A Comprehensive Simulation of a Leader-Follower Robot System Using Extended Kalman Filters

Distributed estimation for robots and vehicles

***Abstract*—This paper presents a comprehensive MatLab code based simulation of a leader-follower robot system. The leader robot navigates using the pure pursuit algorithm, while follower robots maintain a safe distance using PID controllers. the leader robot navigates a predefined path using the pure pursuit algorithm, while the followers maintain a safe distance from the leader using a combination of camera measurements,wheel encoders and a PID controller. The system employs Extended Kalman Filters (EKF) for state estimation, incorporating sensor data from GPS, wheel encoders, and cameras. The simulation demonstrates the effectiveness of EKF in handling sensor noise and GPS outages, ensuring robust and accurate trajectory tracking for all robots.**

***Index Terms*—Leader-Follower Robots, EKF, PID Control, Pure Pursuit, Dead Reckoning, Low-Pass Filter**

I. Introduction

Mobile robots have become integral to various applications, including industrial automation, search and rescue, and autonomous driving. Accurate navigation and control are crucial for these robots to perform tasks efficiently and safely. This paper explores a leader-follower robot system where the leader navigates a predefined path, and the followers maintain a specified distance using state estimation and control algorithms.

II. Kinematic Modeling and Dead Reckoning

Kinematic modeling of differential drive mobile robots involves describing their motion using linear and angular velocities. Let x and y represent the position coordinates, θ represent the orientation, v the linear velocity, and ω the angular velocity. In continuous time, the model is given by the following differential equations:

$$\begin{aligned} \dot{x} &= v\cos(\theta) \\ \dot{y} &= v\sin(\theta) \\ \dot{\theta} &= \omega \end{aligned} \tag{1}$$

In discrete time, these equations can be represented as:

$$\begin{aligned} x_{k+1} &= x_k + v_k\cos(\theta_k)\Delta t \\ y_{k+1} &= y_k + v_k\sin(\theta_k)\Delta t \\ \theta_{k+1} &= \theta_k + \omega_k\Delta t \end{aligned} \tag{2}$$

Dead Reckoning (DR) is a technique for estimating the current position of a robot based on its previous position and motion information. DR is prone to cumulative errors over time, especially in mobile robots, affecting their navigation accuracy. DR is particularly used during GPS outages to maintain position estimates.

III. Extended Kalman Filter (EKF)

The EKF is a recursive algorithm used for state estimation in nonlinear systems. It consists of two main stages:

A. Prediction Stage

In the prediction stage, the EKF uses the robot's kinematic model to predict its next state and update the error covariance matrix. The state prediction equations are:

$$\mathbf{x}_{k|k-1} = f(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}) \tag{3}$$

The error covariance prediction is:

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k\mathbf{P}_{k-1}\mathbf{F}_k^T + \mathbf{Q}_k \tag{4}$$

where $\mathbf{F}_k$ is the Jacobian of $f(\cdot)$ with respect to the state, $\mathbf{P}$ is the error covariance matrix, and $\mathbf{Q}$ is the process noise covariance matrix.

B. Update Stage

In the update stage, the EKF incorporates sensor measurements to correct the predicted state.

1) GPS and Wheel Encoder: GPS provides global position data, while wheel encoders measure the rotational speed of the wheels. These sensors are crucial in real-world applications for accurate navigation and position tracking.

The EKF uses a measurement model that combines GPS and wheel encoder data to update the robot's state.

The measurement model for GPS with noise is:

$$\mathbf{z}_k = \begin{bmatrix} x \\ y \end{bmatrix} + \mathbf{v}_k^{GPS} \tag{5}$$

where $\mathbf{v}_k^{GPS}$ is the GPS measurement noise, typically modeled as a zero-mean Gaussian noise.

The measurement model for wheel encoders with noise is:

$$\mathbf{z}_k = \begin{bmatrix} \omega_l \\ \omega_r \end{bmatrix} + \mathbf{v}_k^{Enc} \tag{6}$$

where ω_l and ω_r are the left and right wheel angular velocities, and $\mathbf{v}_k^{Enc}$ represents the encoder noise, also modeled as zero-mean Gaussian noise.

The Jacobian matrix $\mathbf{H}_k$ for GPS measurements is:

$$\mathbf{H}_k = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{7}$$

For wheel encoder measurements, the Jacobian matrix $\mathbf{H}_k$ is:

$$\mathbf{H}_k = \begin{bmatrix} 0 & 0 & \frac{-1}{r} \\ 0 & 0 & \frac{1}{r} \end{bmatrix} \tag{8}$$

The innovation vector is:

$$\mathbf{y}_k = \mathbf{z}_k - h(\mathbf{x}_{k|k-1}) \tag{9}$$

The measurement update equations are:

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k \tag{10}$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^T \mathbf{S}_k^{-1} \tag{11}$$

$$\mathbf{x}_k = \mathbf{x}_{k|k-1} + \mathbf{K}_k \mathbf{y}_k \tag{12}$$

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1} \tag{13}$$

where $\mathbf{H}_k$ is the measurement Jacobian, and $\mathbf{R}_k$ is the measurement noise covariance.

2) Camera and Wheel Encoder: Cameras provide visual information about the environment, while wheel encoders measure wheel rotations. These sensors enhance state estimation accuracy.

The EKF integrates camera measurements with wheel encoder data to refine the robot's state estimates.

The measurement model for the camera and wheel encoder with noise is:

$$\mathbf{z}_k = \begin{bmatrix} d \\ \alpha \end{bmatrix} + \mathbf{v}_k^{Cam} \tag{14}$$

where d is the distance to the leader, and α is the angle to the leader. The noise $\mathbf{v}_k^{Cam}$ is typically modeled as zero-mean Gaussian noise.

The Jacobian matrix $\mathbf{H}_k$ for camera measurements is:

$$\mathbf{H}_k = \begin{bmatrix} \frac{\partial d}{\partial x} & \frac{\partial d}{\partial y} & 0 \\ \frac{\partial \alpha}{\partial x} & \frac{\partial \alpha}{\partial y} & \frac{\partial \alpha}{\partial \theta} \end{bmatrix} \tag{15}$$

where the partial derivatives are calculated as follows:

$$\frac{\partial d}{\partial x} = \frac{x - x_{leader}}{\sqrt{(x - x_{leader})^2 + (y - y_{leader})^2}} \tag{16}$$

$$\frac{\partial d}{\partial y} = \frac{y - y_{leader}}{\sqrt{(x - x_{leader})^2 + (y - y_{leader})^2}} \tag{17}$$

$$\frac{\partial \alpha}{\partial x} = -(\frac{\partial \alpha}{\partial x} = \frac{-(y - y_{leader})}{(x - x_{leader})^2 + (y - y_{leader})^2} \tag{18}$$

$$\frac{\partial \alpha}{\partial y} = \frac{(x - x_{leader})}{(x - x_{leader})^2 + (y - y_{leader})^2} \tag{19}$$

$$\frac{\partial \alpha}{\partial \theta} = -1 \tag{20}$$

The innovation vector is:

$$\mathbf{y}_k = \mathbf{z}_k - h(\mathbf{x}_{k|k-1}) \tag{21}$$

The measurement update equations are the same as for the GPS and wheel encoder model, with different $\mathbf{H}_k$ and $\mathbf{R}_k$.

IV. Low-Pass Filter

A low-pass filter is applied to the state estimates to smooth out high-frequency noise and produce more stable state estimates. The low-pass filter can be described by:

$$\mathbf{x}_{filtered} = \alpha \mathbf{x} + (1 - \alpha) \mathbf{x}_{previous} \tag{22}$$

where α is the smoothing factor.

V. PID Control

PID controllers are used for the follower robots to maintain a safe distance from the leader. The control law for the PID controller is given by:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \tag{23}$$

where $e(t)$ is the error between the desired and actual positions, K_p is the proportional gain, K_i is the integral gain, and K_d is the derivative gain.

Relative Velocity and Distance Maintenance: The PID controller takes into account the relative velocity and distance between the leader and the follower robots. The error $e(t)$ is defined as the difference between the desired distance $d_{desired}$ and the actual distance d_{actual}:

$$e(t) = d_{desired} - d_{actual} \tag{24}$$

The controller also uses the relative velocity $v_{relative}$, which is the difference between the velocities of the leader and follower robots. The control input $u(t)$ adjusts the follower's velocity to maintain the desired distance:

$$u(t) = v_{leader} + K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \tag{25}$$

By continuously adjusting the follower's velocity based on the PID control law, the system ensures that the follower robot maintains the desired distance from the leader. This approach helps to minimize oscillations and ensures smooth following behavior.

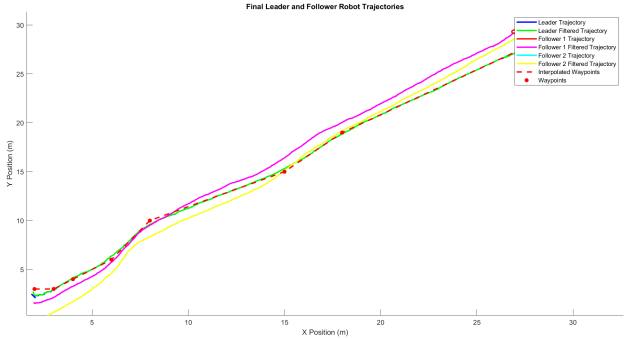
VI. Pure Pursuit Algorithm

The leader robot navigates the path using the pure pursuit algorithm, which adjusts the robot's steering angle to follow a lookahead point on the path. The lookahead point is selected such that it is at a fixed distance ahead of the robot's current position. The control law for pure pursuit is:

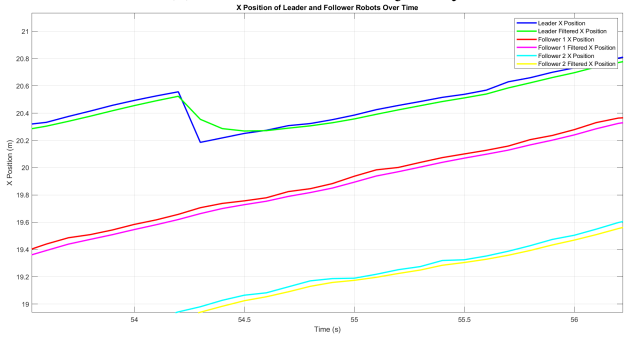
$$\delta = \arctan\left(\frac{2L \sin(\alpha)}{L_d}\right) \tag{26}$$

where δ is the steering angle, L is the wheelbase, α is the angle to the lookahead point, and L_d is the lookahead distance.

The leader robot's trajectory is determined by this control law, while the follower robots use their updated state information to maintain the desired formation behind the leader.



(a) Leader-follower Trajectory



(b) Zoom view to see the effect of filter on trajectory

VII. Simulating Sensor Measurements and Handling GPS Outages

The simulation incorporates sensor measurements with noise and handles potential GPS outages. The process is as follows:

A. Simulating Sensor Measurements

1. **Wheel Encoders:** - Simulate noisy measurements for the left and right wheel angular velocities:

$$\omega_l = \frac{v_{leader}}{r} + 0.05 \cdot \text{randn}(1) \tag{27}$$

$$\omega_r = \frac{v_{leader}}{r} + 0.05 \cdot \text{randn}(1) \tag{28}$$

- Clamp the encoder measurements to realistic values.

(29)

2. **GPS Measurements:** - Simulate GPS outages with a certain probability:

$$\text{gpsOutageProbability} = 0.005 \tag{30}$$

- During an outage, set GPS measurements to NaN and use dead reckoning:

$$\text{gpsMeasurement} = [\text{NaN}; \text{NaN}] \tag{31}$$

- Otherwise, simulate noisy GPS measurements:

$$\text{gpsMeasurement} = \text{state}(1:2) + 0.5 \cdot \text{randn}(2,1) \tag{32}$$

- Clamp GPS measurements to the map size.

$$\text{gpsMeasurement} = \min(\max(\text{gpsMeasurement}, 0), \text{mapSize}) \tag{33}$$

Simulating GPS Outages determines the probability of a GPS outage occurring at each time step. A random number is generated, and if it's less than this probability, a GPS outage is simulated. When a GPS outage occurs, the GPS Measurement is set to NaN (Not a Number), indicating that no valid GPS data is available.

GPS Outages timeout detects persistent outages defines the duration of a GPS outage before it's considered "persistent." The algorism checks if the time elapsed since the last valid GPS update exceeds the GPS Outage Timeout. If so, it means the GPS outage has lasted longer than the timeout, and the code considers it a persistent outage. When a persistent GPS outage is detected, the GPS Outage is Active. A message is printed to the console indicating that a persistent GPS outage has occurred.

When a GPS outage occurs (whether persistent or not), the leader's robot state is set to the Dead Reckoning State. This means the leader's position and orientation are now based on the last valid GPS update and the robot's motion model.

Dead Reckoning State variable stores the leader's state at the last valid GPS update. It's used to maintain the leader's position estimate during GPS outages. When a valid GPS measurement is received, the dead Reckoning State is updated to the current leader's state . This ensures that the dead reckoning state is always based on the most recent valid GPS data. If a persistent GPS outage is active, the code only uses encoder measurements to update the leader's state using the extended Kalman filter update function. This means the leader's position estimate is refined based on its wheel rotations, but without any GPS corrections.

When a valid GPS measurement is received, the GPS Outage become deactivate, indicating that the GPS outage has ended. The code resumes using both encoder and GPS measurements to update the leader's state using the EKF update function. This allows the leader to correct for any drift that occurred during the GPS outage.

3. **Error Detection and Handling:** - Check for invalid measurements and use previous valid state if necessary.

This condition checks for the validity of the sensor measurements. If the measured angular velocities of the left ('ω_l') or right ('ω_r') wheels exceed the maximum allowable wheel speed ('maxWheelSpeed'), or if any value in the encoder or GPS measurements is 'NaN' (Not a Number), the measurement is considered invalid. This ensures that the code can detect and handle erroneous sensor measurements, reverting to the previous valid state when necessary.

VIII. Handling Signal Shortages for Followers

In the event of a signal shortage, Follower 1 and Follower 2 use dead reckoning as a fallback mechanism.

A. Follower 1 Signal Shortage

The code checks if Follower 1 receives an update from the leader within a specified timeout period . This timeout is set to 5 seconds. If the time elapsed since the last update from the leader exceeds this timeout, Follower 1 is considered to be in a signal outage.

When Follower 1 detects a signal outage from the leader, it enters "dead reckoning" mode. This means it uses its own internal sensors (encoders) and the last known valid velocity to estimate its position and orientation. The code predicts the follower's state using the extended Kalman filter predict function, but without any updates from the leader.

While in dead reckoning mode, Follower 1 continues to move with the last known valid velocity. However, its position estimate will drift over time due to the lack of updates from the leader. This drift is mitigated by the use of the Extended Kalman Filter (EKF), which incorporates the follower's motion model and sensor noise to refine the state estimate.

When Follower 1 receives a new update from the leader, it exits dead reckoning mode and resumes normal operation. The EKF will then incorporate the new measurements from the leader to improve the follower's state estimate.

B. Follower 2 Signal Shortage

Similarly, in Follower 2 algorism checks if it receives an update from Follower 1 within a timeout period also set to 5 seconds. If the timeout is exceeded, Follower 2 is considered to be in a signal outage. Follower 2 also enters dead reckoning mode when it loses communication with Follower 1.

It uses its own encoders and the last known valid velocity to estimate its state, again using the extended Kalman predict function without updates. Follower 2 behaves similarly to Follower 1 during dead reckoning. It continues to move with the last known valid velocity, and its state estimate is refined using the EKF.

Follower 2 resumes normal operation when it receives a new update from Follower 1. The EKF will then incorporate the new measurements from Follower 1 to improve its state estimate.

In both cases, dead reckoning is used until the signal is re-established, at which point the EKF incorporates the new measurements to correct the state estimates.

IX. Conclusion

This paper demonstrates the MATLAB code simulation implementation of a leader-follower robot system using EKF for state estimation and PID controllers for follower control. it provides with a comprehensive simulation of a multi-robot system with advanced navigation and control techniques. It demonstrates the use of the Pure Pursuit algorithm, EKF state estimation, dead reckoning, camera-based control, GPS usage and PID control in a realistic scenario.The simulation results show that the EKF effectively handles sensor noise and GPS outages, ensuring accurate trajectory tracking. The PID controllers efficiently manage the relative velocity and maintain the desired distance between the robots. Future work may involve real-world implementation and testing in dynamic environments.

References

[1] Daniele Fontanelli. (2023). *Course Lecture Notes on Distributed estimation for robots and vehicles*.Department of Industrial Engineering, University of Trento, Trento.

[2] Siegwart, R., Nourbakhsh, I. R., & Scaramuzza, D. (2011). *Introduction to Autonomous Mobile Robots*. MIT Press.

[3] Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic Robotics*. MIT Press.

[4] Corke, P. (2017). *Robotics, Vision and Control: Fundamental Algorithms In MATLAB*. Springer.

[5] Kelly, A. (2013). *Mobile Robotics: Mathematics, Models, and Methods*. Cambridge University Press.

[6] LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press.

[7] book6 Carpenter, J.R. and D'souza, C.N., 2018. Navigation filter best practices (No. NF1676L-29886).