ENPM – 667

CONTROL OF ROBOTICS SYSTEMS

PROJECT-1 REPORT

Submitted by: **Lavanya Suresh Kannan**

Under the guidance of **Professor Dr. Waseem Malik**

Journal paper approved by TA – **Reza Hadadi**

TABLE OF CONTENTS

1. Enhanced Bioinspired Backstepping Control for a Mobile Robot with Unscented Kalman Filter

Original authors: **ZHE XU, SIMON X. YANG, AND S. ANDREW GADSDEN**

2. Abstract:

Designing controllers for mobile robot has always been a biggest challenge in the robotics industry. The main problem that is focused on the paper is large velocity error changes that occurs in the dynamic system under noisy environment. The author provided a unique approach to solve this issue. The model that integrates two controllers which are bio-inspired backstepping controller and torque controllers with Kalman filter and unscented Kalman filter that reduces the large velocity jumps and provides smooth velocity commands. The author also used Lynapunov method for stability analysis. This novel strategy created by the author provides the results of accurate state estimation, reduced velocity jumps. This method can be implemented under the hard system conditions and noisy environment.

Keywords: Controllers, stability, dynamics, Lyapunov, unscented Kalman filter, Kalman filter, backstepping control, torque control.

3. Introduction:

A system is said to be non-holonomic when the derivatives such as velocity (magnitude and direction) and other derivatives of the positions are constrained. The mobile robots are non-holonomic as they are subjected to the non- integrable equality non-holonomic constraints(velocity). Creating a desired trajectory for such mobile robots under the complex environment with lots of noises in the real world is always challenging. The accuracy of such trajectory can be improved by implementing a better tracking control method.

The author also discussed about multiple tracking control methods. Table 1 discussed about the types of methods along with its pros and cons. Out of these methods the author was inspired by the neural systems and introduced this novel backstepping control method. This method is simple, effective and efficient to implement.

One of the efficient and optimized state estimation technique for linear systems is by using Kalman filter. Kalman filter is a widely used recursive technique that takes the dynamic system inputs and sensor values and gives optimized state estimation values. Later multivariant Kalman filter has been developed for the non-linear systems which are unscented Kalman filter and extended Kalman filter.

The aim of the novel proposal is to allow the mobile robot to operate smoothly in the noisy environment with smooth velocity and more accurate state estimates. The main advantage of this method is,

- Operate under complex environment.
- The systems and noises are taken into consideration while designing the system which provides accurate state estimates.
- Eliminates high velocity jumps.

The report is organized as follows, the next section (section 4) discusses the background that includes the explanations and proofs of dynamics, kinematics models, Kalman filter and unscented Kalman filter for the mobile robots etc. Followed by the next section which explains about the controllers and the mathematical calculation involved. The final section contains the simulation and the results that was made to prove the authors results.

No	Tracking method	Pros	Cons
1.	Linearization	Uses state feedback linearization through decoupling matrix	Large velocity changes in initial stages
2.	Sliding mode	Capable of dealing large initial tracking error.	Suffers from unknown chattering issue
3.	Backstepping	Simple method.	Deals with large initial error
4.	Neural networks	This approach is a pretty good method to deal with large velocity jumps.	It requires online learning which are complex and expensive to implement.
5.	Fuzzy systems	Deals with high velocity jumps	Difficult in setting up the rules.

Table 1

4. **Background:** This section contains five sub sections, A. Kinematics and dynamic model of the mobile robot, B. Error Dynamics, C. Kalman filter algorithm derivation and proof, D. Unscented Kalman filter derivation and proof, E. KF and UKF model for the mobile robot with proofs.

⇨ **Kinematics and dynamic models**:

The non-holonomic robot has two rear wheels that are differential driving wheels and a front wheel. The model of the non-holonomic mobile robot is shown on figure 1. The model is demonstrated in the two-dimensional (2-D) plane in the global cartesian coordinate system. Also, the bot possesses three

dimensional (3D) for its positioning. The posture of the current frame is given by

$$P_C = \begin{pmatrix} X_c \\ Y_{C,} \\ Z_C \end{pmatrix}$$

Similarly, the robotic movement is controlled by two velocities which are linear velocity (for straight line) and angular velocity (rotation, spins etc.). Since the velocities are the inputs, it is given by,

$$U = \begin{bmatrix} v \\ \omega \end{bmatrix}$$

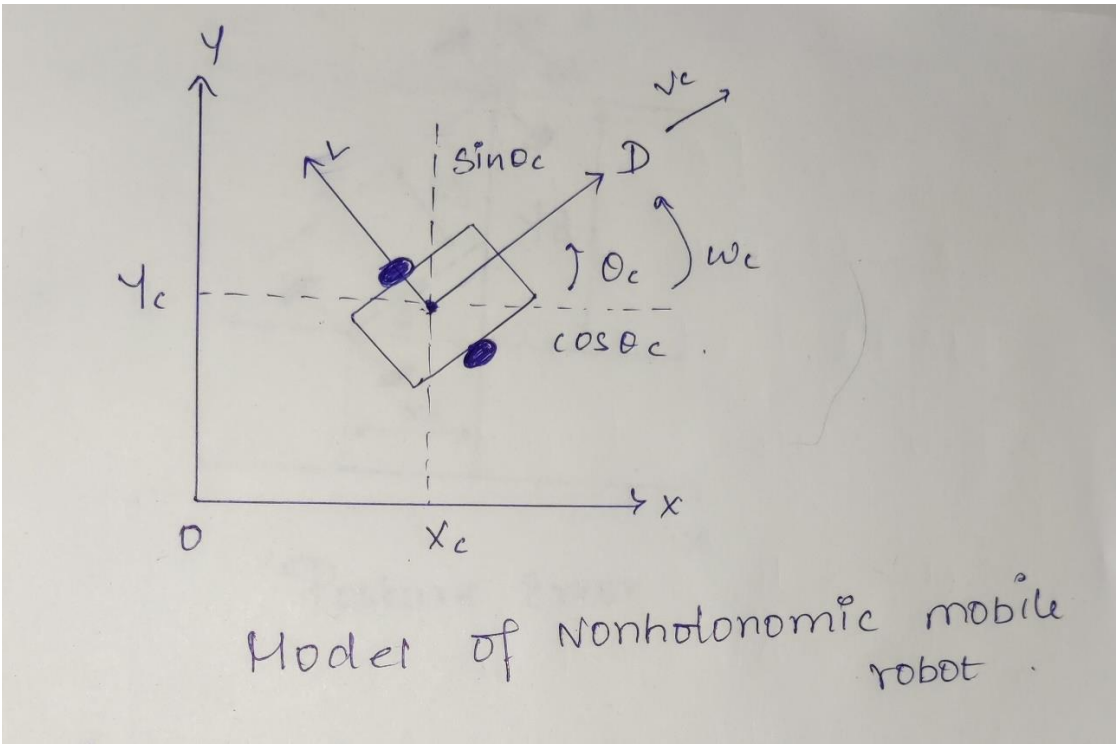


Figure 1

The kinematic constraint of the mobile robot is given as

$$\dot{Y}c \cos\theta_c - \dot{X}c \sin\theta c = 0 \qquad (1)$$

⇨ $\dot{Y}_c \cos\theta_c = \dot{X}_C \sin\theta_c$

⇨ $\frac{\dot{Y}_c}{\dot{X}_C} = \tan\theta_c$ => which makes sense as the robot moves normal to V_c

The kinematics model of the non-holonomic robot that is obtained through Jacobian matrix is,

$$\dot{P}_c = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta_c & 0 \\ \sin\theta_c & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_c \\ \omega_c \end{bmatrix} \quad (2)$$

Ignoring the effect of gravity, the derivation of equation of motion of the non-holonomic robot is,

$$\bar{M}(P_c)\dot{V} + \overline{F}(P_c, \dot{P}_c)V + \tau_d = \bar{B}\tau_c \quad (3)$$

- $\bar{M}(P_c)$ is the matrix for inertia. $\bar{M} = \begin{bmatrix} m & 0 \\ 0 & I \end{bmatrix}$, m = mass of the mobile robot and I is the Inertia.
- $\bar{F}$ is the centripetal and Coriolis matrix for inertia. The matrix value of F is given by $\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$
- τ_d are the disturbances.
- $\bar{B}$ – The matrix value is given by $\frac{1}{r}\begin{bmatrix} 1 & 1 \\ l & l \end{bmatrix}$. r denotes the radius of the driving wheels and l denotes the azimuth length from point C to the driving wheels.
- τ_c is the torque on wheels. $\tau_c = \begin{bmatrix} \tau_R \\ \tau_L \end{bmatrix}$ where, the τ_R is the right wheel torque input and τ_L is the left wheel torque input. (4)

Now, substituting the respective values in equation (3), we get,

$$\bar{M}\dot{V} + 0 + \tau_d = \bar{B}\tau_c$$

Ignoring the disturbances (τ_d), we get the simplifies dynamic equation as follows,

$$\dot{V} = \vec{M}^{-1}B\tau_c \quad (5)$$

⇨ **Error Dynamics:**

Now, the posture of the desired matrix is given by $P_d = \begin{bmatrix} x_d \\ y_d \\ \theta_d \end{bmatrix}$ and the tracking error is defined as $e = \begin{bmatrix} e_D \\ e_L \\ e_\theta \end{bmatrix}$, where e_D is the error is the driving direction, e_L is the error is the lateral direction and e_θ is the error is the orientation of the robot. The tracking error calculated in global and local coordinate system from a inertial to the fixed frame through transformation matrix Te is given by,

$$e_p = \begin{bmatrix} e_D \\ e_L \\ e_\theta \end{bmatrix} = \begin{bmatrix} \cos\theta_C & \sin\theta_c & 0 \\ -\sin\theta_c & \cos\theta_C & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} \tag{6}$$

- e_x is the error in x direction and defined as $x_d - x_c$(desired x – current x)
- e_y is the error in y direction and defined as $y_d - y_c$(desired y – current y)
- e_θ is the orientation and defined as $\theta_d - \theta_c$ (desired θ – current θ)

Taking time derivative on equation (6),

$$\begin{bmatrix} e_D \\ e_L \\ e_\theta \end{bmatrix} = \begin{bmatrix} \cos\theta_C \ e_x + \sin\theta_c \ e_y + 0 \\ -\sin\theta_c \ e_x + \cos\theta_C \ e_y \\ e_\theta \end{bmatrix} \tag{6.1}$$

$$\begin{bmatrix} \dot{e}_D \\ \dot{e}_L \\ \dot{e}_\theta \end{bmatrix} = \begin{bmatrix} \frac{d}{dt}(\cos\theta_C \ e_x + \sin\theta_c \ e_y) \\ \frac{d}{dt}(-\sin\theta_c \ e_x + \cos\theta_C \ e_y) \\ \frac{d}{dt}(e_\theta) \end{bmatrix} \tag{6.2}$$

$$\dot{e}_D = -\sin\theta_c \cdot e_x + \cos\theta_c \cdot \dot{e}_x + \cos\theta_c \cdot \theta_c \cdot e_y + \sin\theta_c \cdot e_y \tag{6.3}$$

⇨ $\left(-e_x \sin\theta_c + e_y \cos\theta_c\right)\omega_c + \dot{e}_x \cos\theta_c + \dot{e}_y \sin\theta_c$ (6.4)

⇨ $e_L = -\sin\theta_x \cdot e_x + \cos\theta_c \cdot e_y$

⇨ Therefore, substituting the value in equation (6.4), we get,

$$e_L\omega_c + (\dot{e}_x \cos\theta_c + \dot{e}_y \sin\theta_c) \quad (6.5)$$

⇨ Now, $e_x = x_d - x_c$, $e_y = y_d - y_c$

⇨ $\dot{e}_x = \frac{d}{dt}(x_d - x_C) = -\dot{x}_C$

⇨ $\dot{e}_y = \frac{d}{dt}(y_d - y_c) = -\dot{y}_c$

⇨ Therefore, equation (6.5) becomes,

$$e_L\omega_c - (\dot{x}_c \cdot \cos\theta_c + \dot{y}_c \cdot \sin\theta_c) \quad (6.6)$$

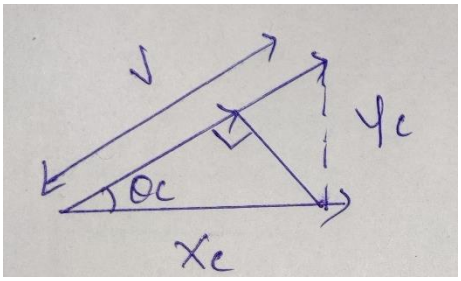


Figure 2

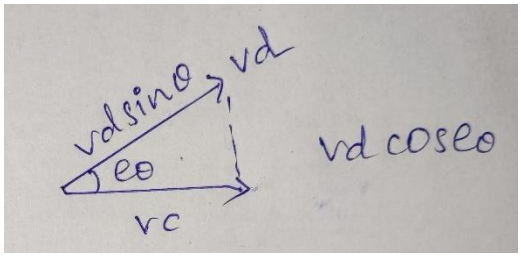


Figure 3

⇨ From figure 2 and figure 3 we get,

⇨ $\dot{e}_D = e_L\omega_c - V_c + V_d \cos e_\theta$

Similarly, for $\dot{e}_L$ we get,

⇨ $\dot{e}_L = -\sin\theta_c \cdot \dot{\theta}_c \cdot e_y - \cos\theta_c \cdot \dot{\theta}_c e_x + \cos\theta_c \cdot \dot{e}_{\dot{y}} - \sin\theta_c \cdot \dot{e}_x$

⇨ $(-e_y \sin\theta_c - e_x \cos\theta_c)\dot{\theta}_c + \dot{e}_y \cos\theta_c - \dot{e}_x \sin\theta_c$

⇨ $-e_D\omega_c + (-\dot{x}_c \cos\theta_c + \dot{y}_C \sin\theta_c)$ (6.7)

Finally, for $\dot{e}_\theta$, we get,

$$\omega_d - \omega_c \quad (6.8)$$

Now, the matrix (6.2) becomes

$$\begin{bmatrix} \dot{e}_D \\ \dot{e}_L \\ \dot{e}_\theta \end{bmatrix} = \begin{bmatrix} e_L\omega_c - V_c + V_d \cos e_\theta \\ -e_D\omega_c + (-\dot{x}_c \cos\theta_c + \dot{y}_C \sin\theta_c) \\ \omega_d - \omega_c \end{bmatrix} \quad (7)$$

To find the linear and angular velocities we need to differentiate angular velocity with respect to time. This is because a very small movement is involved. So ω_d at that instant is given as

$$\omega_d = \frac{\vec{r} \times \vec{v}}{|v|^2}$$

⇨ $\frac{\frac{d\vec{r}}{dt} \times \frac{\overrightarrow{dv}}{dt}}{\left|\frac{\overrightarrow{dr}}{at}\right|^2}$

⇨ $\frac{\overrightarrow{v_d} \times \vec{a}_d}{|v_d|^2}$

⇨ We know that $v_d = \sqrt{x_d^2 + y_d^2}$ (8)

$$\frac{\dot{x}_d\ddot{y}_d - \ddot{x}_d y_d}{\dot{x}_d^2 + y_d^2} \quad (9)$$

⇨ **Kalman filter algorithm derivation and proof:**

One of the most common and famous state estimation technique for linear systems is by using Kalman filter. This was first introduced by Rudolf E. Kalman in the year 1960. The main goal of the Kalman filter is to take probabilistic estimate of the state and update it in real time using two steps:

- Prediction
- Correction

We need to create two models. One is motion model which we predict based on our observation and the other one is measurement model which is based on the sensor values.

Prediction – Motion model:

The linear system with noises is given by,

$$X_t = F_t X_{t-1} + B_t U_t + \varepsilon_t \quad (10)$$

X_t => Next state

X_{t-1} =>Previous state

F_t => System matrix

B_t=> Input matrix (which influences)

U_t=> Input

ε_t => error

Prediction – Measurement model:

In this step we are usually creating a model based on the sensor values or measurement values.

$$z_t = H_t X_t + V_t \quad (11)$$

z_t=> Measurement value

H_t=> Measurement matrix

V_t=> Gaussian noise error

Kalman filter procedure- correction model:

- The posterior state estimate is given by,

$$\hat{X}_t = F_t X_{t-1} + B_t U_t \quad (12)$$

$\hat{X}_t$ => posterior state

X_{t-1}=> previous state

- The covariance is given by

$$P_t = F_t \cdot P_{t-1} \cdot F_t^T + Q_t \quad (13)$$

$$(\text{cov}(x) = \Sigma, \text{cov}(Ax) = A\Sigma A^T)$$

P_t=> covariance matrix

P_{t-1}=> new covariance

Q_t=> additional uncertainty

- We want to minimize the error $e_t = \|x_t - \bar{x}\|^2$ (13.1)
- The value of $\bar{X}$ is given by $\bar{X} = \hat{x}_t + K_t(z_t - \quad H_t \quad \hat{x}_t)$ (14)

Note: Equation (14) is the proof of equation (15) in the paper.

- Now, the covariance error matrix is given by $\bar{P}_t$,

$$\bar{P}_t = \text{cov}(X_t - \bar{X}) \quad (14.1)$$

$$\text{i.e., } \bar{P}_t = (X_t - \bar{X})\Sigma(X_t - \bar{X})^T$$

⇨ Substituting equation 14 in equation (14.1), we get

$$\text{cov}\left(X_t - \hat{X}_t - K_t\left(z_t - H_t\hat{X}_t\right)\right)$$

⇨ $\text{cov}\left(X_t - \hat{X}_t - K_t z_t + K_t H_t \hat{X}_t\right)$

⇨ $\text{cov}\left(X_t - (1 - k_t H_t)\hat{X}_t - k_t z_t\right)$ (14.2)

⇨ Substituting equation 11 in equation (14.2), we get

$$\text{cov}\left(X_t - (1 - k_t H_t)\hat{X}_t - k_t(H_t X_t + V_t)\right)$$

⇨ $\text{cov}\left(x_t - (1 - K_t H_t)\hat{X}_t - K_t H_t X_t - K_t V_t\right)$

⇨ $\text{cov}\left((1 - k_t H_t)X_t - (1 - k_t H_t)\hat{X}_t - k_t V_t\right)$

⇨ $\text{cov}\left((1 - K_t H_t)\left(X_t - \hat{X}_t\right) - K_t V_t\right)$

⇨ $(1 - K_t H_t)\,\text{cov}\left(X_t - \hat{X}_t\right)(1 - K_t H_t)^T + K_t \,\text{cov}\, V_t (K_t^T)$

⇨ $\bar{P}_t = (1 - k_t H_t)\hat{P}_t (1 - k_t H_t)^T + K_t R K_t^T$ (15)

Note: Equation (15) is the proof of equation (16) in the paper.

⇨ Let the error function be J = $\Sigma|(X_t - \bar{X})^2|$

⇨ $\Sigma|(X_t - \bar{X})^T (X_t - \bar{X})|$

⇨ $\Sigma|\text{tr}(X_t - \bar{X})(x_t - \bar{x})^T|$ $\quad where\ X^T X = Tr(XX^T)$

⇨ $\text{tr}\,\Sigma|(X_t - \bar{X})(X_t - \bar{X})^T|$

⇨ $J = \text{tr}\,\bar{P}_t$ $\quad i.e.,\ \text{cov}(X_t - \bar{X})$

⇨ We want to find Kalman gain (Kt) such that J is minimized

i.e., $\frac{\partial J}{\partial K_t} = \frac{\partial}{\partial K_t}(\text{tr}\,\bar{P}_t)$

⇨ From equation (15) we get,

$$\frac{\partial}{\partial k_t}\text{tr}\left((1 - k_t H_t)\hat{P}_t (1 - K_t H_t)^T + \frac{\partial}{\partial k_t}(k_t R k_t^T)\right.$$

⇨ Using the property $\frac{\partial}{\partial A}\text{tr}(ABA^T) = 2AB$. If B is symmetric, we can write as

$$\frac{\partial J}{\partial k_t} = 2(1 - k_t H_t)\hat{P}_t(-H_t^{\top}) + 2K_t R$$

⇨ Setting $\frac{\partial J}{\partial k_t} = 0$, we get

$$2(1 - K_t H_t)\hat{P}_t(-H_t^T) = -2K_t R$$

⇨ $\hat{P}_t H_t^T - k_t H_t \hat{P}_t H_t^T = K_t R$

⇨ $\hat{P}_t H_t^T = k_t\left[H_t \hat{P}_t H_t^T + R\right]$

⇨ $K_t = \hat{P}_t H_t^T\left[H_t \hat{P}_t H_t^T + R\right]^{-1}$ (16)

Note: Equation 16 is the proof of equation 14 from the paper.

⇨ **Unscented Kalman filter algorithm and proof:**

In real world application, the system is always non-linear. So, we need a better state estimation technique with higher accuracy. Here author used one of the most famous technique, that is Unscented Kalman filter. UKF uses the unscented transform in the prediction and correction steps. The basic idea of UKF includes three steps:

- Choose sigma points from our input values. These points are not random but deterministic samples chosen to be certain number of standard deviations away from the mean. Due to this reason, they are called sigma appoints and sometimes UKF are called the sigma point transform.
- Now, we have to pass these sigma points through our nonlinear function producing a new set of sigma points belonging to the output distribution.
- Finally, we can compute the sample mean and covariance of the output sigma points with sone chosen weights which will us the approximation of the mean and covariance of the true output distribution.

The general non-linear system function for motion and the measurement model is given as,

$$X_t = f(X_{t-1}, U_t) + \delta_t \qquad (17)$$

$f(X_{t-1}, U_t)$ => Non-linear process of system

δ_t => Gaussian noise

$$Z_t = H(X_t) + \gamma_t \tag{18}$$

$H(X_t)$ => Measurement model

γ_t => Measurement noise

For N-dimensional probability distribution function we need 2N+1 sigma points. Here N is the dimension of state variable X_t.

PREDICTION:

Step 1: Let X_t be the set of 2N+1 sigma points.

$$X_t = \{(X_t, w^i)|i = 0 \dots 2n|\}$$

Let's say that the initial point be X_t^0 which has a mean and denoted as $\hat{X}_t$

$$X_t^0 = \hat{X}_t \tag{19}$$

The corresponding weight for the initial sigma point is given as,

$$W_0 = \frac{\lambda}{n+\lambda} \tag{20}$$

Here λ is the tunning parameter that controls the sigma point spread. This is significantly set lesser than 1. According to the Cholesky decomposition ($NN^T = \Sigma_{xx}$) of the covariance matrix ($P_t = E[(X_0 - \mu_0)(X_0 - \mu_0)^T]$), the value of sigma vectors $X_{i,t}$ which is the second (n+1) sigma points with its corresponding weights W_i is given by,

$$X_{i,t} = \hat{X}_t + \left(\sqrt{(n+\lambda)P_t}\right)_i \quad i = 1, \dots \dots \dots \dots n \tag{21}$$

$$W_i = \frac{\lambda}{[2(n+\lambda)]} \tag{22}$$

The final n sigma points are defined as,

$$X_{i+n,t} = \hat{X}_t - \left(\sqrt{(n-\lambda)P_t}\right)_i \quad i = n+1, \ldots\ldots\ldots 2\,n \tag{23}$$

$$W_{i+n} = \frac{\lambda}{[2(n+\lambda)]} \tag{24}$$

Step 2: Propagating the sigma points into non-linear function.

$$\hat{X}_{i,t} = f\left(\hat{X}_{i,t-1}, U_t\right) \tag{25}$$

$$\hat{Z}_{i,t} = h_t\left(\hat{X}_{i,t}\,, U_t\,\right) \tag{26}$$

$$\hat{z}_t = \sum_{i=0}^{2n} w_i \hat{Z}_{i,t} \tag{27}$$

$\hat{Z}_{i,t}$ => i-th measurement

$\hat{z}_t$ => predicted measurement

$Note$: Equation (26) and (27) are the proof of equation (28) and (29) in the paper.

Step 3: Transformed points are used to compute mean and covariance. The mean and the state error covariance is defined as,

$$\hat{X}_t = \sum_{i=0}^{2n} W_i\ \hat{\chi}_{i,t} \tag{28}$$

$$W_i = \begin{cases} \frac{\lambda}{n+\lambda} & \text{i = 0} \\ \frac{1}{2}\frac{1}{n+\lambda} \end{cases}$$

$\hat{\chi}_{i,t}$ => sigma points

W_i => weights

$\hat{X}_t$ => mean

$$P_t = \sum_{i=0}^{2n} w_i\left(\hat{X}_{i,t} - \hat{X}_t\right)\left(\hat{X}_{i,t} - \hat{X}_t\right)^T + Q_k \tag{29}$$

$P_t =>$ error covariance

$Q_k =>$ additive process noise

CORRECTION:

To correct the state estimate using measurements at time t, we use the non-linear measurement model and sigma points from the prediction step to predict the measurements.

Step 1: Predict measurements from propagated sigma points.

We already have the predicted sigma points from equation (26). i.e.,

$$\hat{Z}_{i,t} = h_t\left(\hat{X}_{i,t}, 0\right) \qquad \text{i} = 0\ldots..2\text{N}$$

Step 2 : Estimate the mean and covariance of predicted measurements:

From equation (26) we already know that the mean value is defined as,

$$\hat{z}_t = \sum_{i=0}^{2n} w_i \hat{Z}_{i,t}$$

$$P_y = \sum_{i=0}^{2n} W_i\left(\hat{Z}_{i,t} - \hat{Z}_t\right)\left(\hat{Z}_{i,t} - \hat{Z}_t\right)^T + R_t \qquad (30)$$

$R_t =>$ additive measurement noise

Step 3: Computing cross-covariance and Kalman gain:

From the above equations we can compute the cross -covariance by,

$$P_{xy} = \sum_{i=0}^{2n} W_i\left(\hat{X}_{i,t} - \hat{X}_t\right)\left(\hat{Z}_{i,t} - \hat{Z}_t\right)^T \qquad (31)$$

The Kalman gain is given by

$$k_t = P_{xy}\, P_y^{-1} \qquad (32)$$

Step 4: computing corrected mean and covariance

The corrected mean and covariance are given by $\hat{X}_{tt},\ \hat{P}_{tt}$ respectively.

$$\hat{X}_{tt} = \hat{X}_t + k_t\left(Z_t - \hat{Z}_t\right) \quad (33)$$

$$\hat{P}_{tt} = \hat{P}_t - K_t P_y K_t^T \quad (34)$$

Thus, UKF provides accurate state estimation for the non-linear system. The author chooses to implement UKF instead of EKF is because UKF does not require linearization compared to EKF.

- ⇨ **The UKF and KF model for the mobile robot:**

  From equation (3-5), we applied Euler's approximation to those equations. Euler's approximation is another iterative method that we use to solve the first order differential equation ($x_i = x_i + \Delta x$). We are applying Euler's rule at time Δt. We get,

$$\dot{\mathrm{V}} = \vec{M}^{-1} B\tau_c\ \tau_c \cdot \Delta t$$

  The KF state model for the dynamics of the robot is given by

$$\hat{V}_t^- = \begin{bmatrix}\hat{v}_{c,t}\\ \hat{\omega}_{c,t}\end{bmatrix} = \begin{bmatrix}\hat{v}_{c,t-1}\\ \hat{\omega}_{c,t-1}\end{bmatrix} + \vec{M}^{-1} B\tau_c\ \tau_c \cdot \Delta t + \alpha_t \quad (35)$$

  $\hat{v}_{c,t}$ => Estimated Linear velocity at time t

  $\hat{\omega}_{c,t}$ => Estimated angular velocity at time t

  $\hat{v}_{c,t-1}$ => Linear velocity at previous state at time t-1

  $\hat{\omega}_{c,t-1}$ => Angular velocity at previous state at time t-1

  α_t => System noises

  The measurement model for the dynamics of the mobile robot is given as,

$$\tilde{V}_t = H\left(\hat{V}_t^-, \beta_t\right)$$

$$\tilde{V}_t = \mathrm{H}\begin{bmatrix}\hat{v}_{c,t}\\ \hat{\omega}_{c,t}\end{bmatrix} + \beta_t \quad (36)$$

$\tilde{V}_t$ => Measured velocity vector

β_t => Measured noise

The UKF state model for the mobile robot is given as,

$$\hat{P}_{c,t}^- = \begin{bmatrix}\hat{x}_{c,t}^-\\ \hat{y}_{c,t}^-\\ \hat{\theta}_{c,t}^-\end{bmatrix} = \begin{bmatrix}\hat{x}_{c,t-1} + \cos\hat{\theta}_{c,t-1}\,\hat{v}_{c,t-1}\cdot\Delta t\\ \hat{y}_{c,t-1} + \sin\hat{\theta}_{c,t-1}\,\hat{v}_{c,t-1}\cdot\Delta t\\ \hat{\theta}_{c,t-1} + \hat{\omega}_{c,t\cdot\Delta t}\end{bmatrix} + \delta_t \quad (37)$$

$\hat{x}_{c,t}^-, \hat{y}_{c,t}^-, \hat{\theta}_{c,t}^-$ => positions of the robots at estimated time t

$\hat{x}_{c,t-1}, \hat{y}_{c,t-1}, \hat{\theta}_{c,t-1}$ => Positions of the robots at the previous time t-1

The UKF measurement model for the mobile robot is given as,

$$\tilde{P}_{c,t}^- = h\left(\hat{P}_{c,t}^-, \gamma_t\right) = h\begin{bmatrix}\hat{x}_{c,t}^-\\ \hat{y}_{c,t}^-\\ \hat{\theta}_{c,t}^-\end{bmatrix} + \gamma_t \quad (38)$$

Here, the author considered 7 sigma points for the UKF model. Also, the author considered the system, and the measurement model noises to be zero mean gaussian noises.

5. **<u>Design of controllers</u>**

   The author used two types of controllers here. One is kinematic and the other one is dynamic controller. These controllers integrate KF and UKF system models and solves the velocity jump issues. As mentioned in figure 4, the block diagram of the tracking system, the author used two closed

feedbacks. We assume that the motion planner plans the path and gives us appropriate set points pd in the world frame. The Te transformation matrix transforms the error in the world frame to the robot frame. Then the inputs are again fed to the controllers and the dynamics and kinematics feedbacks a repropagated through KF and UKF to generate accurate state estimates for the accurate and better performances under hard noisy conditions.

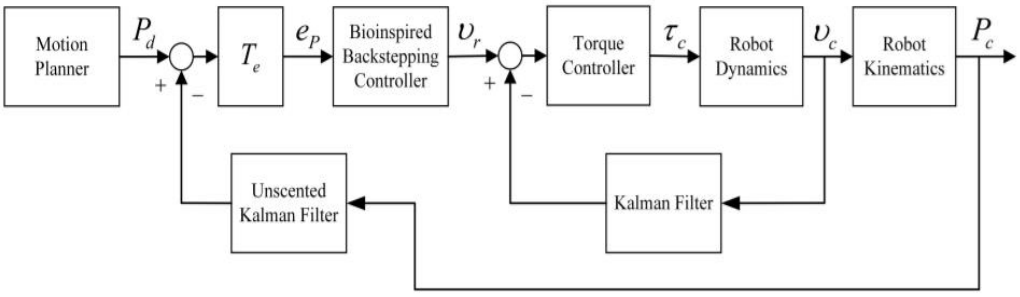


FIGURE 4

A. BIOINSPIRED STEPPING CONTROLLER:

From the reference paper (4), the conventional backstepping control law for the non-holonomic mobile robot is defined as,

$$v_r = C_1 e_D + v_d \cos e_\theta \quad (39)$$

$$\omega_r = \omega_d + C_2 v_d e_L + C_3 v_d \sin e_\theta \quad (40)$$

C1, C2, C3 => designed parameters.

v_r => Linear velocity

ω_r => Angular velocity

The stepping control that used the dynamics of voltage across the membrane, that is called as shutting model. This model is written as,

$$C_m \frac{dv_m}{dt} = -(E_P + v_m)g_p + (E_{Na} - V_m)gN_a - (E_k + V_m)g_k \quad (41)$$

C_m => Membrane capacitance

E_P => Nernst potentials for the passive leak

E_{Na} => Sodium ions in the membrane

E_k => Potassium ions in the membrane.

g_p => conductance of the passive channel

gN_a => Conductance of the sodium

g_k => Conductance of the potassiu0m

Considering $C_m = 1\ and\ Ep = 0$ and substituting the values $E_P + v_m = x_i,\ A1 = g_p, B1 = E_{Na} + E_P,\ D1 = E_k - E_P\ ,\ S_i^+ = gN_a\ , S_i^- = g_k$ in equation (41)

We get, $\frac{dx_i}{dt} = -A_1 x_i + (B_1 - x_i)S_i^+ - (D_1 + x_i)S_i^-$ (42)

x_i => This is the neural activity of the i-th neuron.

A1, B1, D1 => non-negative constants.

S_i^+ => Excitatory inputs

S_i^- => Inhibitory inputs.

This proposed control strategy was able to eliminate the velocity jumps and provide smooth velocity commands. The author implemented the shunting model

into conventional backstepping control, in which he defined the bio-inspired backstepping controls as,

$$v_r = v_s + v_d \cos e_\theta \tag{43}$$

$$\omega_r = \omega_d + C_2 v_d e_L + C_3 v_d \sin e_\theta \tag{44}$$

v_s => It is from the neural dynamic equation.

Differentiating with respect to the error in the driving direction as,

$$\frac{dv_s}{dt} = -A_1 v_s + (B_1 - v_s)\, f(e_D) - (D_1 + v_s) g(e_D) \tag{45}$$

$f(e_D)$ => max $\{e_D, 0\}$ => Linear above threshold function.

$g(e_D)$ => max $\{-e_D, 0\}$ => Non-linear function.

A1 => Passive decay rate

B1, D1 => upper and lower bound of velocity.

According to Lyapunov candidate function we get,

$$\text{V1} = \frac{1}{2} e_D^2 + \frac{1}{2} e_L^2 + \frac{1}{C_2}(1 - \cos e_\theta) + \frac{1}{2B_1} v_s^2 \tag{46}$$

$$\frac{dv_1}{dt} = 2\left(\frac{1}{2}\right) e_D \cdot \dot{e}_D + 2\left(\frac{1}{2}\right) e_L \dot{e}_L + \frac{1}{c_2}(0 + \sin e_\theta \cdot \dot{e}_\theta) + 2\left(\frac{1}{2B_1}\right) v_s \cdot \dot{v}_s$$

$$\dot{v}_1 = \dot{e}_D e_D + \dot{e}_L e_L + \frac{1}{C_2} \dot{e}_\theta \sin e_\theta + \frac{1}{B_1} \dot{v}_s v_s \tag{47}$$

B. TORQUE CONTROLLER:

In this part, the author implemented torque control law τ_c. The difference in the velocities from the backstepping controller is due to the noises. The velocity control error is given as

$$e_\eta = \begin{bmatrix} e_v \\ e_\omega \end{bmatrix} = \begin{bmatrix} v_r - v_c \\ \omega_r - \omega_c \end{bmatrix} \tag{48}$$

The torque law is proposed as $[\tau_L, \tau_R]^T$ is defined as,

$$\tau_L = \frac{mr}{2}(\dot{v}_r + C_4 e_v) - \frac{Ir}{2c}(\dot{\omega}_r + C_5 e_\omega) \quad (49)$$

$$\tau_R = \frac{mr}{2}(\dot{v}_r + C_4 e_v) - \frac{Ir}{2c}(\dot{\omega}_r + C_5 e_\omega) \quad (50)$$

The Lyapunov candidate function is given as,

$$V_2 = V_1 + \frac{1}{2}e_v^2 + \frac{1}{2}e_w^2 \quad (51)$$

Taking the time derivative we get,

$$\frac{dV_2}{dt} = \dot{V}_1 + 2\left(\frac{1}{2}\right)e_v \dot{e}_v + 2\left(\frac{1}{2}\right)e_\omega \dot{e}_\omega$$

$$V_2 = \dot{V}_1 + e_v(\dot{v}_r - \dot{v}_c) + e_w(\dot{\omega}_r - \dot{\omega}_c) \quad (52)$$

C. STABILITY ANALYSIS:

According to the control strategy, the controllers are firstly to be proven asymptotically stable. To do this, the author assumed that the posture and velocity errors are bounded.

$$\dot{V}_1 = -v_s e_D - \frac{C_3}{C_2} v_d \sin^2 e_\theta + \frac{1}{B_1}[-A_1 - f(e_D) - g(e_D)]v_s^2$$

$$+\frac{1}{B_1}[B_1 f(e_D) - D_1 g(e_D)]v_s \quad (53)$$

Here we are assuming B1 = D1, so we get,

$$\dot{V}_1 = -\frac{C_3}{C_2} v_d \sin^2 e_\theta + \frac{1}{B_1}[-A_1 - f(e_D) - g(e_D)]v_s^2 + [f(e_D) -$$

$$-g(e_D) - (e_D)]v_s \quad (54)$$

Based on the definition of $f(e_D)$ and $g(e_D)$. If $e_D \geq 0$, $f(e_D) = e_D$ and $g(e_D) = 0$, we get,

$$[f(e_D) - g(e_D) - e_D]v_S = e_D - 0 - e_D = 0 \quad (55)$$

Similarly, applying the same concept. If If $e_D \leq 0$, $f(e_D) = 0$ and

$g(e_D) = e_D$, then we get,

$$[f(e_D) - g(e_D) - e_D]v_S = -0 - (-e_D) - e_D = 0 \quad (56)$$

Now equation (55) becomes,

$$\dot{V}_1 = -\frac{C_3}{C_2} v_d \sin^2 e_\theta + \frac{1}{B_1}[-A_1 - f(e_D) - g(e_D)]v_S^2 \quad (57)$$

C2, C3, v_d => Positive constants

$g(e_D), f(e_D)$ => non-negative constants

Therefore $\dot{V}_1 \leq 0$. If $e_D = 0$ and $e_\theta = 0$, then the controller is asymptotically stable.

To prove the stability of the torque controller, we need to apply the dynamic equation (5) into equation (52), we get

$$\dot{V}_2 = \dot{V}_1 + e_v \left(\dot{v}_r - \frac{1}{mr}\tau_L - \frac{1}{m}\tau_R \right) + e_w \left(\dot{\omega}_r + \frac{1}{Ir}\tau_L - \frac{1}{Ir}\tau_R \right) \quad (58)$$

Putting the equation (49) and (50) for the values of τ_L, τ_R. We get new

$$\dot{V}_2 = \dot{V}_1 + e_v \left(\dot{v}_r - \frac{1}{mr} \left(\frac{mr}{2}(\dot{v}_r + C_4 e_v) - \frac{Ir}{2c}(\dot{\omega}_r + C_5 e_\omega) \right) - \frac{1}{m} \left(\frac{mr}{2}(\dot{v}_r + C_4 e_v) - \frac{Ir}{2c}(\dot{\omega}_r + C_5 e_\omega) \right) \right) + e_w \left(\dot{\omega}_r + \frac{1}{Ir} \left(\frac{mr}{2}(\dot{v}_r + C_4 e_v) - \frac{Ir}{2c}(\dot{\omega}_r + C_5 e_\omega) \right) - \frac{1}{Ir} \left(\frac{mr}{2}(\dot{v}_r + C_4 e_v) - \frac{Ir}{2c}(\dot{\omega}_r + C_5 e_\omega) \right) \right)$$

Solving we get,

$$\dot{V}_2 = \dot{V}_1 - C_4 e_v^2 - C_5 e_\omega^2 \quad (59)$$

Here all the values are less than one so $\dot{V}_2$ will becomes almost zero. Therefore, torque controller is asymptotically stable. Now for the final and overall stability analysis can be defined as,

$$V_3 = V_1 + V_2 \quad (60)$$

$$V_3 = \left(\frac{1}{2}e_D^2 + \frac{1}{2}e_L^2 + \frac{1}{C_2}(1 - \cos e_\theta) + \frac{1}{2B_1}v_s^2\right) + \left(V_1 + \frac{1}{2}e_v^2 + \frac{1}{2}e_w^2\right)$$

$$\dot{V}_3 = 2\left(-\frac{C_3}{C_2}v_d \sin^2 e_\theta + \frac{1}{B_1}[-A_1 - f(e_D) - g(e_D)]v_s\right) - C_4e_v^2 - C_5e_\omega^2 \quad (61)$$

If $e_p, e_\eta = 0$, then V3= 0, then the control system becomes globally asymptotically stable.

6. **<u>Simulations and results</u>**:

   The author used KF and UKF model to track the system in straight line and curved line. But I worked on curve path. Author used his own values to create plots which he didn't mention on the paper. So, I provided dummy values to trace the path. I have also included the code and the plot obtained. And mentioned the values in the comments in-between the codes.

   Code:

```
import numpy as np
import matplotlib.pyplot as plt
# Robot Constants mentioned in paper
C2 = 5
C3 = 2
C4 = 5
C5 = 5
A1 = 5
B1 = 3
D1 = 3

# robot physical parameters
m = 10
I = 0.1
l = 1
r = 1
delta_t = 0.1 # sec

# Initialize the KALMAN variables (According to paper)
Q = np.array([[np.exp(-5), 0], [0, np.exp(-6)]])
R = np.array([[np.exp(-2), 0], [0, np.exp(-3)]])
P = 10*Q
```

```
H = np.array([[1, 0], [0, 1]])
vc_prev = 0
wc_prev = 0

# Robot location xc, yc, thetac - in world coordinates
X_robot_world = np.array([[0, 0, 0]]).T
# Robot location xcdot, ycdot, thetacdot - in world coordinates
X_robot_dot_world = np.array([[0, 0, 0]]).T

# Dynamic robot parameters
M = np.array([[m, 0],[0, 1]])
B = np.array([[1,1],[1, 1]])*(1/r)


# TODO: Desired position and orientation is supposed to be provided by
path planner.
P_d_world = np.array([[2, 2, 0]]).T
P_ddot_world = np.array([[0, 0, 0]]).T
P_ddotdot_world = np.array([[0, 0, 0]]).T

# ex, ey, etheta
e_robot_prev = np.array([[0, 0, 0]]).T
e_d_robot_prev = np.array([[0, 0, 0]]).T
e_world = P_d_world - X_robot_world
count = 200

vr_prev = 0
wr_prev = 0

xcoord = []
ycoord = []

xnoise = []
ynoise = []

# Initialize parameters for UKF - _u means parameters for ukf
Q_u = np.array([[np.exp(-5), 0, 0], [0, np.exp(-5), 0], [0, 0, np.exp(-
6)]])
R_u = np.array([[np.exp(2), 0, 0], [0, np.exp(2), 0], [0, np.exp(3),
np.exp(3)]])
P_u = 10*Q_u
H_u = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])

while count > 0:
    ###################### Robot coordinate frame
#########################
    # eD, eL, etheta
    robot_T_world = np.array([[np.cos(X_robot_world[2,0]),
np.sin(X_robot_world[2,0]), 0], [-np.sin(X_robot_world[2,0]),
np.cos(X_robot_world[2,0]), 0], [0, 0, 1]])
    e_robot = robot_T_world @ e_world
    print("Erobot", e_robot)
    e_d_robot = (e_robot - e_robot_prev) / delta_t
    e_robot_prev = e_robot
```

```
    e_ddot_robot = (e_d_robot - e_d_robot_prev) / delta_t
    e_d_robot_prev = e_d_robot
    # desired velocity and desired angular velocity -
    vd = np.sqrt(e_robot[0,0]*e_robot[0,0] + e_robot[1,0]*e_robot[1,0]) *
0.05 + np.sqrt(e_d_robot[0,0]*e_d_robot[0,0] +
e_d_robot[1,0]*e_d_robot[1,0]) * 0.05
    wd = e_robot[2,0] * 0.01 + e_d_robot[2,0] * 0.3

    # Equation 2
    vc = X_robot_dot_world[1,0] / np.cos(X_robot_world[2,0])
    wc = X_robot_dot_world[2,0]

    ########## Backtracking controller
##############################################
    vr = vd * np.cos(e_robot[1,0])
    wr = wd + C2*vd*e_robot[1,0] + C3*vd*np.sin(e_robot[2,0])

    # print("Vr", vr)
    # print("wr", wr)

    vrdot = (vr - vr_prev) / delta_t
    wrdot = (wr - wr_prev) / delta_t
    vr_prev = vr
    wr_prev = wr
    # Equation 48
    en = np.array([vr - vc, wr - wc])
    print("En", en)

    # Equation 49 and 50
    tL  = m*r*0.5*(vrdot*C4*en[0]) - I*r*0.5*(wrdot*C5*en[1])
    tR  = m*r*0.5*(vrdot*C4*en[0]) + I*r*0.5*(wrdot*C5*en[1])

    # Prediction
    Xpred = np.linalg.inv(M) @ B @ np.array([[tR, tL]]).T * delta_t * 0.1
    F = np.array([[1, 0],[0, 1]])
    Ppred = F @ P @ F.T + Q
    # print("Xpred", Xpred)
    # print("tL", tL)
    # print("tR", tR)
    # Xpred[0,0] = vr
    # Xpred[1,0] = wr
    # Correction
    # TODO: Get a sequence of observations here
    z = Xpred + np.random.rand(2,1)*0.001
    print("z",z)
    y = z - Xpred
    S = H @ Ppred @ H.T + R
    K = Ppred * H.T * np.linalg.inv(S)

    Xestimate = Xpred + K@y
    Pestimate = (np.identity(2) - K@H) @ Ppred

    print("Xestimate", Xestimate)
```

```
    # print("Pestimate", Pestimate)

    # just renaming for future use
    VelcityInputEstimate = Xestimate

    ################################################### UKF helper
functions ############################################
    def ukf_predict(x0, X_robot_world, velocityInputEstimate):
        x_pred = np.array([[0,0,0]],dtype=np.float64).T
        x_pred[0,0] = (x0[0,0] +
velocityInputEstimate[0]*np.cos(X_robot_world[2,0])*delta_t)[0]
        x_pred[1,0] = (x0[1,0] +
velocityInputEstimate[0]*np.sin(X_robot_world[2,0])*delta_t)[0]
        x_pred[2,0] = (x0[2,0] + velocityInputEstimate[1]*delta_t)[0]
        # print("X", x_pred)
        return x_pred

    ################################################### UNSCENTED KALMAN
FILTER ###########################################
    # step1: generate L from P matrix
    L = np.linalg.cholesky(P_u)
    # print("L", L)
    # print("L0", L[:,0])

    # step 1 generate 2N+1 = 7 sigma points
    x0 = X_robot_world
    x1 = X_robot_world + np.array([np.sqrt(3)*L[:,0]]).T
    x2 = X_robot_world + np.array([np.sqrt(3)*L[:,1]]).T
    x3 = X_robot_world + np.array([np.sqrt(3)*L[:,2]]).T
    x4 = X_robot_world - np.array([np.sqrt(3)*L[:,0]]).T
    x5 = X_robot_world - np.array([np.sqrt(3)*L[:,1]]).T
    x6 = X_robot_world - np.array([np.sqrt(3)*L[:,2]]).T

    # predict next output of sigma points
    x0 = ukf_predict(x0, X_robot_world, VelcityInputEstimate)
    x1 = ukf_predict(x1, X_robot_world, VelcityInputEstimate)
    x2 = ukf_predict(x2, X_robot_world, VelcityInputEstimate)
    x3 = ukf_predict(x3, X_robot_world, VelcityInputEstimate)
    x4 = ukf_predict(x4, X_robot_world, VelcityInputEstimate)
    x5 = ukf_predict(x5, X_robot_world, VelcityInputEstimate)
    x6 = ukf_predict(x6, X_robot_world, VelcityInputEstimate)

    # compute mean
    x_mean = x0 + x1 + x2 + x3 + x4 + x5 + x6
    x_mean = x_mean / 7
    # print("Xmean:",x_mean)

    P_u_pred = (x0 - x_mean) @ (x0 - x_mean).T
    P_u_pred += (x1 - x_mean) @ (x1 - x_mean).T
    P_u_pred += (x2 - x_mean) @ (x2 - x_mean).T
    P_u_pred += (x3 - x_mean) @ (x3 - x_mean).T
    P_u_pred += (x4 - x_mean) @ (x4 - x_mean).T
    P_u_pred += (x5 - x_mean) @ (x5 - x_mean).T
    P_u_pred += (x6 - x_mean) @ (x6 - x_mean).T
```

```
    P_u_pred = P_u_pred / 7


    ############### Correction ###################

    # Since paper uses identity matrix for measurement, predicted sigma
points are the observations.
    L = np.linalg.cholesky(P_u_pred)
    y0 = x_mean
    y1 = x_mean + np.array([np.sqrt(3)*L[:,0]]).T
    y2 = x_mean + np.array([np.sqrt(3)*L[:,1]]).T
    y3 = x_mean + np.array([np.sqrt(3)*L[:,2]]).T
    y4 = x_mean - np.array([np.sqrt(3)*L[:,0]]).T
    y5 = x_mean - np.array([np.sqrt(3)*L[:,1]]).T
    y6 = x_mean - np.array([np.sqrt(3)*L[:,2]]).T

    # compute observation mean
    y_mean = y0 + y1 + y2 + y3 + y4 + y5 + y6
    y_mean = y_mean / 7
    # print("Y mean", y_mean)

    Py = (y0 - y_mean) @ (y0 - y_mean).T
    Py += (y1 - y_mean) @ (y1 - y_mean).T
    Py += (y2 - y_mean) @ (y2 - y_mean).T
    Py += (y3 - y_mean) @ (y3 - y_mean).T
    Py += (y4 - y_mean) @ (y4 - y_mean).T
    Py += (y5 - y_mean) @ (y5 - y_mean).T
    Py += (y6 - y_mean) @ (y6 - y_mean).T

    Py = Py/ 7 + R_u

    Pxy = (x0 - x_mean) @ (y0 - y_mean).T
    Pxy += (x1 - x_mean) @ (y1 - y_mean).T
    Pxy += (x2 - x_mean) @ (y2 - y_mean).T
    Pxy += (x3 - x_mean) @ (y3 - y_mean).T
    Pxy += (x4 - x_mean) @ (y4 - y_mean).T
    Pxy += (x5 - x_mean) @ (y5 - y_mean).T
    Pxy += (x6 - x_mean) @ (y6 - y_mean).T


    K_u = Pxy @ np.linalg.inv(Py)
    # print("K_u",K_u)

    # Estimation step
    # TOD0: Assume measurements are almost accurate
    y_measurement = x_mean + np.random.rand(3,1)*0.003
    print("Y measure: ", y_measurement)
    X_robot_estimated = x_mean + K_u @ (y_measurement - y_mean)
    P_u_estimated = P_u_pred - (K_u @ Py @ K_u.T)

    world_T_robot = np.array([[np.cos(X_robot_estimated[2,0]), -
np.sin(X_robot_estimated[2,0]), 0], [np.sin(X_robot_estimated[2,0]),
np.cos(X_robot_estimated[2,0]), 0], [0, 0, 1]])
```

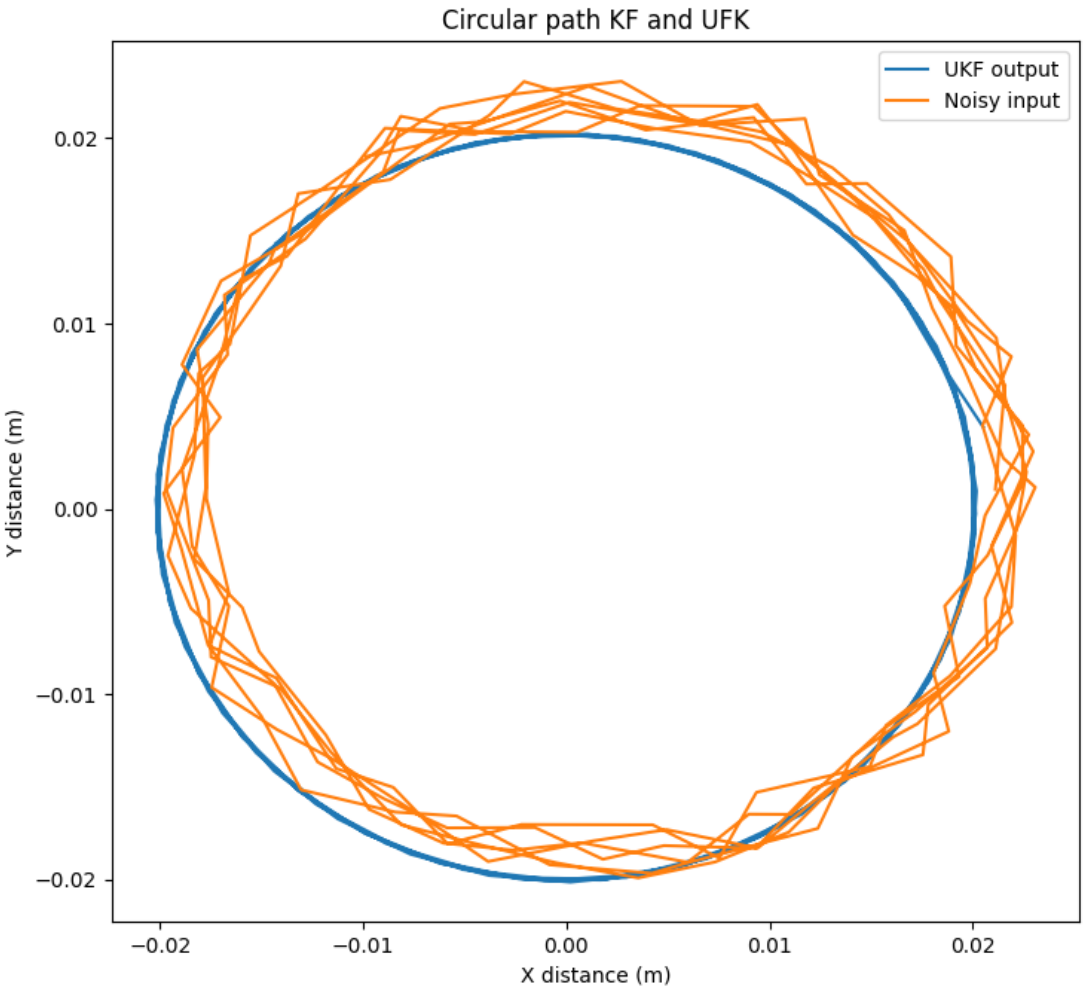
```
    X_robot_world_estimated = world_T_robot @ X_robot_estimated

    # print(e_robot_T_world)
    # print("Input velocity\n", VelcityInputEstimate)
    print("Robot final position\n", X_robot_world_estimated)
    # print("P final\n", P_u_estimated)

    # Update the states
    X_robot_dot_world = X_robot_world_estimated - X_robot_world
    X_robot_dot_world /= delta_t
    X_robot_world = X_robot_world_estimated
    P_u = P_u_estimated
    e_world = P_d_world - X_robot_world
    count -= 1
    # print("E:", e_world)
    print("Xrobot dot:", X_robot_dot_world)

    xcoord.append(X_robot_world[0,0])
    ycoord.append(X_robot_world[1,0])
    xnoise.append(y_measurement[0,0])
    ynoise.append(y_measurement[1,0])

plt.plot(xcoord, ycoord, label="UKF output")
plt.plot(xnoise, ynoise, label="Noisy input")
plt.title("Circular path KF and UFK")
plt.xlabel("X distance (m)")
plt.ylabel("Y distance (m)")
plt.legend()
plt.show()
```

The Figure 5 shows the plot obtained from the tracking algorithm. It shows that the robot tracks the desired path more accurately compared to the noisy values. The author also used the root mean square error formula which calculates the mobile robot path and the path without noises. This is used to validate the KF and UKF in the proposed control strategy. The root mean square formula is defined as,

$$\text{RMSE} = \sqrt{\sum_{i=1}^{N} \frac{(p-\tilde{P})^2}{N}}$$

P- mobile robot state without noise

$\tilde{P}$= Estimated state of the mobile robot.



7. Conclusion:

The novel tracking metho proposed by the author was able to track the robot trajectory accurately. The closed loop system is asymptotically stable and is guaranteed by Lyapunov stability theory. Thus, the practical solution proposed by the author is proved theoretically.

8. References:

1. http://msl.cs.uiuc.edu/~lavalle/cs576_1999/projects/junqu/
2. https://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/

3. https://calcworkshop.com/first-order-differential-equations/eulers-method-table/
4. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.436.8434&rep=rep1&type=pdf
5. https://www.sciencedirect.com/science/article/pii/B9780128175828000088
6. Coursera – self driving cars – state estimation
7. https://groups.seas.harvard.edu/courses/cs281/papers/unscented.pdf
8. https://www.cse.sc.edu/~terejanu/files/tutorialUKF.pdf
9. Udacity – Self driving cars - Kalman filter