86805: Software Architectures for Robotics
Localization system for a wheeled humanoid robot

Date: Wed 23:55, 01.02.16

Prof. F. Mastrogiovanni, PhD C. Recchiuto

Rabbia Asghar, BEng, Ernest Skrzypczyk, BSc

Contents

Rollo - Humanoid robot

Odometry

Odometry is currently the most widely used technique for determining the position of a mobile robot. It mainly involves use of various encoders, for example on wheels, as sensors to estimate the robot's position relative to a starting or previous location. Usually, it is used for real-time positioning in the between the periodic absolute position measurements, for example GPS (Global Positioning System) provides absolute position feedback, however it updates at $0.1 \div 1s$ interval, during which odometry could be used for localization. One of the major downsides of odometry is its sensitivity to errors. There are various error sources discussed in section Odometry errors on page 5 in detail. One significant source of error influencing the accuracy of odometry that is worth mentioning however, is the integration of velocity measurements over time to give position estimates.

First the odometry based motion model for the robot will be derived.

The model is derived based on the following important assumptions:

1. The robot is a rigid body
2. The model represents a differential drive robot
3. The wheels of the robot are perfect discs and have the same diameters
4. the wheels have no thickness
5. There is no slip in the wheels
6. Both wheels are turning in the forward direction

A differential drive robot runs straight when the linear speed of both the left and right wheel is same. If the speed of one wheel is greater than the other, the robot runs in an arc. This derivation can be divided in three distinct cases of robot motion:

The basic premise for the odometry model of the Rollo humanoid robot is presented in figures (**??**) and (**??**).

1. Clockwise direction
2. Counterclockwise direction
3. Straight line

Clockwise direction

First, the case is considered when the speed of left wheel is greater than the right, and the robot will run in clockwise direction. Both right and left wheel will rotate around the same center of a circle.

P_i represents the initial position of the robot, defined by the center of the line joining two wheels, while l represents wheel base, ergo distance between two wheels. S_L and S_R represent the distance travelled by left and right wheel respectively. $\Delta\Theta$ represents the angle of travel for both the wheels and r is the radius of travel from the center of robot. With encoder feedback from left and right wheel, S_L and S_R can simply be computed using equations (1) and (2)

$$S_L = \frac{n_L}{60} 2\pi r_L \tag{1}$$

$$S_R = \frac{n_R}{60} 2\pi r_R \tag{2}$$

where n_L and n_R are the revolutions per minute ([rpm]) of left and right wheel, and r_L and r_R are the radii of the 2 wheels.

S_L and S_R can be related to r and $\Delta\Theta$ using formulas (3) and (4)

$$S_L = (r + \frac{l}{2})\Delta\Theta \tag{3}$$

$$S_R = (r - \frac{l}{2})\Delta\Theta \tag{4}$$

The equations (3) and (4) on page ??an be solved simultaneously to compute $\Delta\Theta$.

$$\Delta\Theta = \frac{S_L - S_R}{l} \tag{5}$$

The length between initial position, P_i and final position, P_f can be approximated by taking average of S_L and S_R.

$$\Delta S = \frac{S_L + S_R}{2} \tag{6}$$

Final position of the robot and its orientation can then be derived using basic trigonometry and geometry relations as displayed in equation (7). In general this length [FIG] is approximated to dels S for a very small step of time.

$$P_f = \left[P_{ix} + \Delta S(\cos(\Theta_i - \tfrac{\Delta\Theta}{2})), \quad P_{iy} + \Delta S(\sin(\Theta_i - \tfrac{\Delta\Theta}{2})\right] \tag{7}$$

$$\Theta_\mathrm{f} = \Theta_\mathrm{i} - \Delta\Theta$$

where Θ_f and Θ_i are the final and initial orientation respectively.

Counterclockwise direction

Similar derivation of the robot can be derived for counterclockwise rotation, when the rpm of right wheel is higher than that of the left wheel.

$$\Delta\Theta = \frac{S_R - S_L}{l} \tag{8}$$

The relations for final position and rotation are as follows for this case:

$$P_f = \left[P_{ix} + \Delta S(\cos(\Theta_i + \tfrac{\Delta\Theta}{2})), \quad P_{iy} + \Delta S(\sin(\Theta_i + \tfrac{\Delta\Theta}{2})\right] \tag{9}$$

$$\Theta_\mathrm{f} = \Theta_\mathrm{i} + \Delta\Theta$$

where Θ_f and Θ_i are the final and initial orientation respectively.

Straight line motion

In case of straight line motion, the orientation of the robot stays the same. Thus, Θ is this case is 0. Also, S_L is the same as S_R.

$$P_f = \left[P_{ix} + S_L(\cos(\Theta_i)), \quad P_{iy} + S_L \sin(\Theta_i)\right] \tag{10}$$

$$\Theta_\mathrm{f} = \Theta_\mathrm{i}$$

Final set of equations

With the derivation of formulas understood for all three cases,for the ease of computation it is desirable to have the same set of equations in all cases.

The equations selected for this model are selected as follows.

$$\Delta\Theta = \frac{S_L - S_R}{l} \tag{11}$$

$$\Delta S = \frac{S_L + S_R}{2} \tag{12}$$

$$P_f = \left[P_{ix} + \Delta S(\cos(\Theta_i - \tfrac{\Delta\Theta}{2})), \quad P_{iy} + \Delta S(\sin(\Theta_i - \tfrac{\Delta\Theta}{2})\right] \tag{13}$$

$$\Theta_\mathrm{f} = \Theta_\mathrm{i} - \Delta\Theta$$

Adaption of odometry model for Rollo

Currently, the encoders feedback is unavailable in the robot and true odometry model can not be implemented. However, an attempt has been made to implement its modified version. The distance covered by right and left wheels is instead estimated from the control command.

Rollo was run in a straight line at different commands and its location feedback from the motion capture system was logged. For a given command, average speed of left and right wheel was determined from the logs. The logs were then analyzed using Matlab as follows.

1. The position x and y of the robot was plotted and line of best fit was plotted for the data set.
2. For the displacement of the robot, total distance traveled by the robot and change in orientation was determined using the line of best fit.
3. Using the timestamps in the logs, distance traveled and the change in orientation was determined for unit time.
4. The rpm of the wheels was determined using the relation in equations (1) through (6).

The above procedure is based on the assumption that there was no linear acceleration at a given velocity command and the rate of change of robot's orientation was constant.

The procedure was repeated for clockwise and counterclockwise rotation of the robot. For simplicity only the forward motion was modeled.

Table 1 shows the predetermined angular velocities of the wheel for given command velocities.

Velocity command left [%]	Velocity command right [%]	Left wheel angular velocity [rad/s]	Right wheel angular velocity [rad/s]
6	6	1.381	1.382
12	12	9.530	9.531
19	19	16.729	16.736
56	56	32.040	31.250
12	19	11.778	11.998
19	12	11.978	11.720
31	38	27.974	28.113
31	31	23.798	23.684

Table 1: Predetermined wheels angular velocity for given forward command.

Few challenges in the modeling of Rollo

1. Because of the mechanical difference in the two legs, it is almost impossible for the two wheels to run at same speed . Moreover, the two motors/wheels are powered using 2 different battery packs and are run in open loop. The difference in voltage level further results in different speed of the motor at the same pwm. This is a major source for deviation in behavior of robot's adjusted odometry model.
2. One of the Rollo's wheels is visibly skewed.
3. more?

Odometry errors

As briefly discussed above, odometry is very sensitive to errors. Results of an odometry based models can be improved by rapid and accurate data collection, equipment calibration, and processing.

The sources of odometry errors can be broadly divided into two categories.

1. Systematic errors
2. Non-systematic errors

Systematic errors

Systematic errors are caused by inherent properties of the system. They are usually caused by imperfections in the design and mechanical implementation of a mobile robot. On most smooth indoor surfaces, systematic errors play a much bigger role in odometry errors than non-systematic errors.Also, systematic errors accumulate constantly.

The major sources of the systematic error in an odometry model of a robot are unequal wheel diameters and the uncertainty about the effective wheelbase.

Rollo consists of 4 rubber tires, 2 on each side. Rubber tires help improve traction but they are difficult to manufacture to exactly the same diameter. Also, with time, rubber tires compress differently under asymmetric load distribution. The diameters are used in computing S_L and S_R as previously shown in equations (1) and (2).

Furthermore, the contact area between the tires of Rollo and the floor is very wide. This gives trouble in determining the wheelbase of the robot. Wheel base is used in computation of angle of travel as previously shown in equations (5).

Other Examples of sources of these errors include

1. Average of both wheel diameters differ from nominal diameter
2. Misalignment of wheels

3. Limited resolution during integration (encoder sampling rate, encoder measurement resolution)

Non-systematic errors

Non-systematic errors are caused by unknown and unpredictable changes in the system or in the environmental conditions. In mobile robots, they are usually caused by uneven floors. On rough surfaces with significant irregularities, non-systematic errors may be more dominant. On contrary to systematic errors, non-systematic errors do not accumulate with time.

Some examples of non-systematic errors are as follows:

1. Travel over uneven floors or unexpected objects on the floor
2. Wheel-slippage due to:
   (a) slippery floors
   (b) over-acceleration
   (c) fast turning (skidding)
   (d) non-point wheel contact with the floor

Measuring Odometry Errors

University of Michigan Benchmark test (UMBmark) procedure was planned to measure odometry errors and calibrate the system.This procedure was developed by Johann Borenstein and Liqiang Feng around 1995 in the University of Michigan's Advanced Technologies Lab. Details of this test can be found in the research paper.

As discussed in the previous section, there are 2 dominant sources of systematic errors in odometry: unequal wheel diameters and the uncertainty about the effective wheelbase.

Unfortunately, we are estimating S_L and S_R from the command given to the robot due to unavailability of encoder feedback. Even if we are able to determine the error in the relative size of the wheels and error due to the uncertainty in effective wheelbase, they can not be corrected or calibrated in the current model. However, the details of the procedure to determine effective wheelbase is described in this section.

An incorrect wheelbase value will result in incorrect computation of angle of travel, $\Delta\theta$ refer to (11).

This error, E_b is directly proportional to the effective wheelbase divided by the nominal wheelbase, l :

$$E_b = \frac{effectivewheelbase}{l} \tag{14}$$

After computing this error, we can replace the nominal wheelbase in equation (11) by effective wheelbase. so we can correct for this error by multiplying the nominal wheelbase (bn) by the error factor to give us the effective wheelbase (be):

$$\Delta\Theta = \frac{S_L - S_R}{E_b * l} \tag{15}$$

E_b is determined using UMBmark procedure. The summary of the UMBmark setup for Rollo is as follows:

1. At the beginning of the run, measure the absolute position and orientation of the robot using motion capture system and initialize to that position the starting point of the vehicle's odometry program.
2. Run the vehicle through a 2×2 m square path in cw direction, making sure to
   (a) stop after each 2 m straight leg;
   (b) make a total of four 90 -turns on the spot;

(c) run the vehicle slowly to avoid slippage.

3. Upon return to the starting area, measure the absolute position and orientation of the robot using motion capture system

4. Compare the absolute position to the robot's calculated position using equations (16), (17) and (18).

$$\epsilon x = x_a bs - x_c alc \tag{16}$$

$$\epsilon y = y_a bs - y_c alc \tag{17}$$

$$\epsilon \theta = \theta_a bs - \theta_c alc \tag{18}$$

5. Repeat steps 1-4 for four more times (i.e., a total of five runs).

6. Repeat steps 1-5 in ccw direction.

7. Compute coordinates of center of gravity for each cluster in cw and ccw direction using equations (19) and (20) where n is 5.

$$x_{c.g.,cw/ccw} = \frac{1}{n} \sum_{i=1}^{n} \epsilon x_{i,cw/ccw} \tag{19}$$

$$y_{c.g.,cw/ccw} = \frac{1}{n} \sum_{i=1}^{n} \epsilon y_{i,cw/ccw} \tag{20}$$

8. Compute E_b using equations (21),(22) and (23)

$$E_b = \frac{90^\circ}{90^\circ - \alpha} \tag{21}$$

where α can be computed as

$$\alpha = \frac{x_{c.g.,cw} + x_{c.g.,ccw}}{-4L} \frac{180^\circ}{\pi} \tag{22}$$

or

$$\alpha = \frac{y_{c.g.,cw} - y_{c.g.,ccw}}{-4L} \frac{180^\circ}{\pi} \tag{23}$$

and L represents the side of square pathi.i.e. 2m.

System and Measurements Model

System Model

In order to implement Kalman Filter, the previously described model must be first represented in state space representation.

We can define the location of the robot at instant k using state variables. This will include position in x and y coordinates and orientation.

$$x_k = \begin{bmatrix} x_{[x],k} \\ x_{[y],k} \\ x_{[\theta],k} \end{bmatrix} \tag{24}$$

The control input provided to robot is the speed of right and left wheel. This defines the distance covered by both the wheels in unit time. The relative displacement of the robot at instant k can be notated by d_k. Using equations (9),(7) and (10), relative displacement can be expressed in terms of ΔS and θ. Thus, control input u_k can be expressed as a function of relative displacement.

$$u_k = j(d_k) \tag{25}$$

$$u_k = \begin{bmatrix} u_{[\Delta S],k} \\ u_{[\Delta \theta],k} \end{bmatrix}$$

Given x_{k-1} and u_{k-1}, the next location of the robot, x_k can be computed.

$$x_k = f(x_{k-1}, u_{k-1}) = \begin{bmatrix} f_x(x_{k-1}, u_{k-1}) \\ f_y(x_{k-1}, u_{k-1}) \\ f_\theta(x_{k-1}, u_{k-1}) \end{bmatrix} \tag{26}$$

In the above derivation of the system model it was assumed that there are no noise sources. In the next section, we model the noise in the system.

System Model with noise

First, we will add noise to the relative displacement with the assumption that it can be modeled by a random noise vector q_k such that the noise is Gaussian distribution with zero mean,$\hat{q_k}$ and covariance matrix,U_k.

$$q_k \sim N(\hat{q_k}, U_k) \tag{27}$$

where

$$\hat{q_k} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

and

$$U_k = \mathrm{E}(q_k - \hat{q_k})(q_k - \hat{q_k})^T$$

$$U_k = \begin{bmatrix} \sigma^2_{q[\Delta S],k} & \sigma_{q[\Delta\theta],k}\sigma_{q[\Delta S],k} \\ \sigma_{q[\Delta\theta],k}\sigma_{q[\Delta S],k} & \sigma^2_{q[\Delta\theta],k} \end{bmatrix}$$

With the assumption that the noise sources are independent, the off-diagonal elements of the covariance matrix, U_k are equal to zero.The computation of variances $\sigma^2_{q[\Delta S],k}$ and $\sigma^2_{q[\theta],k}$ for the model is discussed in section.

The control input, or relative displacement can be expressed now as shown below.

$$u_k = j(d_k) + q_k \tag{28}$$

$$u_k = \begin{bmatrix} u_{[\Delta S],k} \\ u_{[\Delta \theta],k} \end{bmatrix} + \begin{bmatrix} q_{[\Delta S],k} \\ q_{[\Delta \theta],k} \end{bmatrix}$$

This makes u_k a random vector. Assuming, that $u_{[\Delta S],k}$ and $u_{[\theta],k}$ are deterministic, uncertainty in$u_k$equals the uncertainty in the noise term q_k.

The system noise can similarly be modeled by a random noise vector w_k such that the noise is Gaussian distribution with zero mean , $\hat{w_k}$ and covariance matrix, Q_k.

This noise source is not directly related to the relative displacement but is inherent to the system.???

$$w_k \sim N(\hat{w_k}, Q_k) \tag{29}$$

where

$$\hat{w_k} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

and

$$Q_k = \mathrm{E}(w_k - \hat{w_k})(w_k - \hat{w_k})^T$$

$$Q_k = \begin{bmatrix} \sigma^2_{w[x],k} & \sigma_{w[y],k}\sigma_{w[x],k} & \sigma_{w[\theta],k}\sigma_{w[x],k} \\ \sigma_{w[x],k}\sigma_{w[y],k} & \sigma^2_{w[y],k} & \sigma_{w[\theta],k}\sigma_{w[y],k} \\ \sigma_{w[x],k}\sigma_{w[\theta],k} & \sigma_{w[\theta],k}\sigma_{w[x],k} & \sigma^2_{w[\theta],k} \end{bmatrix}$$

With the assumption that the noise sources are independent, the off-diagonal elements of the covariance matrix, Q_k are equal to zero.The computation of variances $\sigma^2_{w[x],k}$, $\sigma^2_{w[y],k}$ and $\sigma^2_{w[\theta],k}$ for the model is discussed in section.

The system can be expressed now as shown below.

$$x_k = f(x_{k-1}, u_{k-1}) + w_k \tag{30}$$

$$x_k = \begin{bmatrix} f_x(x_{k-1}, u_{k-1}) \\ f_y(x_{k-1}, u_{k-1}) \\ f_\theta(x_{k-1}, u_{k-1}) \end{bmatrix} + \begin{bmatrix} w_{[x],k-1}) \\ w_{[y],k-1}) \\ w_{[\theta],k-1}) \end{bmatrix} \tag{31}$$

w_k consists of noise sources that are not directly related to u_k. Now, x_k is a random vector and with every time step the system noise increases the variance. Thus, the variance of the location grows with every time step.

We have proposed the System Model so that we can implement Kalman Filter on it. Now, we need to prepare the Measurement Model for it.

Measurement Model with noise

Starting with the assumption that there is no noise in the measurement, z_k, it is simply a vector containing for each state variable a variable that takes on the value of the corresponding state variable. The measurement vector, z_k is

$$z_k = \begin{bmatrix} z_{[x],k} \\ z_{[y],k} \\ z_{[\theta],k} \end{bmatrix} \tag{32}$$

In our system, we use the motion capture system in the lab to localize the robot. The motion capture system acts as absolute sensor and provides us directly the position of the robot in x and y coordinates and its orientation. Thus, z_k in this case is simply expressed as (33)

$$z_k = \begin{bmatrix} x_{[x],k} \\ x_{[y],k} \\ x_{[\theta],k} \end{bmatrix}$$

Now, we can add noise to the measurement model. We assume that the noise in the odometry can be modeled by a random noise vector v_k such that the noise is Gaussian distribution with zero mean,$\hat{v_k}$ and covariance matrix,R_k .

$$v_k \sim N(\hat{v_k}, R_k) \tag{33}$$

where

$$\hat{v_k} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

and

$$R_k = \mathrm{E}(v_k - \hat{v_k})(v_k - \hat{v_k})^T$$

$$R_k = \begin{bmatrix} \sigma^2_{v[x],k} & \sigma_{v[y],k}\sigma_{v[x],k} & \sigma_{v[\theta],k}\sigma_{v[x],k} \\ \sigma_{v[x],k}\sigma_{v[y],k} & \sigma^2_{v[y],k} & \sigma_{v[\theta],k}\sigma_{v[y],k} \\ \sigma_{v[x],k}\sigma_{v[\theta],k} & \sigma_{v[\theta],k}\sigma_{v[x],k} & \sigma^2_{v[\theta],k} \end{bmatrix}$$

With the assumption that the noise sources are independent, the off-diagonal elements of the covariance matrix, R_k are equal to zero.

The measurement model can be expressed now as shown below.

$$z_k = x_k + v_k \tag{34}$$

$$z_k = \begin{bmatrix} x_{[x],k} \\ x_{[y],k} \\ x_{[\theta],k} \end{bmatrix} + \begin{bmatrix} v_{[x],k-1}) \\ v_{[y],k-1}) \\ v_{[\theta],k-1}) \end{bmatrix}$$

Since the measurement noise v_k is a Gaussian vector, this makes z_k a random vector.

Kalman Filter

Kalman filter is an algorithm that uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more precise than those based on a single measurement alone.(WIKI REFERENCE). Provide an actual link, and not only to the wikipage, to the actual source

A physical system is driven by a set of external inputs or controls and its outputs are evaluated by sensors, such that the knowledge on the system's behavior is solely given by the inputs and the observed outputs. For example, in our case, the physical system is a mobile robot and we are given the problem of localizing it. The control or the input is defined by the velocity command given to the right and left wheel which determines the relative displacement of the robot. The observed output is its position and orientation measured from Optitack motion capture system. The observations convey the errors and uncertainties in the process, namely the sensor noise and the system errors. Based on the available information (control inputs and observations), Kalman filter computes an estimate of the system's state by minimizing the variance of the estimation error. From a theoretical standpoint, the main assumption of the Kalman filter is that the underlying system is a linear dynamical system and that all error terms and measurements have a Gaussian distribution.

Kalman Filtering is a recursive analytical technique involving 2 main steps: prediction and innovation. Prediction is also known as 'Time update' and refers to prediction of state variables using previous state and control input. This step does not involve using measurement. Innovation which is also known as 'Correction' or 'Measurement update' refers to estimation of state variable using measurement.

Kalman filter is popularly used in the localization of the robot because of its efficiency and accuracy. Some of its characteristics are as follows:

1. Computationally efficient, update filter when adding new measurements to the existing data set
2. Very simple to implement
3. Uncertainty estimates are provided as part of the filter
4. Recursive Algorithm, that means it can be implemented in real time using only the present input measurements and the previously calculated state.

All members of kalman family of filters e.g. extended kalman filter and unscented kalman filter comply with a structured sequence of six steps per iteration The six steps are listed as follows.

1. **State estimate time update:** An updated state prediction $\hat{x}_{k|k-1}$ is made, based on a priori information and the system model.
2. **Error covariance time update:** The second step is to determine the predicted state-estimate error covariance matrix, $\Sigma_{k|k-1}$ based on a priori information and the system model
3. **Estimate system output:** The third step is to estimate the system's output, $\hat{z}_k$ corresponding to the timestamp of the most recently received measurement using present a priori information.
4. **Estimator gain matrix:** The fourth step is to compute the estimator gain matrix,H_k.
5. **State estimate measurement update:** The fifth step is to compute the a posteriori state estimate, $\hat{x}_{k|k}$ by updating the a priori estimate using the estimator gain and the output prediction error.
6. **Error covariance measurement update:** The final step computes the a posteriori error covariance matrix, $\Sigma_{k|k}$.

Kalman Filter Equations For Linear System

In order to implement Kalman Filter, we first represent our linear system as follows.

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k + w_k \\ z_k &= Cx_k + v_k \end{aligned} \tag{35}$$

where $w_k \sim N(0, Q_k)$ and $v_k \sim N(0, R_k)$.

We will proceed with the assumption that A, B, C, Q and R are constant for our system. Given initial state estimate, $\hat{x}_{0|0}$ and initial state covariance matrix, $\Sigma_{0|0}$ the Kalman Filter can be computed using following set of equations.

Prediction or Time update:

$$\begin{aligned} \hat{x}_{k|k-1} &= A\hat{x}_{k-1|k-1} + Bu_{k-1} \\ \Sigma_{k|k-1} &= A\Sigma_{k-1|k-1}A^T + Q \end{aligned} \tag{36}$$

Innovation or Measurement update:

$$\begin{aligned} H_k &= \Sigma_{k|k-1}C^T(C\Sigma_{k|k-1}C^T + R)^{-1} \\ \hat{x}_{k|k} &= \hat{x}_{k|k-1} + H_k[z_k - C\hat{x}_{k|k-1}] \\ \Sigma_{k|k} &= [I - H_kC]\Sigma_{k|k-1} \end{aligned} \tag{37}$$

Kalman Filter Equations For Non-Linear System

Now we consider a nonlinear extension to the Kalman filter, **Extended Kalman Filters (EKF)**. The EKF implements a Kalman filter for a non-linear system dynamics that results from the linearization of the original non-linear filter dynamics around the previous state estimates.

A non linear system can be represented as follows.

$$\begin{aligned} x_k &= f(x_{k-1}, u_{k-1}) + w_{k-1} \\ z_k &= h(x_k) + v_{k-1} \end{aligned} \tag{38}$$

where $w_k \sim N(0, Q_k)$ and $v_k \sim N(0, R_k)$.

We will proceed with the assumption that f(), h(), Q and R are constant for our non linear system. f() and h() are linearized about the prior best estimates of the states at each instant of time by finite difference method as,in order to compute covariance. Given initial state estimate, $\hat{x}_{0|0}$ and initial state covariance matrix, $\Sigma_{0|0}$ the Extended Filter can be computed using following set of equations.

Prediction or Time update:

$$\begin{aligned} \hat{x}_{k|k-1} &= f\left(\hat{x}_{k-1|k-1}, u_{k-1}\right) \\ \Sigma_{k|k-1} &= J_f\Sigma_{k-1|k-1}J_f^T + Q \end{aligned} \tag{39}$$

Innovation or Measurement update:

$$\begin{aligned} H_k &= \Sigma_{k|k-1} J_h^T (J_h \Sigma_{k|k-1} J_h^T + R)^{-1} \\ \hat{x}_{k|k} &= \hat{x}_{k|k-1} + H_k[z_k - h\left(\hat{x}_{k|k-1}\right)] \\ \Sigma_{k|k} &= [I - H_k J_h]\Sigma_{k|k-1} \end{aligned} \tag{40}$$

where J_f and J_h are the jacobians.

J_f is the jacobian matrix with the partial derivatives of the system function f($\cdot$, $\cdot$) with respect to the state x, evaluated at the last state estimate $\hat{x}_{k-1|k-1}$ and control input u_{k-1}.

J_h is the jacobian matrix with partial derivatives of the measurement function h(.) with respect to the state x, evaluated at the prior state estimate $\hat{x}_{k|k-1}$.

Extended Kalman Filter Equations For Odometry

After deriving model of our system for odometry and understanding the extended kalman Filter in previous section, we will now prepare the extended kalman filter equations for odometry.

$$\begin{bmatrix} x_{[x],k} \\ x_{[y],k} \\ x_{[\theta],k} \end{bmatrix} = \begin{bmatrix} f_x(x_{k-1}, u_{k-1}) \\ f_y(x_{k-1}, u_{k-1}) \\ f_\theta(x_{k-1}, u_{k-1}) \end{bmatrix} + \begin{bmatrix} w_{[x],k-1}) \\ w_{[y],k-1}) \\ w_{[\theta],k-1}) \end{bmatrix} \tag{41}$$

$$\begin{bmatrix} x_{[x],k} \\ x_{[y],k} \\ x_{[\theta],k} \end{bmatrix} = \begin{bmatrix} x_{[x],k-1} + u_{[\Delta]S,k-1} \cdot \cos(x_{[\Theta],k-1} - \frac{u_{[\Delta\Theta],k-1}}{2}) \\ x_{[y],k-1} + u_{[\Delta]S,k-1} \cdot \sin(x_{[\Theta],k-1} - \frac{u_{[\Delta\Theta],k-1}}{2}) \\ x_{[\theta],k-1} - u_{[\Delta\Theta],k-1}) \end{bmatrix} + \begin{bmatrix} w_{[x],k-1}) \\ w_{[y],k-1}) \\ w_{[\theta],k-1}) \end{bmatrix} \tag{42}$$

Prediction or Time update:

$$\begin{bmatrix} \hat{x}_{[x],k|k-1} \\ \hat{x}_{[y],k|k-1} \\ \hat{x}_{[\theta],k|k-1} \end{bmatrix} = \begin{bmatrix} \hat{x}_{[x],k-1|k-1} + u_{[\Delta]S,k-1} \cdot \cos(\hat{x}_{[\Theta],k-1|k-1} - \frac{u_{[\Delta\Theta],k-1}}{2}) \\ \hat{x}_{[y],k-1} + u_{[\Delta]S,k-1} \cdot \sin(\hat{x}_{[\Theta],k-1|k-1} - \frac{u_{[\Delta\Theta],k-1}}{2}) \\ \hat{x}_{[\theta],k-1|k-1} - u_{[\Delta\Theta],k-1}) \end{bmatrix} \tag{43}$$

$$\begin{aligned} J_{f,k} &= \frac{\partial f(x)}{\partial x}\Big|_{x=\hat{x}_{k-1|k-1}, u=u_{k-1}} \\ &= \begin{bmatrix} \frac{\partial f_x}{\partial x_{[x]}} & \frac{\partial f_x}{\partial x_{[y]}} & \frac{\partial f_x}{\partial x_{[\theta]}} \\ \frac{\partial f_y}{\partial x_{[x]}} & \frac{\partial f_y}{\partial x_{[y]}} & \frac{\partial f_y}{\partial x_{[\theta]}} \\ \frac{\partial f_\theta}{\partial x_{[x]}} & \frac{\partial f_\theta}{\partial x_{[y]}} & \frac{\partial f_\theta}{\partial x_{[\theta]}} \end{bmatrix}_{x=\hat{x}_{k-1|k-1}, u=u_{k-1}} \\ &= \begin{bmatrix} 1 & 0 & -u_{[\Delta]S} \cdot \sin(x_{[\Theta]} - \frac{u_{[\Delta\Theta]}}{2}) \\ 0 & 1 & +u_{[\Delta]S} \cdot \cos(x_{[\Theta]} - \frac{u_{[\Delta\Theta]}}{2}) \\ 0 & 0 & 1 \end{bmatrix}_{x=\hat{x}_{k-1|k-1}, u=u_{k-1}} \end{aligned} \tag{44}$$

$$\Sigma_{[x],k|k-1} = J_f \Sigma_{[x],k-1|k-1} J_f^T + Q \tag{45}$$

Innovation or Measurement update:

$$\begin{aligned} J_{h,k} &= \frac{\partial h(x)}{\partial x}\Big|_{x=\hat{x}_{k|k-1}} \\ &= \begin{bmatrix} \frac{\partial h_x}{\partial x_{[x]}} & \frac{\partial h_x}{\partial x_{[y]}} & \frac{\partial h_x}{\partial x_{[\theta]}} \\ \frac{\partial h_y}{\partial x_{[x]}} & \frac{\partial h_y}{\partial x_{[y]}} & \frac{\partial h_y}{\partial x_{[\theta]}} \\ \frac{\partial h_\theta}{\partial x_{[x]}} & \frac{\partial h_\theta}{\partial x_{[y]}} & \frac{\partial h_\theta}{\partial x_{[\theta]}} \end{bmatrix}_{x=\hat{x}_{k|k-1}, u=u_{k-1}} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}_{x=\hat{x}_{k|k-1}, u=u_{k-1}} \end{aligned} \tag{46}$$

Using the above derived jacobian matrix (46) for computation of Kalman gain matrix, H_k, a posteriori state estimate and a posteriori error covariance matrix, we get following equations.

$$\begin{aligned} H_k &= \Sigma_{k|k-1}(\Sigma_{k|k-1} + R)^{-1} \\ \hat{x}_{k|k} &= \hat{x}_{k|k-1} + H_k[z_k - \hat{x}_{k|k-1}] \\ \Sigma_{k|k} &= [I - H_k J_h]\Sigma_{k|k-1} \end{aligned} \tag{47}$$

Practical Implementation of EKF

Proceeding with the EKF model for odometry, we need initializing of the model before we can practically implement it. We need to initialize following components.

1. Initial state estimate, $\hat{x}_{0|0}$
2. Initial state covariance matrix, $\Sigma_{0|0}$
3. Process covariance matrix, Q
4. Measurement covariance matrix, R

Initial state estimate, $\hat{x}_{0|0}$

This is the initial estimate of the state vector required for the first iteration of the EKF. In our system, Optitrack motion capture system is used for measurement. Optitrack provides ground truth and positional accuracy of up to sub-millimeter. For our testing, we simply used the measurement sensor reading to determine $\hat{x}_{0|0}$.

Initial state covariance matrix, $\Sigma_{0|0}$

Initial state covariance matrix is based on the initialization error of the state. If the initial state estimate is very close to the actual state, this matrix will contain very small values. With the assumption that noise sources for different state elements are independent, the off diagonal elements of the covariance matrix can be taken to be zero.

Usually if this matrix is unknown, we can use identity matrix since this matrix is updated in every iteration of EKF. For a stable EKF, this matrix should be converging. A better initial estimate will offer faster convergence of the Kalman filter.

For our testing, since we determine the initial state from the measurement sensor, initial state covariance matrix can be taken as zero. However, since we are simulating a sensor not as accurate as motion capture, the matrix is taken as identity.

Process covariance matrix, Q

This is the error covariance matrix of the process and gives an estimate of uncertainty in the state equations. The uncertainty in the process could be in result of various sources of error include modeling errors, odometry errors, discretization, approximations involved in the derivation of the model.

In order to get a feel of error in the model, we computed error based on odometry model from log files.

For testing, we tried different range of values for Q to understand the behaviour of the EKF.

Measurement covariance matrix, R

This is the error covariance matrix of measurement sensor and gives a measure of how uncertain is measurement. As discussed earlier, Optitrack motion capture system is used for measurement and it provides ground truth. For the case when the measurement is very accurate, matrix R will carry very small variances and can be zero. However, in order to simulate EKF for a system whose measurement sensor is not as accurate, we tried different range of values for covariance matrix.

Divergence of state covariance matrix, $\Sigma_{k|k}$

Proceeding with initialization of EKF as discussed above, it was observed that the filter behaved well for some time after initialization. However, after some time the state covariance matrix started diverging very fast, resulting in failure of the Filter.

It was understood that a possible reason behind this could be lack of input excitation that results in growing values of state covariance matrix and large spread of eigen values. In order to cater this issue, different techniques can be used to stabilize estimation and prevent windup of state estimation and covariance matrix.

We resolved this issue by performing Cholesky decompostion in the computation of Kalman gain matrix. Cholesky factorization is a decomposition of a Hermitian, positive-definite matrix into the product of a lower triangular matrix (or an upper triangular matrix)and its conjugate transpose. Cholesky decomposition offers numerical stability to the system.

This resulted in good behavior of the filter, however, it slowed down the performance of EKF. With Cholesky decomposition, the innovation update of EKF can be written as shown in (48)

$$\begin{aligned} U &= CholsekyDecomposition(J_h \Sigma_{k|k-1} J_h^T + R) \\ \hat{x}_{k|k} &= \hat{x}_{k|k-1} + [\Sigma_{k|k-1} U^{-1}] U^{T^{-1}} [z_k - h\left(\hat{x}_{k|k-1}\right)] \\ \Sigma_{k|k} &= \Sigma_{k|k-1} - [\Sigma_{k|k-1} U^{-1}][\Sigma_{k|k-1} U^{-1}]^T \end{aligned} \tag{48}$$

where $CholsekyDecomposition(X)$ produces an upper triangular matrix U from the diagonal and upper triangle of matrix X, satisfying the equation $U^T U = X$.

Simulation and Testing

Simulation of Extended Kalman Filter

EKF algorithm was first simulated in MATLAB using logs from robot's test runs. Measurement logs were collected for a simple robot's straight line test runs and initial value of measurement was set as the initial

state estimate. EKF update was performed using this initial state estimate, control input for the given log and measurement for the log.

Figure 1 shows the results of EKF for a log file. x state[1] and x state[2] represent position of robot in x and y coordinates in meters respectively. x state[3] represents orientation of the robot in radians. The blue lines shows the state estimates updates with EKF while the red lines represent the measurement which is the true state.

It can be observed from the plots that the state estimates are able to correct itself to measurement and generate a smooth transition.

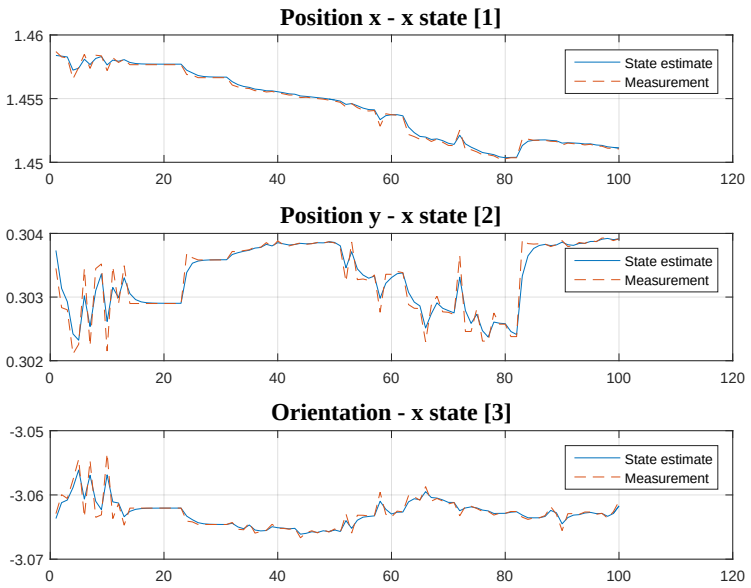

Figure 1: Results of EKF simulation a log file.

Figure 2 shows the results of EKF update for longer iterations to verify the stability of EKF. If original EKF equations are directly used with the system, the state estimate and state covariance matrix starts to

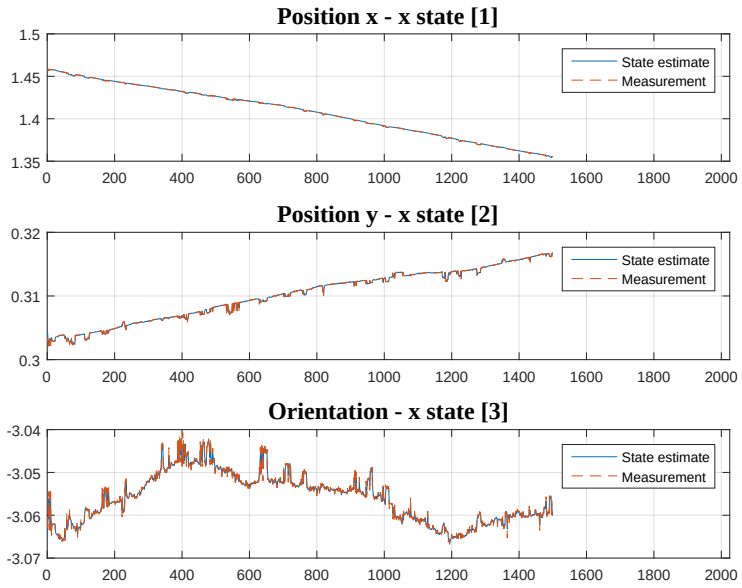

Figure 2: Results of EKF simulation a log file.

diverge after sometime, as discussed in previous section. This figure shows result of performing Cholesky decomposition for measurement update.

Testing of Extended Kalman Filter

Test Setup

In order to conduct the tests both battery packs were fully charged and five markers were placed on Rollo at different points so it could be reliably tracked by motion capture system.

Motion capture system was calibrated so that the x-axis and y-axis for localization were defined according to Cartesian coordinate system. The Robot was placed in the arena with its orientation aligned to x-axis and

then defined as a rigid body in the motion capture system. This ensured that the orientation of the robot was 0 radians when it was aligned to x-axis similar to the odometry model developed.

The complete software that received localization from the motion capture system, controlled Rollo, process data to implement EKF and visualization was developed in ROS framework. Rollo was manually controlled using the keyboard and communication with Rollo was done at 10Hz. The measurement node was updated 25 Hz. Hence, the EKF update was also performed at a rate of approximately 25Hz.

Unfortunately, the encoder feedback was unavailable for the test but the communication node was pre-programmed to send certain angular velocities for left at a given command. The complete range was not covered, only the command values shown in the table 1 were tested.

In the start of each test run, the robot position and orientation read from the motion capture system was taken as the initial state estimate. Initial state covariance matrix was taken as identity matrix.

State prediction using the odometry based model alone was also studied for comparison with EKF based model. However, in the absence of encoder, this model was very crude. Moreover, the testing floor was uneven and had visible irregularities, which introduced non systematic errors that are not modeled in the odometry.

Initial state for the odometry model was taken the same as for the EKF based model. The logs were collected sometime after the test was conducted and analyzed. This was due to lag in the system.However, The videos of online visualization for the tests were recorded from the very start of the run.

Testing Variables

We tested Rollo with different process covariance matrix,Q and measurement covariance matrix to analyze EKF. We define the two matrices here again. (For details, sections and can be referred again.)

For process noise covariance matrix, it was assumed that the noise sources were independent for the three states (position in x and y coordinates and the orientation) and the noise variances for all three states were taken equal and constant. Assigning the standard deviation for all states as q, the Q matrix can be written as shown in (49).

$$Q_k = \begin{bmatrix} q^2 & 0 & 0 \\ 0 & q^2 & 0 \\ 0 & 0 & q^2 \end{bmatrix} \tag{49}$$

For measurement noise covariance matrix, it was assumed that the noise sources were independent for the three states (position in x and y coordinates and the orientation) and the noise variances for all three states were taken equal and constant. Assigning the standard deviation for the all states as r, the R matrix can be written as shown in (50).

$$R_k = \begin{bmatrix} r^2 & 0 & 0 \\ 0 & r^2 & 0 \\ 0 & 0 & r^2 \end{bmatrix} \tag{50}$$

Following combinations of **q** and **r** were tested.

1. $q = 0.1$ and $r = 0.1$
2. $q = 0.1$ and $r = 1.0$
3. $q = 0.1$ and $r = 10.0$
4. $q = 1.0$ and $r = 10.0$
5. $q = 10.0$ and $r = 10.0$

6. $q = 100.0$ and $r = 10.0$
7. $q = 8.0$ and $r = 32.0$

The EKF update rate was 10Hz for all test runs. For every test the initial state estimate was reassigned and the EKF node was restarted.

Test Results and analysis

During on line testing the localization was studied using the visualization node. A snapshot of the real time visualization is shown in figure.

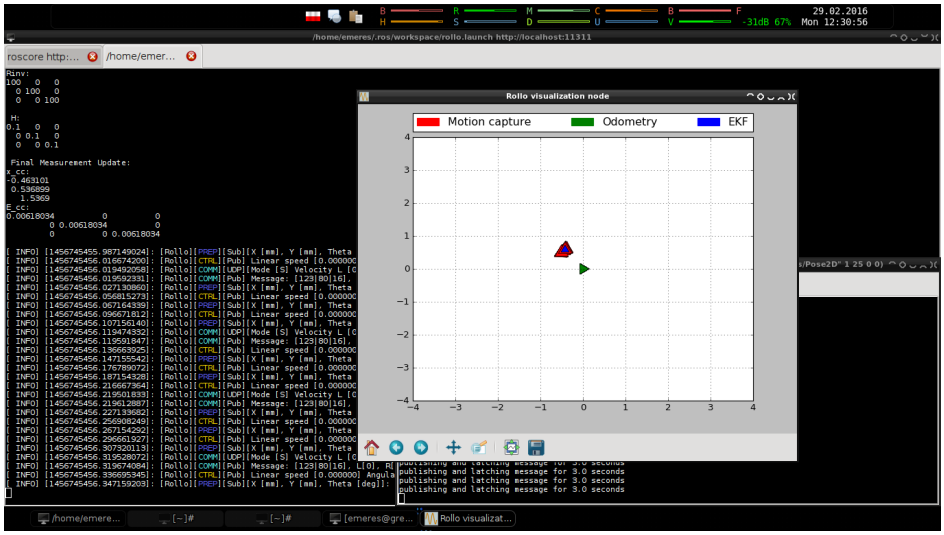

Figure 3: Real time visualization: localization of the robot based on EKF, odometry alone and the measurement.

Logs were generated for off line and in depth analysis of the system. For every test run, two different plots were generated. One showing position of the robot computed according to odometry, EKF and motion capture system. Second plot displayed position and orientation of the robot with respect to time. Some plots of the test runs are shown in this section.

Initial tests were conducted for $q = 0.1$ and $r = 0.1$. Figures 4 and 5 show a straight line run for 19% velocity command. It can be observed from the plot that the state estimate is almost superimposed on measurement line. This shows EKF is able to estimate the state very well. Odometry results have however drifted far from actual position of the robot with time. Since this was only crudely modeled without encoders feedback, no efforts were made to improve it.

The initial values of q and r involved in EKF yielded respectable performance. Further tests were performed by adding more noise to the measurement model by taking a larger r.

By taking $q = 0.1$ and $r = 1.0$, the tracking ability of EKF was reduced. Figures 6 and 7 show a straight line run for 12% velocity command. It can be observed from the plots that the state estimate is close to the measurement line but is unable to reach it. It remains at a varying offset to the measurement.

We added further noise to the measurement model by taking $q = 0.1$ and keeping $r = 10.0$. Figures 8 and 9 show plots of a counter clockwise rotation for 12% left velocity command and 19% right velocity command. State estimate is visibly drifted from the true position. With the increase in measurement noise, measurement loses its wightage in EKF. Further tests were conducted with keeping r constant at 10.

Figures 10 and 11 show another counter clockwise rotation at 12% left velocity command and 19% right velocity command for $q = 1.0$ and $r = 10.0$. Now, the EKF response is widely improved. By increasing the

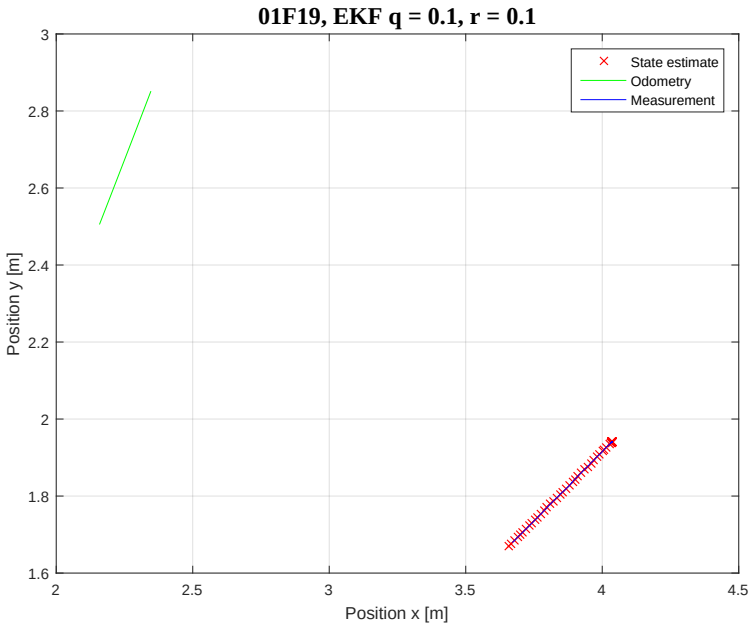

Figure 4: Test 1, velocity command 19% for both left and right wheels. q = 0.1, r = 0.1.

value of q, we are increasing the uncertainty in our process model. In comparison to the test anaylzed above, the measurement has gained its weightage here. However, it can be seen that there is still an error between the EKF based state estimate and the measurement.

We further added more noise to the process model by increasing q. Figures 12 and 13 show plots of a counter clockwise rotation at 12% left velocity command and 19% right velocity command for $q = 10.0$ and $r = 10.0$. It can be observed that the EKF is able to track the localisation of the robot very well.The performance is comparable to results seen in figure 1 when the values for q and r are the same as 0.1 and 0.1. It can be understood that it is basically the relative ratio between Q and R matrices that determine the performance of EKF not the values of Q and R alone.

In figure 14 and 15, we plot a clockwise rotation at 19% left velocity command and 12% right velocity

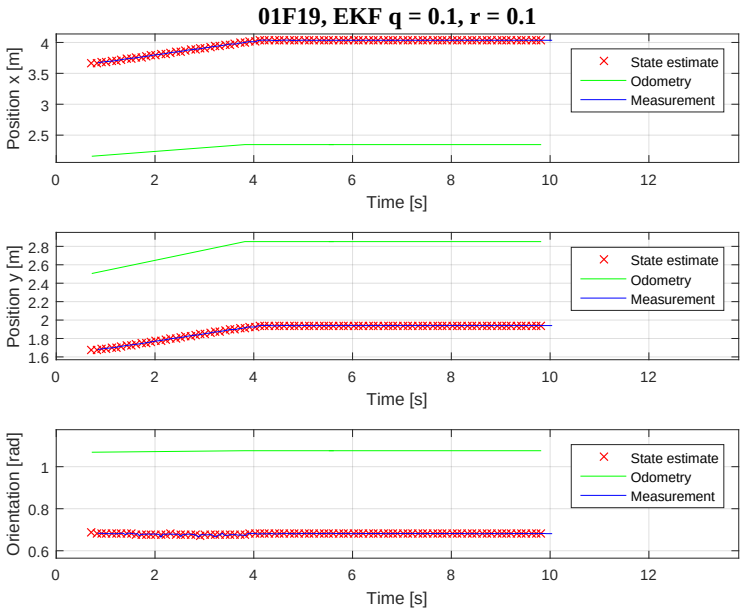

Figure 5: Test 1, velocity command 19% for both left and right wheels. q = 0.1, r = 0.1. Figure shows all three states with respect to time.

command for $q = 100.0$ and $r = 10.0$. By increasing the value of q, we are adding more noise to the process model, the measurement gains weightage in the EKF model. As can be observed in the figure, the EKF is able to localize the robot very well. It must be understood that this is because our measurement sensor provides very accurate localization of the robot.

Final test was performed with values $q = 8.0$ and $r = 32.0$. Figures 16 and 17 show plots of a counter clockwise rotation at 12% left velocity command and 19% right velocity command. Since r is greater than q, the measurement model has a lower weightage as compared to the process model. It can be observed from the plots that the EKF based state estimate is unable to localize the robot well and state estimate is visibly drifted from the true position especially when running a curve.

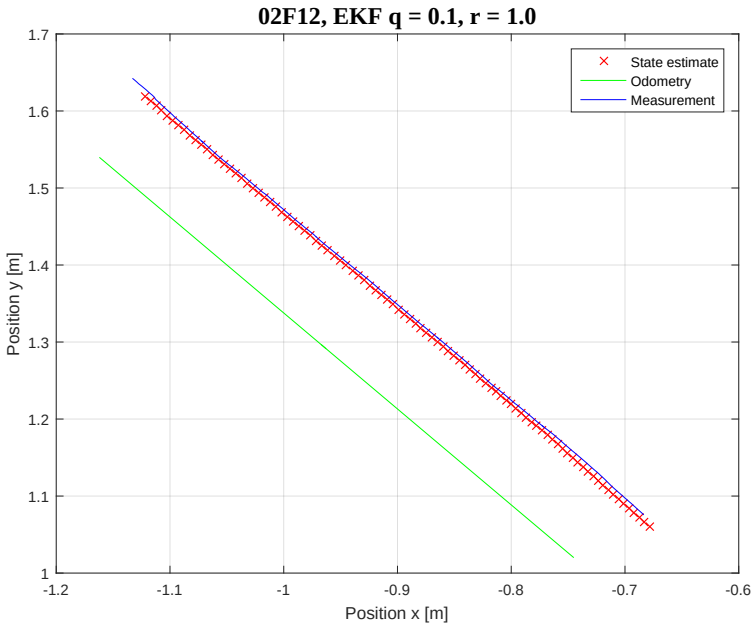

Figure 6: Test 2, velocity command 12% for both left and right wheels. q = 0.1, r = 1.0.

Conclusions

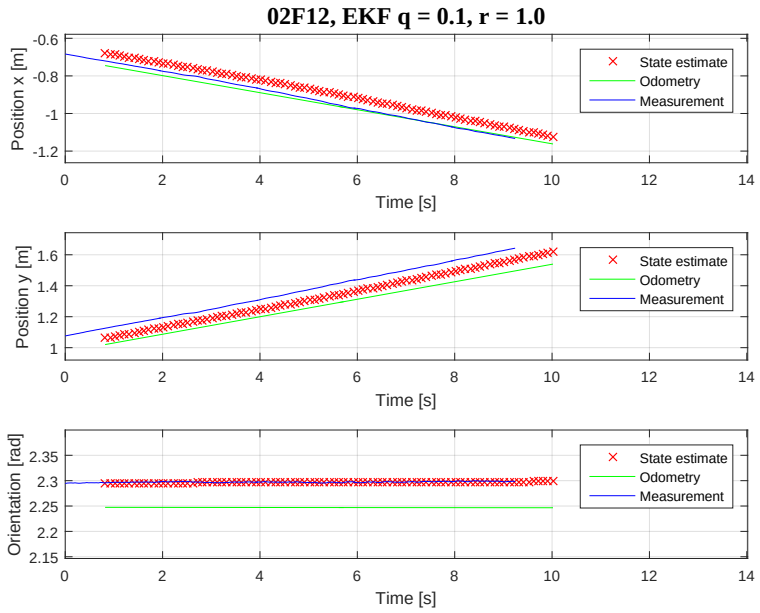

Figure 7: Test 2, velocity command 12% for both left and right wheels. q = 0.1, r = 1.0. Figure shows all three states with respect to time.

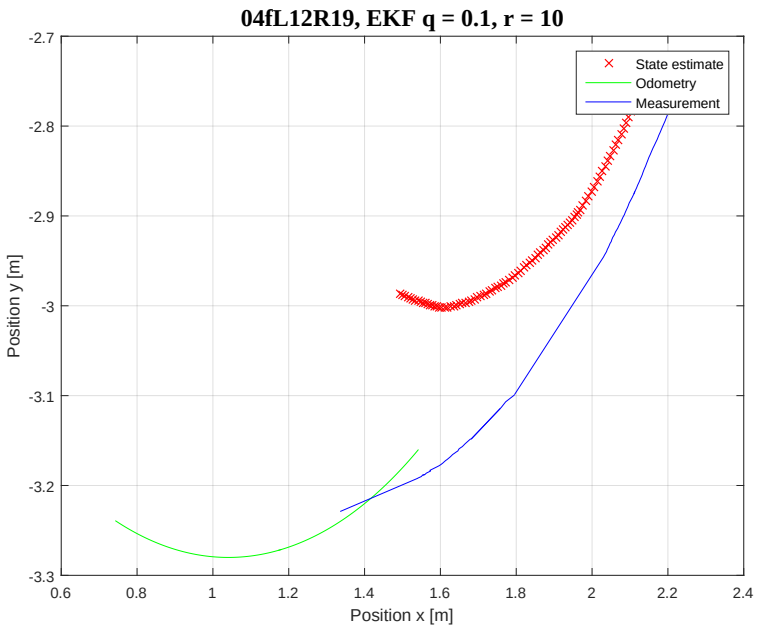

Figure 8: Test 3, velocity command 12% for left wheel and 19% for right wheel. q = 0.1, r = 10.

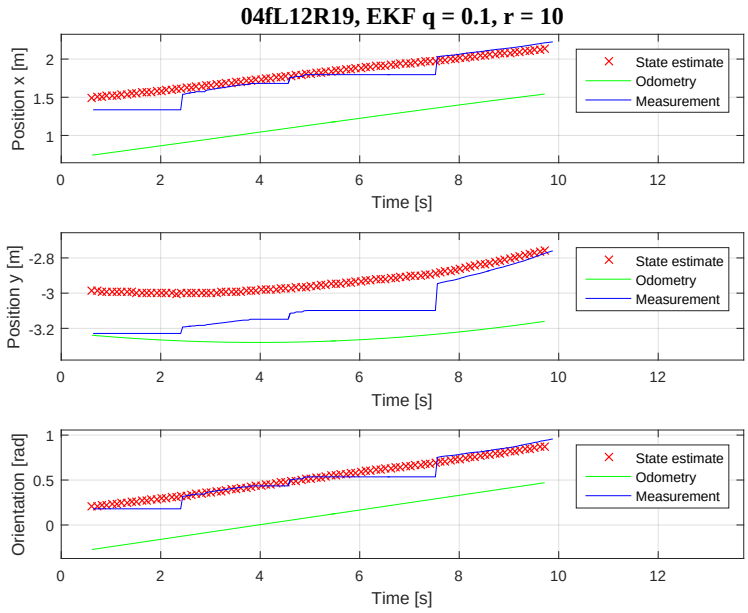

Figure 9: Test 3, velocity command 12% for left wheel and 19% for right wheel. q = 0.1, r = 10. Figure shows all three states with respect to time.

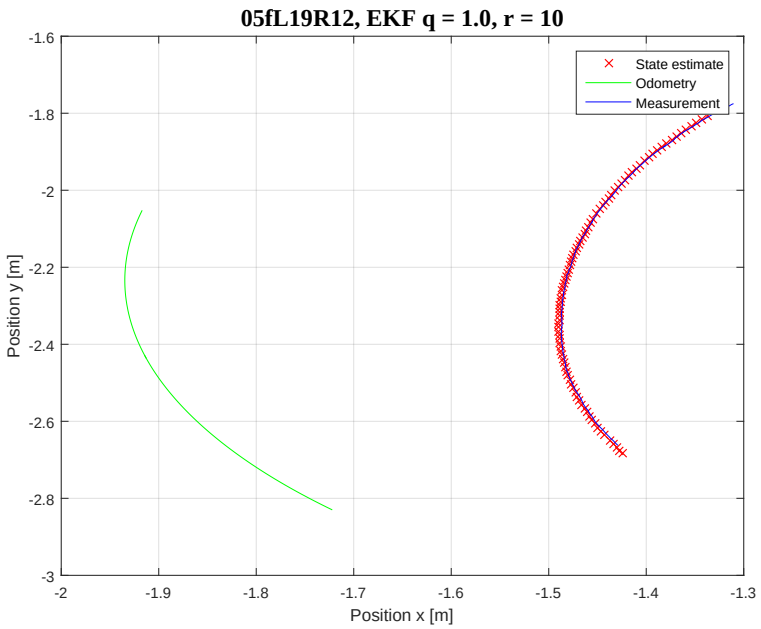

Figure 10: Test 4, velocity command 19% for left wheel and 12% for right wheel. q = 1.0, r = 10.

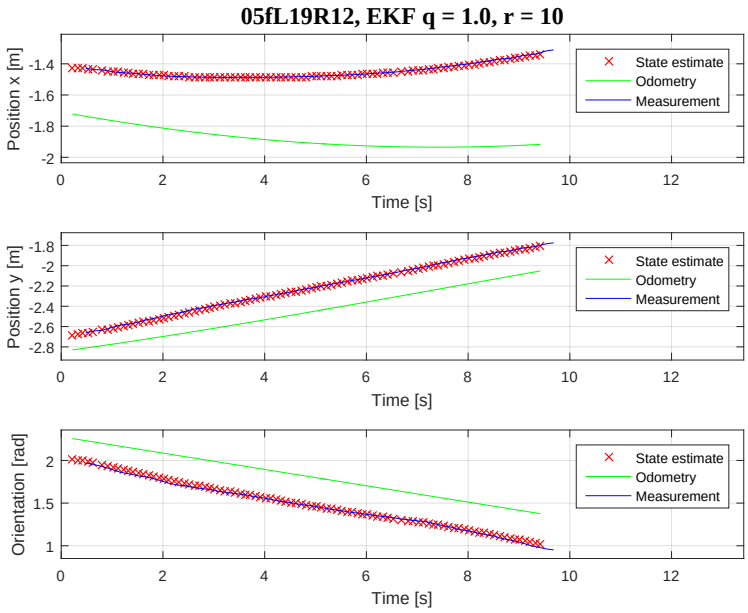

Figure 11: Test 4, velocity command 19% for left wheel and 12% for right wheel. q = 1.0, r = 10. Figure shows all three states with respect to time.

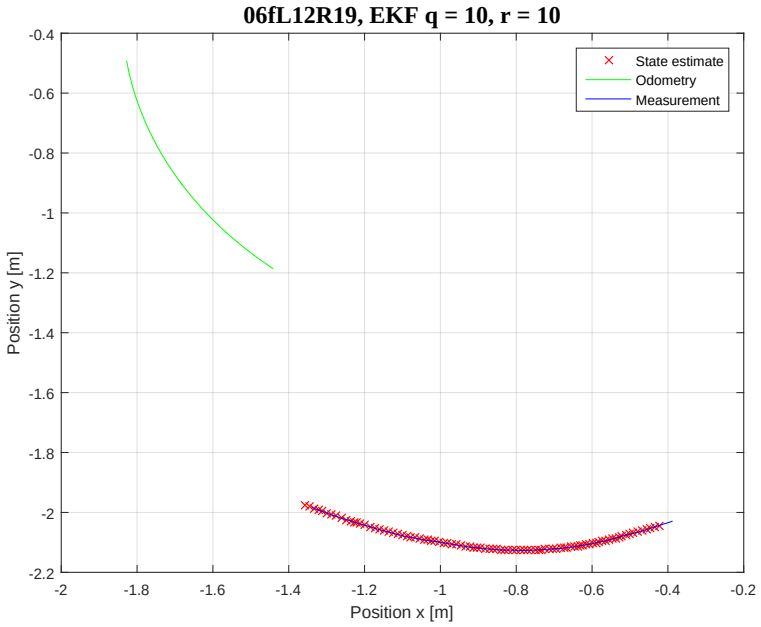

Figure 12: Test 5, velocity command 12% for left wheel and 19% for right wheel. q = 10, r = 10.

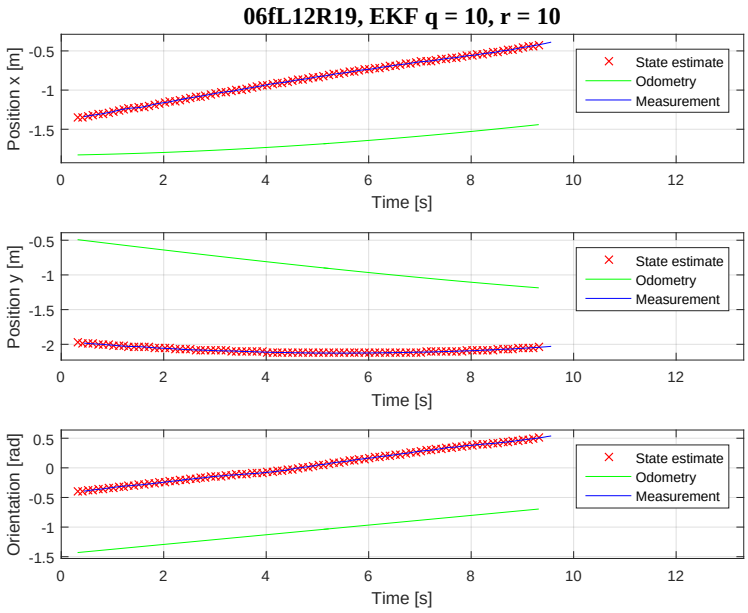

Figure 13: Test 5, velocity command 12% for left wheel and 19% for right wheel. q = 10, r = 10. Figure shows all three states with respect to time.

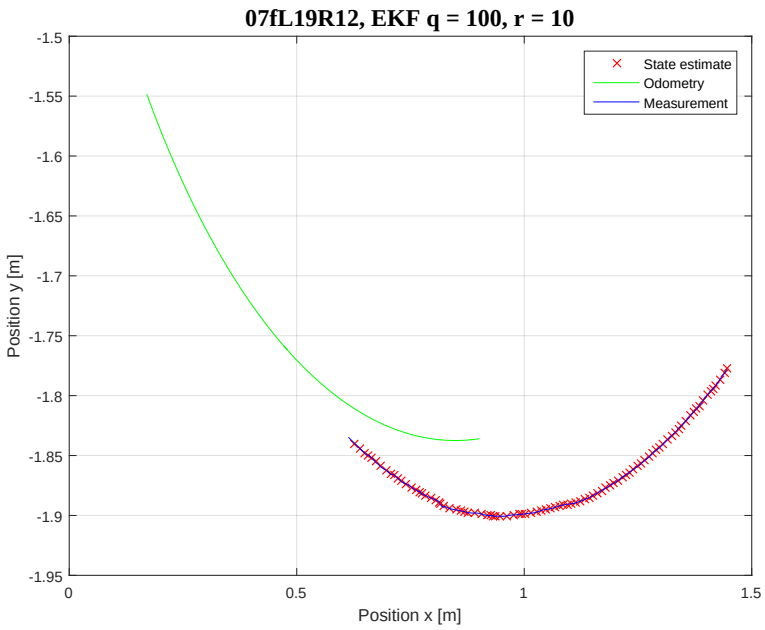

Figure 14: Test 6, velocity command 19% for left wheel and 12% for right wheel. q = 100, r = 10.

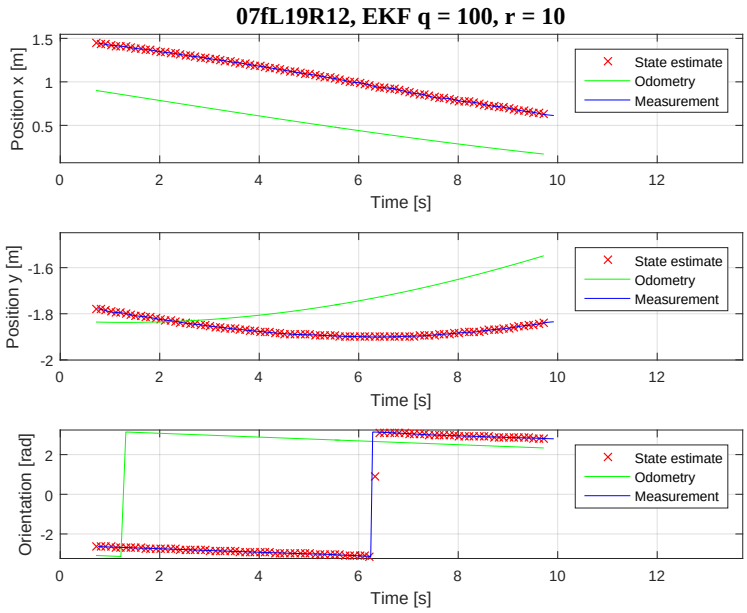

Figure 15: Test 6, velocity command 19% for left wheel and 12% for right wheel. q = 100, r = 10. Figure shows all three states with respect to time.

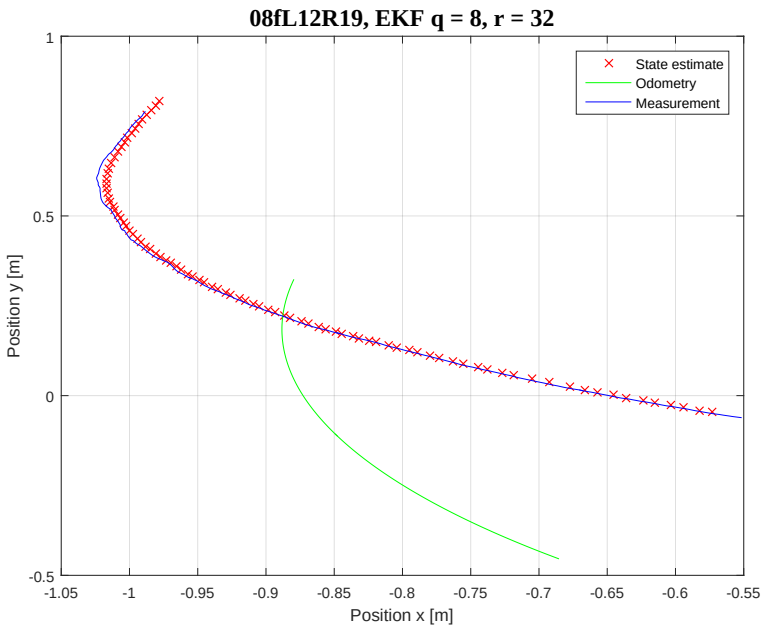

Figure 16: Test 7, velocity command 12% for left wheel and 19% for right wheel. q = 8, r = 32.

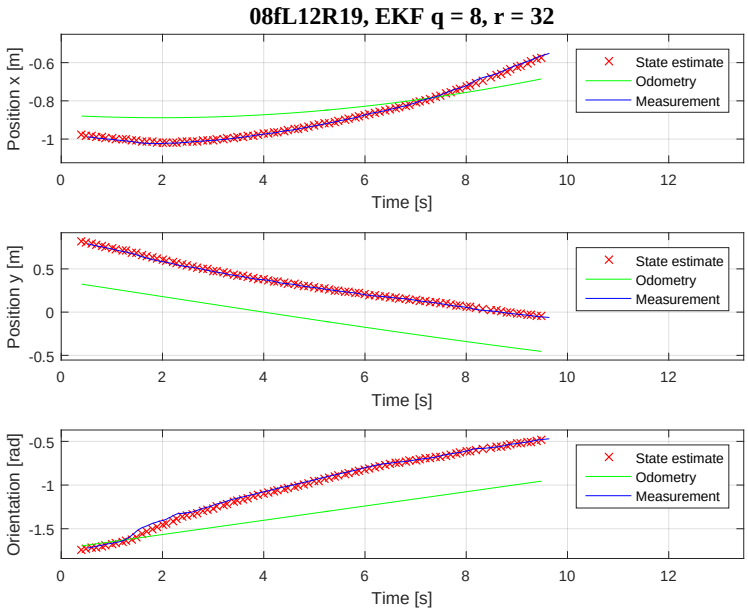

Figure 17: Test 8, velocity command 12% for left wheel and 19% for right wheel. q = 8, r = 32. Figure shows all three states with respect to time.