

## 付録

使用ソースコードを以下に記載する．実行環境は以下の通りである．

使用言語:	$C++(C++11)$
実行環境:	Mac OS Sierra (Ver 10.12.6)
コンパイラ:	$g++(GCC) 5.4.0$
コンパイラオプション:	$g++ -Wall -g -ggdb -std=c++11$

Table 1: 実行環境

なお、プロットは python を用いて matplotlib で行ったが、プロットを行っているだけなので、省略する．また関数の中でファイル作成を行っているが、ディレクトリ構成は以下の通りである．

```
sc.controlsystem
├── datafile (Output files produced here)
├── report1
│   ├── Makefile
│   └── report1 (execution file)
├── report2
│   ├── Makefile
│   └── report2 (execution file)
└── src
    ├── Eigen
    ├── unsupported
    ├── kalman_filter.cpp
    ├── kalman_filter.hpp
    ├── non_linear.cpp
    ├── non_linear.hpp
    ├── report1.cpp
    └── report2.cpp
```

Listing 1: report1.cpp (report1 用 main)

```

1  #include "Eigen/Core"
2  #include "Eigen/Geometry"
3  #include "non_linear.hpp"
4  #include <iostream>
5
6  int main() {
7      Eigen::Vector3d I;
8      I.x() = 1.9; // I_x
9      I.y() = 1.6; // I_y
10     I.z() = 2.0; // I_z
11
12     Eigen::Vector3d omega_init; //ノミナル角速度
13     omega_init.x() = 0.1;
14     omega_init.y() = 17 * 0.1047 + 0.1;
15     omega_init.z() = 0;
16
17     Eigen::Vector4d qt_init; //初期 Quaternion
18     qt_init << 1.0, 0, 0, 0;
19
20     // Initialize State
21     Eigen::VectorXd omega_qt;
22     Eigen::Matrix<double, 3, 3> dcm;
23     omega_qt = Eigen::VectorXd::Zero(7);
24
25     omega_qt << qt_init[0], qt_init[1], qt_init[2], qt_init[3], omega_init[0],
26               omega_init[1], omega_init[2];
27
28     double SD_Q = 0.0; //外乱トルクの標準偏差
29     double SD_R = 0.0; //観測ノイズの標準偏差
30     double SEED =
31         1; //外乱の乱数生成メルセンヌツイスタ用シード,0の時シードもランダムに生成される
32
33     NonLinear noli(I, SD_Q, SD_R, SEED);
34
35     //ルンゲクッタ用パラメータ
36     double dt = 0.01;
37     double T_END = 100;
38
39     int counter = 0; // print用カウンター
40
41     for (double t = 0; t < T_END; t += dt) {
42         if (!(counter % 100)) {
43             std::cout << "time:" << t << std::endl;
44         }
45         noli.update_vW(); // make disturbance noise
46         noli.propagation(dt, omega_qt, dcm, t, t + dt);
47         noli.writefile(t, omega_qt, dcm, "R1");
48         counter += 1;
49     }
50 }

```

Listing 2: report2.cpp (report2 用 main)

```

1  #include "Eigen/Core"
2  #include "Eigen/Geometry"
3  #include "iostream"
4  #include "kalman_filter.hpp"
5  #include "non_linear.hpp"
6
7  int main() {
8
9      // setup for system model
10
11     // settings for satelite attitude
12     Eigen::Vector3d I;
13     I.x() = 1.9; // I_x
14     I.y() = 1.6; // I_y
15     I.z() = 2.0; // I_z
16
17     Eigen::Vector3d omega_init; //初期角速度
18     Eigen::Vector3d omega_init_error;
19     omega_init << 0.1, 17 * 0.1047 + 0.1, 0;
20     omega_init_error << 0.01, 0.01, 0.01;
21
22     Eigen::Vector4d qt_init; //初期Quaternion
23     Eigen::Vector4d qt_init_error;
24     qt_init << 1.0, 0, 0, 0;
25     qt_init_error << -0.01, 0.08, 0.08, 0.08;
26
27     // initial state
28     Eigen::VectorXd vX0_True;
29     Eigen::VectorXd vX0_Est;
30     vX0_True = Eigen::VectorXd::Zero(7);
31     vX0_Est = Eigen::VectorXd::Zero(7);
32
33     vX0_True << qt_init[0], qt_init[1], qt_init[2], qt_init[3], omega_init[0],
34                omega_init[1], omega_init[2];
35     omega_init += omega_init_error;
36     qt_init += qt_init_error;
37     vX0_Est << qt_init[0], qt_init[1], qt_init[2], qt_init[3], omega_init[0],
38                omega_init[1], omega_init[2];
39
40     double SD_Q = 0.01; //外乱トルクの標準偏差
41     double SD_R = 0.01; //観測ノイズの標準偏差
42     double SEED =
43         0; //外乱の乱数生成メルセンヌツイスタ用シード,0の時シードもランダムに生成される
44
45     NonLinear noli(I, SD_Q, SD_R, SEED);
46
47     // System Parameters
48     int dimI = 3;
49     int dimO = 3;
50     int dimS = 7;
51
52     // Define Initial Vectors and Matrix
53     Eigen::VectorXd vz;
54     Eigen::MatrixXd Q;
55     Eigen::MatrixXd R;
56     Eigen::MatrixXd P0;
57
58     // init Vectors and Matrix
59     vz = Eigen::VectorXd::Zero(dimO);
60     Q = Eigen::MatrixXd::Identity(dimI, dimI);
61     R = Eigen::MatrixXd::Identity(dimO, dimO);
62     P0 = Eigen::MatrixXd::Identity(dimS, dimS);
63
64     // Q process noise
65     Q(0, 0) = SD_Q * SD_Q;
66     Q(1, 1) = SD_Q * SD_Q;
67     Q(2, 2) = SD_Q * SD_Q;
68     // R measurement noise
69     R(0, 0) = SD_R * SD_R;
70     R(1, 1) = SD_R * SD_R;
71     R(2, 2) = SD_R * SD_R;
72     // P0
73     for (int i = 0; i < dimS; i++) {
74         P0(i, i) = SD_R;
75     }

```

```

76
77 // simulation用パラメータ
78 double dt = 0.01;
79 double T_END = 100;
80 double obs_interval = 1.0;
81
82 // end model setup
83
84 // make model
85 KalmanFilter KFD(dimI, dimO, dimS, vXO_Tru, vXO_Est, Q, R, PO, dt);
86
87 // simulate
88 KFD.simulation(nol, T_END, obs_interval);
89 }

```

Listing 3: non.linear.hpp (衛星シミュレーション用クラス (header))

```

1 // attitude for spinning satellite
2 #ifndef NONLINEAR_H
3 #define NONLINEAR_H
4
5 #include "Eigen/Core"
6 #include "Eigen/Geometry"
7 #include <cstdio>
8 #include <string>
9
10 class NonLinear {
11 public:
12     NonLinear(Eigen::Vector3d &I, double SD_Q, double SD_R, int SEED) {
13         Ix_ = I[0];
14         Iy_ = I[1];
15         Iz_ = I[2];
16         SD_Q_ = SD_Q;
17         SD_R_ = SD_R;
18         SEED_ = SEED;
19     };
20
21     double Ix_, Iy_, Iz_;
22     Eigen::Vector3d vV_; //観測ノイズ
23     Eigen::Vector3d vW_; //外乱ノイズ
24
25     // functions for state propagation
26     void propagation(double dt, Eigen::VectorXd &omega_qt,
27                     Eigen::Matrix<double, 3, 3> &dcm, double t_start,
28                     double T_END);
29     void cal_omega_qt_dot(Eigen::VectorXd &omega_qt,
30                          Eigen::VectorXd &omega_qt_dot);
31     void quartanion_to_dcm(Eigen::VectorXd &omega_qt,
32                           Eigen::Matrix<double, 3, 3> &dcm_);
33
34     // functions for producing/ deleting noise
35     void update_vW(); // disturbance noise
36     void update_vV(); // observation noise
37     void del_noise();
38
39     // writing result in files
40     void writefile(double t, Eigen::VectorXd &omega_qt,
41                   Eigen::Matrix<double, 3, 3> &dcm, std::string fileprefix);
42
43 private:
44     double SD_Q_, SD_R_; /*ノイズ標準偏差 SD_Q_:外乱トルク SD_R_:観測ノイズ*/
45     double SEED_; /*メルセンツウィスタ関数用シード 0-シードも乱数で決定*/
46 };
47
48 #endif

```

Listing 4: non\_linear.cpp (衛星シミュレーション用クラス (source))

```

1  #include "non_linear.hpp"
2  #include <fstream>
3  #include <iomanip>
4  #include <iostream>
5  #include <random>
6  #include <string>
7
8  /*****
9      state propgation function
10     propagation()
11     cal_omega_qt_dot()
12     quartanion_to_dcm()
13 *****/
14
15 /* propagation
16    @Input
17    dt: 時間ステップ
18    omega_qt: 伝搬するstate(角速度 $\omega$ , qt)
19    dcm: qtに対応するdcm
20    t_start: 積分開始時間
21    t_end: 積分終了時間
22 */
23 void NonLinear::propagation(double dt, Eigen::VectorXd &omega_qt,
24                             Eigen::Matrix<double, 3, 3> &dcm, double t_start,
25                             double T_END) {
26     for (double t = t_start; t < T_END; t += dt) { // update
27         Eigen::VectorXd k1, k2, k3, k4;
28         Eigen::VectorXd omega_qt_2, omega_qt_3, omega_qt_4;
29         k1 = Eigen::VectorXd::Zero(7);
30         k2 = Eigen::VectorXd::Zero(7);
31         k3 = Eigen::VectorXd::Zero(7);
32         k4 = Eigen::VectorXd::Zero(7);
33         omega_qt_2 = Eigen::VectorXd::Zero(7);
34         omega_qt_3 = Eigen::VectorXd::Zero(7);
35         omega_qt_4 = Eigen::VectorXd::Zero(7);
36         cal_omega_qt_dot(omega_qt, k1);
37
38         omega_qt_2 = omega_qt + dt / 2 * k1;
39         cal_omega_qt_dot(omega_qt_2, k2);
40
41         omega_qt_3 = omega_qt + dt / 2 * k2;
42         cal_omega_qt_dot(omega_qt_3, k3);
43
44         omega_qt_4 = omega_qt + dt * k3;
45         cal_omega_qt_dot(omega_qt_4, k4);
46
47         omega_qt = omega_qt + dt / 6 * (k1 + 2 * k2 + 2 * k3 + k4);
48         quartanion_to_dcm(omega_qt, dcm);
49     }
50 }
51
52 //  $\omega$ , qtの微分を求める
53 void NonLinear::cal_omega_qt_dot(Eigen::VectorXd &omega_qt,
54                                   Eigen::VectorXd &omega_qt_dot) {
55     double q0 = omega_qt(0, 0);
56     double q1 = omega_qt(1, 0);
57     double q2 = omega_qt(2, 0);
58     double q3 = omega_qt(3, 0);
59     double ox = omega_qt(4, 0);
60     double oy = omega_qt(5, 0);
61     double oz = omega_qt(6, 0);
62
63     omega_qt_dot(0, 0) = 0.5 * (-q1 * ox + -q2 * oy + -q3 * oz);
64     omega_qt_dot(1, 0) = 0.5 * (q0 * ox + -q3 * oy + q2 * oz);
65     omega_qt_dot(2, 0) = 0.5 * (q3 * ox + q0 * oy + -q1 * oz);
66     omega_qt_dot(3, 0) = 0.5 * (-q2 * ox + q1 * oy + q0 * oz);
67     omega_qt_dot(4, 0) = (Iy_ - Iz_) / Ix_ * oy * oz + vW_[0] / Ix_;
68     omega_qt_dot(5, 0) = (Iz_ - Ix_) / Iy_ * oz * ox + vW_[1] / Iy_;
69     omega_qt_dot(6, 0) = (Ix_ - Iy_) / Iz_ * ox * oy + vW_[2] / Iz_;
70 }
71
72 // qt->dcm
73 void NonLinear::quartanion_to_dcm(Eigen::VectorXd &omega_qt,
74                                   Eigen::Matrix<double, 3, 3> &dcm) {
75     double q0 = omega_qt(0, 0);

```

```

76 double q1 = omega_qt(1, 0);
77 double q2 = omega_qt(2, 0);
78 double q3 = omega_qt(3, 0);
79 // first column
80 dcm(0, 0) = q0 * q0 + q1 * q1 - q2 * q2 - q3 * q3;
81 dcm(1, 0) = 2 * (q1 * q3 + q0 * q3);
82 dcm(2, 0) = 2 * (q1 * q3 - q0 * q2);
83 // second column
84 dcm(0, 1) = 2 * (q1 * q2 - q0 * q3);
85 dcm(1, 1) = q0 * q0 - q1 * q1 + q2 * q2 - q3 * q3;
86 dcm(2, 1) = 2 * (q2 * q3 + q0 * q1);
87 // third column
88 dcm(0, 2) = 2 * (q1 * q3 + q0 * q2);
89 dcm(1, 2) = 2 * (q2 * q3 - q0 * q1);
90 dcm(2, 2) = q0 * q0 - q1 * q1 - q2 * q2 + q3 * q3;
91 }
92
93 /*****
94 noise producing function
95 update_vW() : 外乱ノイズ
96 update_vV() : 観測ノイズ
97 del_noise() : ノイズ消去
98 *****/
99
100 // 外乱ノイズ生成関数 seed=0の場合, seedもランダム生成
101 void NonLinear::update_vW() {
102     if (SEED_ == 0) {
103         std::random_device seed_gen;
104         std::mt19937 randMt(seed_gen());
105         static std::normal_distribution<> normW(0.0, SD_Q_);
106         vW_(0) = normW(randMt);
107         vW_(1) = normW(randMt);
108         vW_(2) = normW(randMt);
109     } else {
110         std::mt19937 randMt(SEED_); //メルセンツウィスタ乱数
111         static std::normal_distribution<> normW(0.0, SD_Q_);
112         vW_(0) = normW(randMt);
113         vW_(1) = normW(randMt);
114         vW_(2) = normW(randMt);
115     }
116 }
117
118 // 観測ノイズ生成関数, seed=0の時はseedもランダム生成
119 void NonLinear::update_vV() {
120     if (SEED_ == 0) {
121         std::random_device seed_gen;
122         std::mt19937 randMt(seed_gen());
123         static std::normal_distribution<> normW(0.0, SD_Q_);
124         vV_(0) = normW(randMt);
125         vV_(1) = normW(randMt);
126         vV_(2) = normW(randMt);
127     } else {
128         std::mt19937 randMt(SEED_); //メルセンツウィスタ乱数
129         static std::normal_distribution<> normW(0.0, SD_Q_);
130         vV_(0) = normW(randMt);
131         vV_(1) = normW(randMt);
132         vV_(2) = normW(randMt);
133     }
134 }
135
136 // システムノイズ/観測ノイズ消去
137 void NonLinear::del_noise() {
138     vV_(0) = 0;
139     vV_(1) = 0;
140     vV_(2) = 0;
141     vW_(0) = 0;
142     vW_(1) = 0;
143     vW_(2) = 0;
144 }
145
146 /*****
147 Writing quaternion, omega, dcm data
148 writefile()
149 *****/
150
151 /* @input
152 t: 時間

```

```

153     omega_qt, dcm: state
154     fileprefix: ファイル先頭につける文字
155 */
156
157 void NonLinear::writefile(double t, Eigen::VectorXd &omega_qt,
158                          Eigen::Matrix<double, 3, 3> &dcm,
159                          std::string fileprefix) {
160     // write data of quartanion
161     std::fstream fout_q;
162     std::string filename_q;
163     if (SD_Q_ > 0) {
164         filename_q = "../datafile/" + fileprefix + "qt_error.txt";
165     } else {
166         filename_q = "../datafile/" + fileprefix + "qt.txt";
167     }
168     if (t == 0)
169         fout_q.open(filename_q, std::ios::out);
170     else
171         fout_q.open(filename_q, std::ios::app);
172     fout_q << t << " ";
173     for (int n = 0; n < 4; n++) {
174         fout_q << omega_qt(n, 0) << " ";
175         if (n == 3) {
176             fout_q << std::endl;
177         }
178     }
179     // write data of omega
180     std::fstream fout_o;
181     std::string filename_o;
182     if (SD_Q_ > 0) {
183         filename_o = "../datafile/" + fileprefix + "omega_error.txt";
184     } else {
185         filename_o = "../datafile/" + fileprefix + "omega.txt";
186     }
187     if (t == 0)
188         fout_o.open(filename_o, std::ios::out);
189     else
190         fout_o.open(filename_o, std::ios::app);
191     fout_o << t << " ";
192     for (int m = 0; m < 3; m++) {
193         fout_o << omega_qt(m + 4, 0) << " ";
194         if (m == 2) {
195             fout_o << std::endl;
196         }
197     }
198     // write data for dcm
199     for (int m = 0; m < 3; m++) {
200         std::fstream fout_d;
201         std::string filename_d;
202         if (SD_Q_ > 0) {
203             filename_d = "../datafile/" + fileprefix + "dcm_column" +
204                         std::to_string(m) + "_error" + ".txt";
205         } else {
206             filename_d = "../datafile/" + fileprefix + "dcm_column" +
207                         std::to_string(m) + ".txt";
208         }
209         if (t == 0)
210             fout_d.open(filename_d, std::ios::out);
211         else
212             fout_d.open(filename_d, std::ios::app);
213         fout_d << t << " " << dcm(0, m) << " " << dcm(1, m) << " " << dcm(2, m)
214             << std::endl;
215     }
216 }

```

Listing 5: kalman\_filter.cpp (カルマンフィルタ用クラス (header))

```

1  /*****
2  * Discreted Kalman filter Program
3  *
4  * < Linearized Equation of State >
5  *    $\Delta x' = mA * \Delta x + mB * (u+vW)$ 
6  *    $y = mH * \Delta x + vV$ 
7  *
8  * < Discreted Linearized Equation of State >
9  *    $\Delta x[k]' = mPhi * \Delta x[k-1] + mGmm * vW[k-1]$ 
10 *
11 * < Matrices >
12 *   mA - System dynamics matrix (State Matrix) (SxS)
13 *   mB - input matrix (SxI)
14 *   mH - Output matrix (OxS)
15 *   mQ - Process noise covariance (IxI)
16 *   mR - Measurement noise covariance (OxO)
17 *   mP - Estimate error covariance (SxS)
18 *   mK - Kalman Gain
19 *
20 * < Vectors >
21 *   vSysTru - True State Vector
22 *   vSysEst - State Vector for Estimated Models
23 *   vW - disturbance noise (defined in NonLinear Class)
24 *   vV - Observation noise (defined in NonLinear Class)
25 *
26 * Non-Linear equation of state is written in class NonLinear.
27 * For propagation of state vectors in both True State and
28 * Estimated Model, propagation function in class NonLinear
29 * is used. In addition, system parameters in class NonLinear
30 * might be used to update A,B,H matrices.
31 *
32 *****/
33
34 #ifndef KFD_H
35 #define KFD_H
36 #include "Eigen/Core"
37 #include "Eigen/Geometry"
38 #include "non_linear.hpp"
39 #include <cstdio>
40 #include <iostream>
41
42 class KalmanFilter {
43
44 public:
45     KalmanFilter(int dimI, int dimO, int dimS, const Eigen::VectorXd &vX0_True,
46                 const Eigen::VectorXd &vX0_EST, const Eigen::MatrixXd &Q,
47                 const Eigen::MatrixXd &R, const Eigen::MatrixXd &P0,
48                 const double dt)
49         : dimI(dimI), dimO(dimO), dimS(dimS), vSysTru(vX0_True), vSysEst(vX0_EST),
50           mQ(Q), mR(R), mP(P0), dt(dt) {
51         // init Vectors and Matrix
52         mA = Eigen::MatrixXd::Zero(dimS, dimS);
53         mB = Eigen::MatrixXd::Zero(dimS, dimI);
54         mH = Eigen::MatrixXd::Zero(dimO, dimS);
55         vZ = Eigen::VectorXd::Zero(dimO);
56
57         // init
58         mM = mP;
59         mGmm = Eigen::MatrixXd::Zero(dimS, dimI);
60         mK = Eigen::MatrixXd::Zero(dimS, dimO);
61         // display Initial Matrix
62         std::cout << "A" << std::endl << mA << std::endl;
63         std::cout << "B" << std::endl << mB << std::endl;
64         std::cout << "H" << std::endl << mH << std::endl;
65         std::cout << "Q" << std::endl << mQ << std::endl;
66         std::cout << "R" << std::endl << mR << std::endl;
67         std::cout << "P" << std::endl << mM << std::endl;
68         std::cout << "M" << std::endl << mM << std::endl;
69         std::cout << "K" << std::endl << mK << std::endl;
70         std::cout << "H" << std::endl << mH << std::endl;
71         std::cout << "Phi" << std::endl << mPhi << std::endl;
72         std::cout << "Gmm" << std::endl << mGmm << std::endl;
73     };
74
75     // System Parameters (class SpinSat)

```



```

76 // System Matrices
77 int dimI, dimO, dimS; // Number of Input, Output, State
78 Eigen::MatrixXd mA; // State Matrix
79 Eigen::MatrixXd mB; // Input Matrix
80 Eigen::MatrixXd mH; // Output Matrix
81 // Matrices for Discretized State Equation
82 Eigen::MatrixXd mPhi, mGmm;
83 // Vectors for states (vSys = x)
84 Eigen::VectorXd vSysTru, vSysEst;
85 Eigen::VectorXd vz; // observed Y - estimated Y
86 // Matrices for errors
87 Eigen::MatrixXd mQ, mR;
88 Eigen::MatrixXd mP, mM, mK;
89 // Discrete time step
90 double dt;
91
92 // @ method
93
94 /* Update functions (need to rewrite when system is changed)*/
95 void UpdateMatrixA(NonLinear &nol);
96 void UpdateMatrixB(NonLinear &nol);
97 void UpdateMatrixH(int col);
98 // Calculate Phi, Gmm using Matrix A and Matrix B
99 void UpdateMatrixPhi();
100 void UpdateMatrixGmm();
101 // Calculating State Vector Norms
102 double CalcQuaternionNorm(Eigen::VectorXd &vX);
103 void NormalizeQuaternion(Eigen::VectorXd &vX);
104 double CalcErrNorm(Eigen::VectorXd &vSysTru, Eigen::VectorXd &vSysEst);
105
106 // select column of DCM observed
107 int select_column(int seed);
108
109 /* Whole Update Sequence (need to rewrite when system is changed)*/
110 void simulation(NonLinear &nol, double T_END, double obs_interval);
111
112 // write files
113 void write_file_pe(double t);
114 void write_file_k(double t);
115 };
116
117 #endif

```

Listing 6: kalman\_filter.cpp (カルマンフィルタ用クラス (source))

```

1 #include "Eigen/LU"
2 #include "unsupported/Eigen/MatrixFunctions"
3 #include <fstream>
4 #include <iomanip>
5 #include <iostream>
6 #include <random>
7 #include <stdexcept>
8
9 #include "kalman_filter.hpp"
10
11 /*****
12 Update Functions for A,B,H
13 void UpdateMatrixA(NonLinear &nol)
14 void UpdateMatrixB(NonLinear &nol)
15 void UpdateMatrixH(int col)
16 *****/
17
18 void KalmanFilter::UpdateMatrixA(NonLinear &nol) {
19     double Ix = nol.Ix_;
20     double Iy = nol.Iy_;
21     double Iz = nol.Iz_;
22     double q0 = vSysEst(0);
23     double q1 = vSysEst(1);
24     double q2 = vSysEst(2);
25     double q3 = vSysEst(3);
26     double wx = vSysEst(4);
27     double wy = vSysEst(5);
28     double wz = vSysEst(6);
29     mA(0, 0) = 0.0; mA(0, 1) = -0.5 * wx; mA(0, 2) = -0.5 * wy; mA(0, 3) = -0.5 * wz;
30     mA(1, 0) = +0.5 * wx; mA(1, 1) = 0.0; mA(1, 2) = +0.5 * wz; mA(1, 3) = -0.5 * wy;

```

```

31  mA(2, 0) = +0.5 * wy; mA(2, 1) = -0.5 * wz; mA(2, 2) = 0.0;          mA(2, 3) = +0.5 * wx;
32  mA(3, 0) = +0.5 * wz; mA(3, 1) = +0.5 * wy; mA(3, 2) = -0.5 * wx; mA(3, 3) = 0.0;
33
34  mA(4, 0) = 0.0; mA(4, 1) = 0.0; mA(4, 2) = 0.0; mA(4, 3) = 0.0;
35  mA(5, 0) = 0.0; mA(5, 1) = 0.0; mA(5, 2) = 0.0; mA(5, 3) = 0.0;
36  mA(6, 0) = 0.0; mA(6, 1) = 0.0; mA(6, 2) = 0.0; mA(6, 3) = 0.0;
37
38  mA(0, 4) = -0.5 * q1; mA(0, 5) = -0.5 * q2; mA(0, 6) = -0.5 * q3;
39  mA(1, 4) = +0.5 * q0; mA(1, 5) = -0.5 * q3; mA(1, 6) = +0.5 * q2;
40  mA(2, 4) = +0.5 * q3; mA(2, 5) = +0.5 * q0; mA(2, 6) = -0.5 * q1;
41  mA(3, 4) = -0.5 * q2; mA(3, 5) = +0.5 * q1; mA(3, 6) = +0.5 * q0;
42
43  mA(4, 4) = 0.0; mA(4, 5) = wz * (Iy - Iz) / Ix; mA(4, 6) = wy * (Iy - Iz) / Ix;
44  mA(5, 4) = wz * (Iz - Ix) / Iy; mA(5, 5) = 0.0; mA(5, 6) = wx * (Iz - Ix) / Iy;
45  mA(6, 4) = wy * (Ix - Iy) / Iz; mA(6, 5) = wx * (Ix - Iy) / Iz; mA(6, 6) = 0.0;
46 }
47
48 void KalmanFilter::UpdateMatrixB(NonLinear &nol) {
49     mB(4, 0) = 1.0 / nol.Ix_;
50     mB(5, 1) = 1.0 / nol.Iy_;
51     mB(6, 2) = 1.0 / nol.Iz_;
52 }
53
54 /*
55  @Input: col: column of DCM to observe
56 */
57
58 void KalmanFilter::UpdateMatrixH(int col) {
59     double q0 = vSysEst(0) * 2.0;
60     double q1 = vSysEst(1) * 2.0;
61     double q2 = vSysEst(2) * 2.0;
62     double q3 = vSysEst(3) * 2.0;
63
64     for (int i = 0; i < 3; i++) {
65         mH(i, 4) = 0.0; mH(i, 5) = 0.0; mH(i, 6) = 0.0;
66     }
67     if (col == 0) {
68         mH(0, 0) = +q0; mH(0, 1) = +q1; mH(0, 2) = -q2; mH(0, 3) = -q3;
69         mH(1, 0) = +q3; mH(1, 1) = +q2; mH(1, 2) = +q1; mH(1, 3) = +q0;
70         mH(2, 0) = -q2; mH(2, 1) = +q3; mH(2, 2) = -q0; mH(2, 3) = +q1;
71     } else if (col == 1) {
72         mH(0, 0) = -q3; mH(0, 1) = +q2; mH(0, 2) = +q1; mH(0, 3) = -q0;
73         mH(1, 0) = +q0; mH(1, 1) = -q1; mH(1, 2) = +q2; mH(1, 3) = -q3;
74         mH(2, 0) = +q1; mH(2, 1) = +q0; mH(2, 2) = +q3; mH(2, 3) = +q2;
75     } else if (col == 2) {
76         mH(0, 0) = +q2; mH(0, 1) = +q3; mH(0, 2) = +q0; mH(0, 3) = +q1;
77         mH(1, 0) = -q1; mH(1, 1) = -q0; mH(1, 2) = +q3; mH(1, 3) = +q2;
78         mH(2, 0) = +q0; mH(2, 1) = -q1; mH(2, 2) = -q2; mH(2, 3) = +q3;
79     } else {
80         std::cout << "DCM COL INDEX ERROR" << std::endl;
81     }
82 }
83
84 /******
85  Update Kalman-filter Matrices
86  void UpdateMatrixPhi()
87  void UpdateMatrixGmm()
88  *****/
89
90 void KalmanFilter::UpdateMatrixPhi() {
91     Eigen::MatrixXd mTemp = dt * mA;
92     mPhi = mTemp.exp();
93 }
94
95 void KalmanFilter::UpdateMatrixGmm() {
96     static Eigen::MatrixXd mI = Eigen::MatrixXd::Identity(dimS, dimS);
97     Eigen::MatrixXd mTemp = dt * mA;
98     mGmm = mA.inverse() * (mTemp.exp() - mI) * mB;
99 }
100
101 /******
102  Fuctions for simulation
103  void KalmanFilter::simulation(NonLinear &nol, double T_END,
104                               double obs_interval)
105  double KalmanFilter::CalcQuaternionNorm(Eigen::VectorXd &vX)
106  void KalmanFilter::NormalizeQuaternion(Eigen::VectorXd &vX)
107  double KalmanFilter::CalcErrNorm(Eigen::VectorXd &vSysTru,

```

```

108                                     Eigen::VectorXd &vSysEst)
109     int KalmanFilter::select_column(int seed)
110     *****/
111
112     /*
113     @input
114     nol: NonLinear Class for non-linear model propagation
115     T_END: simulation time
116     obs_interval: observation interval
117     */
118
119     void KalmanFilter::simulation(NonLinear &nol, double T_END,
120                                 double obs_interval) {
121         Eigen::Matrix<double, 3, 3> dcm_Tru;
122         Eigen::Matrix<double, 3, 3> dcm_Est;
123         Eigen::VectorXd vY_Tru; // one column of dcm_Tru + error
124         Eigen::VectorXd vY_Est; // one column of
125
126         vY_Tru = Eigen::VectorXd::Zero(dim0);
127         vY_Est = Eigen::VectorXd::Zero(dim0);
128         double time_obs = 0.0; // counter
129
130         for (double t = 0; t < T_END + dt; t += dt) {
131             nol.writefile(t, vSysTru, dcm_Tru, "R2True");
132             nol.writefile(t, vSysEst, dcm_Est, "R2Est");
133             // 真值計算
134             nol.update_vW(); // make disturbance noise
135             nol.propagation(dt, vSysTru, dcm_Tru, t, t + dt);
136             // 推定値計算
137             nol.del_noise();
138             nol.propagation(dt, vSysEst, dcm_Est, t, t + dt); // estimate with no noise
139
140             // Matrix更新
141             UpdateMatrixA(nol);
142             UpdateMatrixB(nol);
143             UpdateMatrixPhi();
144             UpdateMatrixGmm();
145             mM = mPhi * mP * mPhi.transpose() + mGmm * mQ * mGmm.transpose();
146             mP = mM;
147             if (time_obs >= obs_interval) {
148                 std::cout << "observe time: " << t << std::endl;
149                 std::cout << " vSysTru:" << vSysTru.transpose() << std::endl;
150                 std::cout << " vSysEst:" << vSysEst.transpose() << std::endl;
151                 time_obs = 0.0;
152                 int seed = 0;
153
154                 int dcm_col = select_column(seed); // select column to observe
155                 std::cout << " column observed:" << dcm_col << std::endl;
156                 UpdateMatrixH(dcm_col);
157                 nol.update_vV(); // make observation noise
158                 vY_Tru = dcm_Tru.col(dcm_col) + nol.vV_; // real
159                 nol.del_noise();
160                 vY_Est = dcm_Est.col(dcm_col) + nol.vV_; // estimation
161                 vz = vY_Tru - vY_Est;
162                 std::cout << " vz:" << vz.transpose() << std::endl;
163                 double err = CalcErrNorm(vSysTru, vSysEst);
164                 std::cout << " Error Norm(before):" << err << std::endl;
165
166                 Eigen::MatrixXd mTemp; // temporary defined matrix used for calculation
167                 mTemp = Eigen::MatrixXd::Zero(3, 3);
168                 mTemp = mH * mM * mH.transpose() + mR;
169                 mP = mM - mM * mH.transpose() * mTemp.inverse() * mH * mM;
170                 mK = mP * mH.transpose() * mR.inverse();
171
172                 vSysEst = vSysEst + mK * vz; // update estimation
173                 err = CalcErrNorm(vSysTru, vSysEst);
174                 std::cout << " Error Norm(after):" << err << std::endl;
175                 std::cout << " Kalman Gain norm:" << mK.norm() << std::endl;
176                 for (int i = 0; i < dimS; i++) {
177                     std::cout << " P" << i << ":" << sqrt(mP(i, i)) << " ";
178                     if (i == dimS - 1)
179                         std::cout << std::endl;
180                 }
181                 std::cout << std::endl;
182                 NormalizeQuaternion(vSysEst);
183                 write_file_k(t);
184             }
185         }

```

```

185     write_file_pe(t); // write mP, mK
186     time_obs += dt;
187 }
188 }
189
190 /* @Input
191    vX: System states Vector
192 */
193 double KalmanFilter::CalcQuaternionNorm(Eigen::VectorXd &vX) {
194     double result = 0;
195     for (int i = 0; i < 4; i++) {
196         result += vX(i) * vX(i);
197     }
198     return sqrt(result);
199 }
200
201 /* @Input
202    vX: System states Vector
203 */
204 void KalmanFilter::NormalizeQuaternion(Eigen::VectorXd &vX) {
205     double norm = CalcQuaternionNorm(vX);
206     for (int i = 0; i < 4; i++) {
207         vX(i) = vX(i) / norm;
208     }
209 }
210
211 /* @Input
212    vSysTru: System states Vector (True(propagated with errors))
213    vSysEst: System states Vector (Estimated(propagated without errors))
214 */
215 double KalmanFilter::CalcErrNorm(Eigen::VectorXd &vSysTru,
216                                 Eigen::VectorXd &vSysEst) {
217     Eigen::VectorXd vD = vSysTru - vSysEst;
218     return vD.norm();
219 }
220
221 // select column od DCM (for observation(updating Matrix H))
222 int KalmanFilter::select_column(int seed) {
223     int column = 0;
224     if (seed == 0) {
225         std::random_device seed_gen;
226         std::mt19937 randMt(seed_gen());
227         std::uniform_int_distribution<int> rand_dist(0, 2);
228         column = rand_dist(randMt);
229     } else {
230         std::mt19937 randMt(seed);
231         std::uniform_int_distribution<int> rand_dist(0, 2);
232         column = rand_dist(randMt);
233     }
234     return column;
235 }
236
237 /******
238    Functions for writing result (Matrix P,K)
239    write_file_pe()
240    write_file_k()
241 *****/
242
243 void KalmanFilter::write_file_pe(double t) {
244     // files to write result
245     // Estimate error covariance
246     std::fstream fout_p;
247     std::string filename_p;
248     filename_p = "../datafile/R2p.txt";
249     // errors
250     std::fstream fout_e;
251     std::string filename_e;
252     filename_e = "../datafile/R2e.txt";
253
254     if (t < 2 * dt) {
255         fout_p.open(filename_p, std::ios::out);
256         fout_e.open(filename_e, std::ios::out);
257     } else {
258         fout_p.open(filename_p, std::ios::app);
259         fout_e.open(filename_e, std::ios::app);
260     }
261 }

```

```

262
263     fout_p << t << " ";
264     fout_e << t << " ";
265     for (int i = 0; i < dimS; i++) {
266         fout_p << sqrt(mP(i, i)) << " ";
267         if (i == dimS - 1)
268             fout_p << std::endl;
269     }
270     Eigen::VectorXd vD = vSysTru - vSysEst;
271     for (int i = 0; i < dimS; i++) {
272         fout_e << vD(i) << " ";
273         if (i == dimS - 1)
274             fout_e << std::endl;
275     }
276 }
277
278 void KalmanFilter::write_file_k(double t) {
279     // files to write result
280     // kalman gain
281     std::fstream fout_k;
282     std::string filename_k;
283     filename_k = "../datafile/R2k.txt";
284
285     if (t < 2) {
286         fout_k.open(filename_k, std::ios::out);
287     } else {
288         fout_k.open(filename_k, std::ios::app);
289     }
290
291     double norm = mK.norm();
292     fout_k << t << " " << norm << std::endl;
293 }

```