Dead Reckoning

Wednesday, April 30, 2025 10:42 AM

This is done every 1ms

```
// Step 0: update B, u
B[0][0] = cosf(ROBOTps.theta)*0.001;
B[1][0] = sinf(ROBOTps.theta)*0.001;
B[2][1] = 0.001;
u[0][0] = 0.5*(LeftVel + RightVel);    // linear velocity of robot
u[1][0] = gyroz*(PI/180.0);    // angular velocity in rad/s (negative for right hand angle)

// Step 1: predict the state and estimate covariance
Matrix3x2_Mult(B, u, Bu);                 // Bu = B*u
Matrix3x1_Add(x_pred, Bu, x_pred, 1.0, 1.0); // x_pred = x_pred(old) + Bu
Matrix3x3_Add(P_pred, Q, P_pred, 1.0, 1.0); // P_pred = P_pred(old) + Q
// Step 2: if there is a new measurement, then update the state
if (1 == newOPTITRACKpose) {
   OptiNumUpdatesProcessed++;  // checking how often OptiTrack is causing a Kalman update
   if ((OptiNumUpdatesProcessed%10)==0) {
      // LED 3
      GpioDataRegs.GPCTOGGLE.bit.GPIO95 = 1;
   }
   newOPTITRACKpose = 0;
   z[0][0] = OPTITRACKps.x;    // take in the Optitrack Motion Capture measurement
   z[1][0] = OPTITRACKps.y;
   // fix for OptiTrack problem at 180 degrees
   if (cosf(ROBOTps.theta) < -0.99) {
      z[2][0] = ROBOTps.theta;
   }
   else {
      z[2][0] = OPTITRACKps.theta;
   }
   // Step 2a: calculate the innovation/measurement residual, ytilde
   Matrix3x1_Add(z, x_pred, ytilde, 1.0, -1.0);    // ytilde = z-x_pred
   // Step 2b: calculate innovation covariance, S
   Matrix3x3_Add(P_pred, R, S, 1.0, 1.0);                       // S = P_pred + R
   // Step 2c: calculate the optimal Kalman gain, K
   Matrix3x3_Invert(S, S_inv);
   Matrix3x3_Mult(P_pred,  S_inv, K);                           // K = P_pred*(S^-1)
   // Step 2d: update the state estimate x_pred = x_pred(old) + K*ytilde
   Matrix3x1_Mult(K, ytilde, temp_3x1);
   Matrix3x1_Add(x_pred, temp_3x1, x_pred, 1.0, 1.0);
   // Step 2e: update the covariance estimate   P_pred = (I-K)*P_pred(old)
   Matrix3x3_Add(eye3, K, temp_3x3, 1.0, -1.0);
   Matrix3x3_Mult(temp_3x3, P_pred, P_pred);
}  // end of correction step
```

This is never executed because no motion capture system.

```
// set ROBOTps to the updated and corrected Kalman values.
ROBOTps.x = x_pred[0][0];
ROBOTps.y = x_pred[1][0];
ROBOTps.theta = x_pred[2][0];
```

This happens every 1ms

$$B = \begin{bmatrix} \cos(ROBOTps.theta)*0.001 & 0 \\ \sin(ROBOTps.theta)*0.001 & 0 \\ 0 & 0.001 \end{bmatrix}$$

$$u = \begin{bmatrix} 0.5*(LeftVel + RightVel) \\ gyroz * \frac{PI}{180.0} \end{bmatrix}$$

Matrix3x2_Mult(B, u, Bu)
Matrix3x1_Add(x_pred, Bu, x_pred, 1.0, 1.0)

Matrix3x1_Add (x_pred, Bu, x_pred, ...)

explaining this code

x_pred is a multidemensional array
float x_pred[3][1]

Can be accessed

x → x_pred[0][0]
y → x_pred[1][0]
theta → x_pred[2][0]

So

$$\begin{bmatrix} x\_pred[0][0] \\ x\_pred[1][0] \\ x\_pred[2][0] \end{bmatrix} = \begin{bmatrix} x\_pred[0][0] \\ x\_pred[1][0] \\ x\_pred[2][0] \end{bmatrix} + B*u$$

x_pred[0][0] = x_pred[0][0] + cos(ROBOTps.theta)*.001*(LeftVel+RightVel)/2.0
x_pred[1][0] = x_pred[1][0] + sin(ROBOTps.theta)*.001*(LeftVel+RightVel)/2.0
x_pred[2][0] = x_pred[2][0] + 0.001*gyroz*PI/180.0

So when you need to update the x, y, theta of the robot change x_pred

So

x_pred[0][0] = new x value
x_pred[1][0] = new y value
x_pred[2][0] = new theta value

Do this only when a relocalization event has occurred