かつ L34LUL に飛行機がいない、かつ RUDL に飛行機がいない、かつ次に R34LS で飛行機が生成されない場合
                    if not r34ls_occupied and not r34l_occupied and not r34lul_occupied and not rudl_occupied and not any((x, y) in newly_generated_particles for (x, y) in r34ls_cells):
                        if not is_cell_occupied(x, y-2):  # 前のセルが空いているかチェック
                            neighbors = [(x, y-1)]  # 上方向に進む
                        else:
                            neighbors = [(x, y)]
                    else:
```

```
                    neighbors = [(x, y)]  # その場で待機
            else:
                neighbors = [(x, y)]  # その場で待機
            move_counters[particle_id] += 1

        elif current_position == "T1W":
            if move_counters[particle_id] % 10 == 0:
                # R05,R05F,R05W に飛行機がいない場合
                r05_cells = [(x, y) for y in range(14) for x in range(20) if label_grid[y, x] == "R05"]
                r05f_cells = [(x, y) for y in range(14) for x in range(20) if label_grid[y, x] == "R05F"]
                r05w_cells = [(x, y) for y in range(14) for x in range(20) if label_grid[y, x] == "R05W"]

                r05_occupied = any((x, y) in particles for (x, y) in r05_cells)
                r05f_occupied = any((x, y) in particles for (x, y) in r05f_cells)
                r05w_occupied = any((x, y) in particles for (x, y) in r05w_cells)

                # R05 に飛行機がいない、かつ R05F に飛行機がいない、かつ R05W に飛行機がいない
場合
                if not r05_occupied and not r05f_occupied and not r05w_occupied:
                    neighbors = [(x+1, y)]  # 右方向に進む
                else:
                    neighbors = [(x, y)]  # その場で待機
            else:
                neighbors = [(x, y)]  # その場で待機
            move_counters[particle_id] += 1

        elif current_position == "R05W":
            # R34RS に着陸したばかりの飛行機がいるかどうかを判定
            r34rs_cells = [(x, y) for y in range(label_grid.shape[0]) for x in range(label_grid.shape[1]) if
label_grid[y, x] == "R34RS"]
            r34rs_occupied_by_newly_generated_plane = any((x, y) in newly_generated_particles for (x,
y) in r34rs_cells)

            # R34RS に離陸する飛行機がいるかどうかを判定
            r34rs_occupied_by_departing_plane = any(
                (x, y) in particles and "T3F" in passed_positions.get(index, {})
                for index, (x, y) in enumerate(particles)
                if (x, y) in r34rs_cells  # R34RS のセルのみを対象とする
            )

            # 条件に応じて移動方向を決定
            if r34rs_occupied_by_newly_generated_plane:
                # R34RS に着陸したばかりの飛行機がいる場合、待機
                neighbors = [(x, y)]
            elif step - last_r05w_exit_step < 40:
                # R05W から離陸したばかりの飛行機がいる場合、待機
                neighbors = [(x, y)]
            else:
                if r34rs_occupied_by_departing_plane or not any((x, y) in particles for (x, y) in r34rs_cells):
                    # R34RS に離陸する飛行機がいる場合、上方向に移動
                    neighbors = [(x, y-1)]
                    last_r05w_exit_step = step
                else:
                    # R34RS に飛行機がいない場合、上方向に移動
```

```
                    neighbors = [(x, y-1)]


        # 新しい位置を選択
        if neighbors:
            new_position = random.choice(neighbors)
            if new_position not in new_particles or new_position == (x, y):
                new_particles.append(new_position)
            else:
                new_particles.append((x, y))
        else:
            # neighbors が空の場合、その飛行機は消える（リストから削除）
            continue


# R05F にいる飛行機を消す
new_particles = [(x, y) for x, y in new_particles if label_grid[y, x] != "R05F"]

# パーティクルを更新
particles = new_particles

# 新たに生成された飛行機のリストを更新
newly_generated_particles = [(x, y) for x, y in newly_generated_particles if (x, y) in particles]

# 10%の確率で新しい飛行機を生成 (R34LS)
if step - last_generated_step >= 40:
    # R34LS と R34L、RUDL に飛行機がいない場合にのみ新しい飛行機を生成
    r34ls_cells = [(x, y) for y in range(14) for x in range(20) if label_grid[y, x] == "R34LS"]
    r34l_cells = [(x, y) for y in range(14) for x in range(20) if label_grid[y, x] == "R34L"]
    r34lul_cells = [(x, y) for y in range(14) for x in range(20) if label_grid[y, x] == "R34LUL"]
    rudl_cells = [(x, y) for y in range(14) for x in range(20) if label_grid[y, x] == "RUDL"]

    r34ls_occupied = any((x, y) in particles for (x, y) in r34ls_cells)
    r34l_occupied = any((x, y) in particles for (x, y) in r34l_cells)
    r34lul_occupied = any((x, y) in particles for (x, y) in r34lul_cells)
    rudl_occupied = any((x, y) in particles for (x, y) in rudl_cells)

    if not r34ls_occupied and not r34l_occupied and not r34lul_occupied and not rudl_occupied:
        empty_cells = [(x, y) for (x, y) in r34ls_cells if (x, y) not in particles]
        if empty_cells:
            new_particle = random.choice(empty_cells)
            particles.append(new_particle)
            newly_generated_particles.append(new_particle)  # 新しく生成された飛行機を追跡
            # 飛行機の ID を管理
            if len(plane_ids) <= len(particles) -1:  # 新しい飛行機の場合
                plane_ids.append(len(plane_ids))  # 新しい ID を割り当てる

            last_generated_step = step # 最後に生成されたステップを更新

# 10%の確率で新しい飛行機を生成 (R34RS)
if step - last_generated_step1 >= 75:
    # R34RS と R34R、T3F に飛行機がいない場合にのみ新しい飛行機を生成
    r34rs_cells = [(x, y) for y in range(14) for x in range(20) if label_grid[y, x] == "R34RS"]
```

```
        r34r_cells = [(x, y) for y in range(14) for x in range(20) if label_grid[y, x] == "R34R"]
        t3f_cells = [(x, y) for y in range(14) for x in range(20) if label_grid[y, x] == "T3F"]

        r34rs_occupied = any((x, y) in particles for (x, y) in r34rs_cells)
        r34r_occupied = any((x, y) in particles for (x, y) in r34r_cells)
        t3f_occupied = any((x, y) in particles for (x, y) in t3f_cells)

        if not r34rs_occupied and not r34r_occupied and not t3f_occupied:
            # 新しい飛行機を R34RS に生成
            empty_cells = [(x, y) for (x, y) in r34rs_cells if (x, y) not in particles]
            if empty_cells:
                new_particle = random.choice(empty_cells)  # 空いているセルからランダムに選択して
飛行機を生成
                particles.append(new_particle)  # 飛行機を particles リストに追加
                newly_generated_particles.append(new_particle)  # 新しく生成された飛行機を追跡
                # 飛行機の ID を管理
                if len(plane_ids) <= len(particles) - 1:  # 新しい飛行機の場合
                    plane_ids.append(len(plane_ids))  # 新しい ID を割り当てる

                last_generated_step1 = step  # 最後に生成されたステップを更新

# グリッドに名前を表示
for y in range(label_grid.shape[0]):
    for x in range(label_grid.shape[1]):
        if label_grid[y, x] != " ":
            ax.text(x, y, label_grid[y, x], ha='center', va='center', fontsize=8, color='blue')

# アニメーション関数
def animate(frame):
    points = np.argwhere(particle_history[frame] == 1)
    scatter.set_offsets(points[:, [1, 0]])  # x, y の順番に注意
    # ステップ数を更新
    step_text.set_text(f'Step: {frame + 1}')  # frame は 0 から始まるので+1 する
    return scatter, step_text # step_text も返すように変更

# FuncAnimation を使ってアニメーションを作成
ani = FuncAnimation(fig, animate, frames=len(particle_history), interval=1000, blit=True)

# Google Colab での再生ボタンを表示
HTML(ani.to_jshtml())
```

謝辞

この場をお借りして、本論文の作成にあたりご支援いただいたすべての方々に、心より感謝申し上げます。特に、桂田祐史准教授には貴重なご意見とご指導を賜り、深く感謝いたします。また、本研究に関連するプログラミングや思考の整理にご協力いただいた桂田研究室の皆様にも、厚く御礼申し上げます。

参考文献

[1]国土交通省，羽田空港の国際線増便の実現方策(最終アクセス日 2025-2-18)，
https://www.mlit.go.jp/koku/haneda/archive/faq/pdf/06.pdf

[2]坂下文規,森地茂,日比野直彦，羽田空港における航空遅延および出発時地上走行時間に関する研究, JSCE 公益社団法人　土木学会(最終アクセス日 2025-2-18)，
http://library.jsce.or.jp/jsce/open/00039/200911_no40/pdf/203.pdf

[3]羽田空港の離陸・着陸の概要(初心者向け基本的データ)空港平面図(最終アクセス日 2025-2-18)，
https://fr24.rgr.jp/ki1/

[4]国土交通省，羽田空港のこれから　-飛行経路の見直しによる羽田空港の国際線増便について-(最終アクセス日 2025-2-18)，
https://haneda.meclib.jp/haneda_panel_6_0_1/book/index.html#target/page_no=1

[5]「のびぃ」の隠れ家，羽田空港での離着陸(滑走路), 2017-9-10，最終更新日 2022-2-13，
https://www.mnob62.com/entry/runways-Haneda-airport

[6]YOSHIAKI KAWAGOE, RYOHEI CHINO,SATORI TSUZUKI, ERI ITOH, AND TOMONAGA OKABE Analyzing Stochastic Features in Airport Surface Traffic Flow Using Cellular Automaton: Tokyo International Airport, IEEE Access, vol 10, pp.95344-95355, 2022,
https://ieeexplore.ieee.org/document/9878328