**ODOMETRY**

Accurate localization of a vehicle is a fundamental challenge and one of the most important tasks of mobile robots. For autonomous navigation, motion tracking, and obstacle detection and avoidance, a robot must maintain knowledge of its position over time. In swarm robots, odometry plays a crucial role in localisation and navigation.

Here are the factors which contribute to accurate localization and navigation:

1. Hardware/Sensors: various sensors are attached such as wheel encoders to sense the no. of wheel rotations, tachometers to measure the rotation of shaft, optical flow sensors to track the movement of ground beneath the robot.
2. Data collection and calibration: Data from one sensor wouldn’t be accurate enough and lead to errors. Therefore, two or more sensors are attached and their data measured collectively, are used to determine position and velocity.
3. Motion model: A motion model/equation will allow us to interpret the linear or angular velocity using the data collected from the sensors, such as wheel movements, shaft rotations, accelerometer and gyroscope.
4. Error accumulation: Keep in mind that odometry measurements are prone to accumulating errors over time due to factors like wheel slippage, uneven terrain, and sensor noise. Implementing error correction mechanisms, such as periodically resetting the robot's position using external references, can help mitigate this issue.
5. Communication: In a swarm, robots can share their odometry information with neighboring robots to improve overall localization accuracy. This cooperative sharing of information can help the swarm maintain formation and perform tasks more effectively.

Here are several algorithms and techniques commonly used for odometry in swarm robots:

**Local Odometry (Encoder-Based) in Swarm Robots:**

1. Introduction:

Local odometry, especially when encoder-based, is a common technique used to estimate the position and movement of individual robots within a swarm. Encoders are sensors attached to the wheels of a robot that measure the rotation of the wheels, allowing the estimation of the distance traveled by the robot. This information is then used to calculate the robot's change in position and orientation.

1. Working Principle:

The basic principle of encoder-based local odometry involves measuring the number of encoder ticks as the wheels of the robot rotate. By knowing the wheel's circumference and the number of ticks per full rotation, you can calculate the distance traveled by each wheel. These distances are then used to estimate the change in position (x, y) and orientation (theta) of the robot over time.

1. Algorithm:

**Initialization:**

Initialize the robot's pose (position and orientation) to (0, 0, 0) or any other known starting point.

Set the initial encoder tick counts to zero.

**Loop:**

In a control loop (e.g., every time step), read the current encoder tick counts for both wheels.

Calculate the change in encoder ticks since the last reading.

**Calculating Wheel Distances:**

Convert the change in encoder ticks to distance for each wheel using the wheel's circumference and the number of ticks per rotation.

For example, distance = (change in encoder ticks) \* (wheel circumference) / (ticks per rotation).

**Calculating Change in Pose:**

Calculate the average distance traveled by the two wheels to estimate the distance the robot has moved forward (d\_center).

Calculate the change in orientation (d\_theta) based on the difference in the distances traveled by the left and right wheels.

**Updating Pose:**

Update the robot's pose based on the calculated d\_center and d\_theta.

Update the robot's x and y position as well as its orientation.

**Loop Continuation:**

Repeat the loop for each time step, continuously updating the robot's pose based on the encoder readings.

1. **Major challenge: Unobservable errors:** Encoder-based odometry is susceptible to errors due to factors like wheel slippage, uneven terrain, and wheel calibration inaccuracies. Over time, these errors can accumulate, leading to a divergence between the estimated and actual positions.
2. **Considerations:** To improve accuracy, some systems incorporate additional sensors, like gyros or accelerometers, to correct for errors that may occur during rotation or sudden changes in velocity. They can also be integrated with other localization methods, such as GPS or visual odometry, for improved accuracy over extended periods.

**COLLABORATING IT WITH ACCELEROMETER AND GYROSCOPE**

**Sensor Data Fusion:**

**Accelerometers:** Accelerometers measure linear acceleration along the three axes (x, y, and z). They can provide information about the robot's linear motion, including acceleration and deceleration.

**Gyroscopes (Gyros):** Gyroscopes measure angular velocity around the three axes. They provide information about the robot's rotational motion, including changes in orientation (angular rate).

**Integration of Sensor Data:**

Integrate accelerometer data twice to estimate the robot's position. First, integrate linear acceleration to velocity, and then integrate velocity to position. This provides an estimate of position over time.

Integrate gyroscope data to estimate changes in orientation (yaw, pitch, and roll angles).

**Use Sensor Fusion Algorithms:**

Implement sensor fusion algorithms, such as Kalman filters or complementary filters, to combine data from encoders, accelerometers, and gyroscopes.

Kalman filters, in particular, are effective at combining noisy sensor data and providing a robust estimate of pose.

**Error Correction:**

Use accelerometer data to correct for drift in encoder-based odometry over time. If the accelerometers detect linear acceleration, you can adjust the robot's position and velocity accordingly.

Gyroscope data can be used to correct for orientation drift or changes in orientation.

**Noise Filtering:**

Apply noise filtering techniques to sensor data to reduce noise and improve accuracy.

Low-pass filters can be applied to accelerometer data to reduce high-frequency noise.

Gyroscope data may require similar filtering to reduce noise in angular velocity measurements.

**Kalman Filters:**

A Kalman filter is a recursive algorithm that estimates the state of a dynamic system from a series of noisy measurements over time. It's particularly effective in situations where sensor measurements are subject to noise and uncertainty. The filter maintains two main components: a state estimate and a covariance matrix.

**Initialization:**

Initialize the state estimate, which includes the robot's position, velocity, and orientation.

Initialize the covariance matrix, which represents the uncertainty in the state estimate.

**Prediction Step:**

Predict the future state based on the system's dynamics model and control inputs (if available).

Predict the covariance matrix to account for the uncertainty introduced during the prediction.

**Update Step:**

Receive sensor measurements (from encoders, accelerometers, gyroscopes, etc.).

Calculate the Kalman gain, which determines how much weight to give to the sensor measurements versus the predicted state.

Update the state estimate and covariance matrix based on the sensor measurements and Kalman gain.

**Repeat:**

Continuously repeat the prediction and update steps as new sensor measurements arrive.

**Complementary Filters:**

A complementary filter combines data from multiple sensors by applying a weighted average. In the case of combining accelerometer and gyroscope data, it uses the accelerometer for estimating orientation in the gravity direction (roll and pitch) and the gyroscope for estimating orientation changes (yaw).

**Initialization:**

Initialize the estimated orientation using accelerometer data.

Set an initial complementary filter gain (often denoted as alpha).

**Read Sensor Data:**

Continuously read data from the accelerometer and gyroscope.

**Orientation Estimation:**

Use the accelerometer data to estimate roll and pitch angles (the angles with respect to gravity).

Use the gyroscope data to estimate angular velocity (rate of change of orientation).

**Complementary Filtering:**

Combine the accelerometer-based roll and pitch angles (low-pass filtered) with gyroscope-based yaw angles (high-pass filtered) using a complementary filter equation.

The complementary filter equation updates the estimated orientation by combining the accelerometer and gyroscope data with a weighted average, determined by the filter gain alpha.

**Repeat:**

Continuously update the estimated orientation as new sensor data arrives.

The complementary filter is computationally simpler than the Kalman filter and is well-suited for many applications where gyroscopes and accelerometers are used together to estimate orientation.

