# Celkové zhrnutie Kalman filtra

V tomto dokumente je uvedená finálna implementácia Kalmana (zdrojové kódy s vysvetlivkami), ktorá bola po sérií doposiaľ vykonaných testov, označená ako najviac vyhovujúca.

Implementácia Kalman filtra je tvorená jednou hlavnou triedou ***KalmanReal***, ktorá obsahuje nenaplnené atribúty pre matice a vektory (Obr. 1).

- *RealVector _x* - predikovaný stav
- *RealMatrix _P* - predikovaná kovariancia stavu
- *RealMatrix_K* – optimálny Kalman zisk
- *RealMatrix _F* - predvolený procesný model (stavová matica prechodu)
- *RealMatrix _Q* - predvolený šum procesu
- *RealMatrix _B* - predvolený model riadenia (mapuje kontrolný vektor na stavový priestor) -> nepoužíva sa
- *RealMatrix _H* - predvolený model pozorovania (mapuje pozorovania do stavového priestoru).
- *RealMatrix _R* - predvolený šum pozorovania

Ďalej obsahuje, okrem getterov a setterov aj hlavné metódy (Obr. 2):

- *predict* - Používa predchádzajúci odhad stavu a poskytnutý model pohybu na vytvorenie odhadu aktuálneho stavu.
- *update* – Aktuálne informácie o meraní (t.j. vektor *_z*, ktorý obsahuje polohu daného objektu (napr. lopty) a rýchlosť, ktorú dosahuje na osi x a y) sa používajú na úpravu odhadov stavu pomocou poskytnutého modelu pozorovania (matica *_H*).
- *getStateV* – Getter pre získanie aktuálneho stavu vektora *_x* vo forme triedy Vector3D.

```
 * Predicted state.
 */
protected RealVector _x;

/**
 * Predicted state covariance.
 */
protected RealMatrix _P;

/**
 * Optimal Kalman gain.
 */
protected RealMatrix _K;

/**
 * Default process model (state transition matrix).
 */
protected RealMatrix _F;

/**
 * Default process noise.
 */
protected RealMatrix _Q;

/**
 * Default control model (maps control vector to state space).
 */
protected RealMatrix _B;

/**
 * Default observation model (maps observations to state space).
 */
protected RealMatrix _H;

/**
 * Default observation noise.
 */
protected RealMatrix _R;
```

Obr. 1: Nenaplnené atribúty pre matice a vektory

```java
/**
 * Uses the previous state estimate and the provided motion model to produce
 * an estimate of the current state.
 * @param F the process model.
 * @param Q the process noise.
 * @param B the control model.
 * @param u the current control input.
 */
public void predict() {
    _x = _F.operate(_x);
    _P = _F.multiply(_P).multiply(_F.transpose()).add(_Q);
}


/**
 * Current measurement information is used to refine the state estimate
 * using the provided observation model.
 * @param H the observation model.
 * @param R the observation noise.
 * @param z the current measurement.
 */
public void update(RealVector z) {


    // Apply the rest of the Kalman update
    RealVector y = z.subtract(_H.operate(_x));
    RealMatrix S = _H.multiply(_P).multiply(_H.transpose()).add(_R);
    RealMatrix invS = new LUDecomposition(S).getSolver().getInverse();
    _K = _P.multiply(_H.transpose()).multiply(invS);

    // Create a non-square identity matrix
    RealMatrix I = MatrixUtils.createRealMatrix(_K.getRowDimension(), _H.getColumnDimension());
    int dim = Math.min(_K.getRowDimension(), _H.getColumnDimension());

    I = I.add(MatrixUtils.createRealIdentityMatrix(dim));
    _x = _x.add(_K.operate(y));
    _P = I.subtract(_K.multiply(_H)).multiply(_P);

}

public Vector3D getStateV(double z) {
    return Vector3D.cartesian(_x.copy().getEntry(0), _x.copy().getEntry(1), z);
}
```

Obr. 2: Metódy predict, update a getStaveV

Pre naplnenie vyššie uvedených matíc a vektorov slúži trieda ***KalmanAdjuster,*** v ktorej dochádza aj k volaniu vyššie uvedených metód *predict*, *update* a *getStaveV* v metóde *adjustBallPosition* po každej správe od servera (Obr. 3).

K naplneniu matíc a vektorov, nachádzajúcih sa v triede KalmanReal slúži metóda *initMatrixes (Obr. 4)* , ktorá sa volá taktiež po každej správe od servera. Matice a vektory sú zadefinované prostredníctvom vytvorenia nového konštruktora KalmanReal. Hodnoty uvedené v maticiach boli testované a momentálne uvedené sú zatiaľ najlepšie. Určite sa s tými hodnotami dá ešte vyhrať a pomocou ďalších testov dosiahnuť ešte lepšie výsledky.

```
public void processNewServerMessage(ParsedData data) {
    initMatrixes();
    this.now = data.SIMULATION_TIME;
    adjustBallPosition(data);
    adjustFixedPointsPosition(data.fixedObjects);

}

private void adjustBallPosition(ParsedData data) {
    if (data.ballRelativePosition == null) {
        return;
    }

    /*if (isObsolete(lastTimeBallSeen) || ballKalman == null) {
        //ballKalman = freshKalman();
        kr = new KalmanReal(x, P, F, Q, H, R);
    }*/

    Vector3D relSpeed = WorldModel.getInstance().getBall().getRelativeSpeed();
    RealVector z = new ArrayRealVector(new double[] { data.ballRelativePosition.getX(),
    data.ballRelativePosition.getY(), relSpeed.getX(), relSpeed.getY() });
    kr.predict();
    kr.update(z);
    data.ballRelativePosition = kr.getStateV(data.ballRelativePosition.getZ());
    lastTimeBallSeen = now;
}
```

Obr. 3: Metóda adjustBallPosition

```
private void initMatrixes() {

    F = new Array2DRowRealMatrix(new double[][] {
                { 1, 0, 0.1, 0 },
                { 0, 1, 0, 0.1 },
                { 0, 0, 1, 0 },
                { 0, 0, 0, 1 }
                });

    //only observe first 2 values - the position coordinates
    H = new Array2DRowRealMatrix(new double[][] {
                { 0.9, 0, 0.1, 0 },
                { 0, 0.9, 0, 0.1 },
                { 0, 0, 0, 0 },
                { 0, 0, 0, 0 }
                });
    Q = new Array2DRowRealMatrix(new double[][] {
                { 0, 0, 0, 0 },
                { 0, 0, 0, 0 },
                { 0, 0, 0.1,0 },
                { 0, 0, 0, 0.1 }
                });


    R = new Array2DRowRealMatrix(new double[][] {
              { 0.0965, 0, 0, 0 },
              { 0, 0.0965, 0, 0 },
              { 0, 0, 0.1480, 0 },
              { 0, 0, 0, 0.1480 }
              });

    P = new Array2DRowRealMatrix(new double[][] {
              { 1, 0, 0.8, 0 },
              { 0, 1, 0, 0.8 },
              { 0, 0, 0, 0 },
              { 0, 0, 0, 0 }
              });

    x = new ArrayRealVector(new double[] { 0, 0, 0, 0 });


    kr = new KalmanReal(x, P, F, Q, H, R);
}
```

Obr. 4: Metóda initMatrixes