Kinematic Model of Robot:

In a global frame of reference, the kinematic equations of motion are:

X_dot = (Vt)*cos α (t)*cos θ (t)

Y_dot = (Vt)*cos α (t)*sin θ (t)

Yaw_dot = (Vt)*sin α (t)/d

Vt = ω*r

There for the pose (tuple) of the robot can be determined by numerically integrating the derivatives of the states. I.e:

X = X_{init} + Δt*X_dot;

Y = Y_{init} + Δt*Y_dot;

Yaw = Yaw_{init} + Δt*Yaw_dot;

Yaw = Yaw_{init} + Δt*imu-data;

Meaning of Symbols

α (t) = steering angle (given in data)

θ (t) = Yaw (given in data)

d = wheel-base (distance between front wheel and rear wheels)

ω = angular velocity of front tires

r = radius of front wheels

Vt = linear velocity of front tires

Pose = $[\ X\ Y\ \theta\ ]^T$

Important Notes:

Since the yaw angle can be calculated with two separate sensory data, the pose tuple can be calculated with two different sensory data sets. I decided to use a complementary filter to combine the pose estimates calculated from the two separate data sources.

Readme for program.

My program consists of three files namely: poseEstimator.h; poseEstimator.cpp; and mainEstimator.cpp

The poseEstimator class has one constructor; four public functions; and 10 private member variables.

The constructor initializes the time variable and the encoder ticks variables because the kinematic equations require a change in these values and not the actual values. For example the intial time is 0, hence dt is 0. So dividing by zero in this case would be a problem.

The class poseEstimator has the function imuPose that returns the pose tuple calculated using the IMU data; the encoderPose that returns the pose tuple calculated using the encoder data; the function complementaryFilter that returns fused estimates of the pose calculated from both the imu data and the encoder data; and the function estimate() that returns the results of the complementary filter.

The mainEstimator illustrates the class used to calculate pose estimates

Things I took note of:

I noticed that the yaw rates calculated from the encoder were very erratic. Also, the file dataset.txt gives a very different ticks/revolution than that in the odom_estimator.pdf. The values for ticks per rev in these files were 35136 and 512 respectively. As a result, the complemetary filter I designed gave a lot of bias towards the pose estimates calculated from the imu data.

Moreover, certain parameters such as wheel-base; tire radii; and distance between the two rear tires were different in both documents.