

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	2
Odometry	2
Clockwise direction	2
Counterclockwise direction	3
Straight line motion	3
Final set of equations	4
Adaptation of odometry model for Rollo	4
Odometry errors	5
Measuring odometry errors	7
System and measurements model	8
System model	8
System model with noise	9
Measurement model with noise	10
Kalman filter	11
Kalman filter equations for linear system	12
Kalman filter equations for non-linear system	13
Extended Kalman filter equations for odometry	13
Practical implementation of extended Kalman filter	15
Divergence of state covariance matrix $\Sigma_{k k}$	16
Robot Operating System (ROS)	16
Simulation and testing	18
Simulation of extended Kalman filter	18
Testing of extended Kalman filter	18
Conclusion	23

Rollo - Humanoid robot

Odometry

Odometry is one of the most widely used technique for determining the position of a mobile robot^{1 2}. 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 physical 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:

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 or axle length of 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)

$$S_L = \frac{n_L}{60} 2\pi r_L t \quad (1)$$

$$S_R = \frac{n_R}{60} 2\pi r_R t \quad (2)$$

¹ UMBmark – A Method for Measuring, Comparing, and Correcting Odometry Errors in Mobile Robots

² Comparative Study of Localization Techniques for Mobile Robots based on Indirect Kalman Filter

where n_L and n_R are the revolutions per minute ($[rpm]$) of left and right wheel, r_L and r_R are the radii of the two wheels and t is time step in seconds.

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

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

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

The equations (3) and (4) above can be solved simultaneously to compute $\Delta\Theta$ defined in (5).

$$\Delta\Theta = \frac{S_L - S_R}{l} \quad (5)$$

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

$$\Delta S = \frac{S_L + S_R}{2} \quad (6)$$

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 is approximated to ΔS for a very small step of time.

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

$$\Theta_f = \Theta_i - \Delta\Theta$$

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 revolutions per minute of right wheel are higher than that of the left wheel. $\Delta\Theta$ can be computed as shown in (8)

$$\Delta\Theta = \frac{S_R - S_L}{l} \quad (8)$$

The relations for final position and rotation are as follows (9).

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

$$\Theta_f = \Theta_i + \Delta\Theta$$

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, therefore $\Delta\Theta$ in this case is 0 and S_L is the same as S_R . The relations for final position and rotation are as follows:

$$P_f = [P_{ix} + S_{L=R} \cos(\Theta_i), \quad P_{iy} + S_{L=R} \sin(\Theta_i)] \quad (10)$$

$$\Theta_f = \Theta_i$$

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 as follows.

$$\Delta\Theta = \frac{S_L - S_R}{l} \quad (11)$$

$$\Delta S = \frac{S_L + S_R}{2} \quad (12)$$

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

$$\Theta_f = \Theta_i - \Delta\Theta$$

Adaptation of odometry model for Rollo

During the project realization the encoders feedback was unavailable in the robot and a physical odometry model could not have been implemented. However, an attempt has been made to implement its modified version. In absence of encoders feedback, the distance covered by left and right wheels is instead estimated from the control command.

In order to implement this simplification initial tests were conducted with Rollo. The robot was run in a straight line, constant curvature arcs and performed rotations at different commands. Rollo's location feedback from the motion capture system was logged. For a given command, average speed of left and right wheel was determined from the recorded data. Matlab environment was used to perform analysis of logged data according to following algorithm:

1. The position in x and y coordinates of Rollo was plotted along with *line of best fit* for the data set.
2. For the displacement of the robot total distance travelled by the robot and change in orientation was determined using the *line of best fit*.
3. Using the timestamps from the recorded data, distance travelled and change in orientation were determined for a unit of time.
4. The rotation speeds of the wheels were determined using the relation in equations (1) ÷ (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.

For simplicity only the forward motion was modelled.

Table 2 shows the predetermined angular velocities of the wheel for given command velocities determined according to the above algorithm.

Velocity command left v_L [%]	Velocity command right v_R [%]	Angular velocity left n_L [rad/s]	Angular velocity right n_R [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 based on test logs for given forward command.

Velocity command		Angular velocity	
Left v_L [%]	Right v_R [%]	Left n_L [rad/s]	Right n_R [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 2: Predetermined wheels angular velocity based on test logs for given forward command.

Challenges in the modelling of Rollo

1. Due to the mechanical differences in the two legs of Rollo, it is almost impossible for the two wheels to run at the same speed. Moreover, the two motors are powered using two different battery packs and are run in open loop. The difference in voltage level of battery packs results in a different current at the same velocity command using pulse-width modulation increasing the overall velocity error.
2. The robot randomly skids and changes its orientation even for a straight line run. This is partly due to mechanical properties of the robot, partly due to the surface of the laboratory, which serves as testing area and is uneven. The rubber of the tyres lacks sufficient grip on the floor to prevent skidding. One of the Rollo's wheels is visibly skewed. This is a major source for deviation in behaviour of robot's adjusted odometry model.
3. The odometry model is based on the assumption that the wheels have no thickness and have only one point of contact with ground surface. Rollo has four powered wheels by two motors, where each wheel is a 4cm thick. Additionally there are four support wheels in front and back of the two major wheel sets, so that Rollo does not fall over.

Odometry errors

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

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 the result of imperfections in the design, production and mechanical construction of a mobile robot and its components. In ideal environment conditions, like most smooth indoor surfaces, systematic errors play a much bigger role in than non-systematic errors. Systematic errors also accumulate constantly over time.

In general case there are three main sources of systematic errors in odometry of wheeled robots during a robot's movement:

- Distance
- Rotation
- Skew

Distance error accumulates with the robot travelling. Rotation and skew errors are of less significance with small distance and speed movements, however they are usually dominant with time and influence the overall error more than the distance error.

The major sources of the systematic error in an odometry model of a wheeled robot from a mechanical point of view are unequal diameters of wheels and the uncertainty about the effective wheelbase. Rollo consists of four plastic tyres covered in rubber stripes, two wheels on each side with additional four supporting wheels preventing Rollo from falling over.

Rubber on tyres helps improve traction, however with time rubber compresses differently under asymmetric load distribution. The diameters of wheels are used in computing S_L and S_R as shown in equations (1) and (2).

Furthermore, the contact area between the wheels of Rollo and the floor is comparatively wide for a robot of its size. This introduces difficulties in determining the wheelbase of the robot. Wheelbase is used in computation of angle of travel $\Delta\theta$ as shown in equation (11).

Other examples of sources of mechanical errors include:

- Average of both wheel diameters differ from nominal diameter
- Misalignment of wheels
- Unbalanced wheels
- Digitalization of acquired data (encoder sampling rate and digital to analog converter resolution)

Non-systematic errors

Non-systematic errors are caused by unknown or unpredictable changes in the system and in the environmental conditions. In wheeled mobile robots, they are usually caused by uneven floors, like in the case of testing area of Rollo. On rough surfaces with significant irregularities, non-systematic errors may be more dominant over systematic errors. On contrary to systematic errors, non-systematic errors do not accumulate with time, but occur randomly.

Possible examples of non-systematic errors are:

- Travel over uneven floors or unexpected objects on the floor
- Wheel slip due to:
 - Slippery floors
 - Over-acceleration
 - Fast turning (skidding)
 - Temporal loss of wheel contact with the floor

Measuring odometry errors

University of Michigan Benchmark test (UMBmark) procedure was supposed to be used to determine odometry errors and calibrate the provided system. This procedure was developed by Johann Borenstein and Liqiang Feng around 1995 at the University of Michigan's Advanced Technologies Laboratories. Further details of the test can be found in the research paper UMBmark: A Benchmark Test for Measuring Odometry Errors in Mobile Robots.

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

Unfortunately, due to unavailability of encoder feedback estimations of S_L and S_R are undertaken using the command given to the robot, so the control input. This means, that equations (1) and (2) are not used in the modified model. 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. Should it be possible 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.

As can be seen in the equation (11), incorrect wheelbase value will result in incorrect computation of angle of travel $\Delta\Theta$. This error, E_b is directly proportional to the effective wheelbase divided by the nominal wheelbase l :

$$E_b = \frac{\text{effective wheelbase}}{l} \quad (14)$$

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

$$\Delta\Theta = \frac{S_L - S_R}{E_b \cdot l} \quad (15)$$

E_b is determined using UMBmark square test 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 clockwise direction making sure to:
 - (a) stop after each $2m$ 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).

$$\epsilon x = x_{abs} - x_{calc} \quad (16)$$

$$\epsilon y = y_{abs} - y_{calc} \quad (17)$$

$$\epsilon \theta = \theta_{abs} - \theta_{calc} \quad (18)$$

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.

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

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

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

$$E_b = \frac{90^\circ}{90^\circ - \alpha} \quad (21)$$

where α can be computed as

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

or

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

and L represents the side of square path 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.

We can define the location of the robot at instant k using state variables, (24). This will include position in x and y coordinates and orientation.

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

The control input provided to robot is the speed of left and right 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 (7), (9) and (10), derived on page on page 3, 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.

$$u_k = j(d_k) \quad (25)$$

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

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

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

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 modelled 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 .

$$q_k \sim N(\hat{q}_k, U_k) \quad (27)$$

where

$$\hat{q}_k = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

and

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

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

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

The control input, or relative displacement can now be expressed as shown in (28).

$$u_k = j(d_k) + q_k \quad (28)$$

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

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. Modeling errors, odometry errors, discretization and approximations involved in the derivation of the model all play a part in this noise source. Noise vector, w_k is represented in equation (29)

$$w_k \sim N(\hat{w}_k, Q_k) \quad (29)$$

where

$$\hat{w}_k = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

and

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

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

With the assumption that the noise sources are independent, the off-diagonal elements of the covariance matrix Q_k are presumably zero. Determination of matrix Q_k is discussed in section ??.

The system can be expressed now as shown below.

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

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

Now, x_k is a random vector and with every time step the variance of system noise increases. 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.

Measurement model with noise

Starting with the assumption that there is no noise in the measurement, z_k 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

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

In our system, we use the motion capture system available in the laboratory 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).

$$z_k = \begin{bmatrix} x_{[x],k} \\ x_{[y],k} \\ x_{[\theta],k} \end{bmatrix} \quad (33)$$

Now, we can add noise to the measurement model. We assume that the noise in the measurement can

be modelled 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 .

$$v_k \sim N(\hat{v}_k, R_k) \quad (34)$$

where

$$\hat{v}_k = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

and

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

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

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

The measurement model can be expressed now as shown in (36).

$$z_k = x_k + v_k \quad (35)$$

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

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).

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 systems 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 the 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 systems 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. Very easy implementation
2. Computationally efficient, update filter when adding new measurements to the existing data set
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 systems 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.

$$\begin{aligned}x_{k+1} &= A_k x_k + B_k u_k + w_k \\ z_k &= C_k x_k + v_k\end{aligned}\tag{36}$$

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

For Kalman filter, A , B , C , Q and R can be time variant. However, for simplicity, we will proceed with the assumption that A , B , C , Q and R are time invariant for provided 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/time update:

$$\begin{aligned}\hat{x}_{k|k-1} &= A\hat{x}_{k-1|k-1} + Bu_{k-1} \\ \Sigma_{k|k-1} &= A\Sigma_{k-1|k-1}A^T + Q\end{aligned}\tag{37}$$

Innovation/measurement update

$$\begin{aligned} H_k &= \Sigma_{k|k-1} C^T (C \Sigma_{k|k-1} C^T + R)^{-1} \\ \hat{x}_{k|k} &= \hat{x}_{k|k-1} + H_k [z_k - C \hat{x}_{k|k-1}] \\ \Sigma_{k|k} &= [I - H_k C] \Sigma_{k|k-1} \end{aligned} \quad (38)$$

Kalman filter equations for non-linear system

Now we consider a non-linear extension to the Kalman filter, *Extended Kalman Filter (EKF)*. The EKF implements 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.

$$\begin{aligned} x_k &= f(x_{k-1}, u_{k-1}) + w_{k-1} \\ z_k &= h(x_k) + v_{k-1} \end{aligned} \quad (39)$$

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

We will proceed with the assumption that $f(\cdot)$, $h(\cdot)$, Q and R are time invariant for our non linear system. $f(\cdot)$ and $h(\cdot)$ are linearized about the prior best estimates of the states at each instant of time by finite difference method in order to compute covariances. 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/time update

$$\begin{aligned} \hat{x}_{k|k-1} &= f(\hat{x}_{k-1|k-1}, u_{k-1}) \\ \Sigma_{k|k-1} &= J_f \Sigma_{k-1|k-1} J_f^T + Q \end{aligned} \quad (40)$$

Innovation/measurement update

$$\begin{aligned} H_k &= \Sigma_{k|k-1} J_h^T (J_h \Sigma_{k|k-1} J_h^T + R)^{-1} \\ \hat{x}_{k|k} &= \hat{x}_{k|k-1} + H_k [z_k - h(\hat{x}_{k|k-1})] \\ \Sigma_{k|k} &= [I - H_k J_h] \Sigma_{k|k-1} \end{aligned} \quad (41)$$

where J_f and J_h are the jacobian matrices.

J_f is the jacobian matrix containing the partial derivative 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 containing partial derivatives of the measurement function $h(\cdot)$ 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.

$$\begin{bmatrix} x_{[x],k} \\ x_{[y],k} \\ x_{[\theta],k} \end{bmatrix} = \begin{bmatrix} f_x(x_{k-1}, u_{k-1}) \\ f_y(x_{k-1}, u_{k-1}) \\ f_\theta(x_{k-1}, u_{k-1}) \end{bmatrix} + \begin{bmatrix} w_{[x],k-1} \\ w_{[y],k-1} \\ w_{[\theta],k-1} \end{bmatrix} \quad (42)$$

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

Prediction/time update:

$$\begin{bmatrix} \hat{x}_{[x],k|k-1} \\ \hat{x}_{[y],k|k-1} \\ \hat{x}_{[\theta],k|k-1} \end{bmatrix} = \begin{bmatrix} \hat{x}_{[x],k-1|k-1} + u_{[\Delta]S,k-1} \cdot \cos(\hat{x}_{[\Theta],k-1|k-1} - \frac{u_{[\Delta\Theta],k-1}}{2}) \\ \hat{x}_{[y],k-1} + u_{[\Delta]S,k-1} \cdot \sin(\hat{x}_{[\Theta],k-1|k-1} - \frac{u_{[\Delta\Theta],k-1}}{2}) \\ \hat{x}_{[\theta],k-1|k-1} - u_{[\Delta\Theta],k-1} \end{bmatrix} \quad (44)$$

$$\begin{aligned} J_{f,k} &= \left. \frac{\partial f(x)}{\partial x} \right|_{x=\hat{x}_{k-1|k-1}, u=u_{k-1}} \\ &= \begin{bmatrix} \frac{\partial f_x}{\partial x_{[x]}} & \frac{\partial f_x}{\partial x_{[y]}} & \frac{\partial f_x}{\partial x_{[\theta]}} \\ \frac{\partial f_y}{\partial x_{[x]}} & \frac{\partial f_y}{\partial x_{[y]}} & \frac{\partial f_y}{\partial x_{[\theta]}} \\ \frac{\partial f_\theta}{\partial x_{[x]}} & \frac{\partial f_\theta}{\partial x_{[y]}} & \frac{\partial f_\theta}{\partial x_{[\theta]}} \end{bmatrix}_{x=\hat{x}_{k-1|k-1}, u=u_{k-1}} \\ &= \begin{bmatrix} 1 & 0 & -u_{[\Delta]S} \cdot \sin(x_{[\Theta]} - \frac{u_{[\Delta\Theta]}}{2}) \\ 0 & 1 & +u_{[\Delta]S} \cdot \cos(x_{[\Theta]} - \frac{u_{[\Delta\Theta]}}{2}) \\ 0 & 0 & 1 \end{bmatrix}_{x=\hat{x}_{k-1|k-1}, u=u_{k-1}} \end{aligned} \quad (45)$$

$$\Sigma_{[x],k|k-1} = J_f \Sigma_{[x],k-1|k-1} J_f^T + Q \quad (46)$$

Innovation/measurement update

$$\begin{aligned} J_{h,k} &= \left. \frac{\partial h(x)}{\partial x} \right|_{x=\hat{x}_{k|k-1}} \\ &= \begin{bmatrix} \frac{\partial h_x}{\partial x_{[x]}} & \frac{\partial h_x}{\partial x_{[y]}} & \frac{\partial h_x}{\partial x_{[\theta]}} \\ \frac{\partial h_y}{\partial x_{[x]}} & \frac{\partial h_y}{\partial x_{[y]}} & \frac{\partial h_y}{\partial x_{[\theta]}} \\ \frac{\partial h_\theta}{\partial x_{[x]}} & \frac{\partial h_\theta}{\partial x_{[y]}} & \frac{\partial h_\theta}{\partial x_{[\theta]}} \end{bmatrix}_{x=\hat{x}_{k|k-1}, u=u_{k-1}} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}_{x=\hat{x}_{k|k-1}, u=u_{k-1}} \end{aligned} \quad (47)$$

Using the above derived jacobian matrix (47) for computation of Kalman gain matrix, H_k , aposteriori state estimate and aposteriori error covariance matrix, we get following equations.

$$\begin{aligned} H_k &= \Sigma_{k|k-1} (\Sigma_{k|k-1} + R)^{-1} \\ \hat{x}_{k|k} &= \hat{x}_{k|k-1} + H_k [z_k - \hat{x}_{k|k-1}] \\ \Sigma_{k|k} &= [I - H_k J_h] \Sigma_{k|k-1} \end{aligned} \quad (48)$$

Practical implementation of extended Kalman filter

Proceeding with the EKF model for odometry, we need initializing of the model before we can practically implement it. Following components need to be initialized.

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 motion capture system, 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 decomposition 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.

We implemented this based on the assumption that the state covariance matrix is symmetrical $\Sigma_{k|k}$. 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 (49)

$$\begin{aligned} U &= CholeskyDecomposition(J_h \Sigma_{k|k-1} J_h^T + R) \\ \hat{x}_{k|k} &= \hat{x}_{k|k-1} + [\Sigma_{k|k-1} U^{-1}] U^{T-1} [z_k - h(\hat{x}_{k|k-1})] \\ \Sigma_{k|k} &= \Sigma_{k|k-1} - [\Sigma_{k|k-1} U^{-1}] [\Sigma_{k|k-1} U^{-1}]^T \end{aligned} \quad (49)$$

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

Robot Operating System (ROS)

Robot Operating System (ROS) is a collection of software frameworks for robot software development, it works as middleware for work with complex robotic problems. It uses the publisher subscriber software architecture model for nodes. The presented problem of localization of a humanoid robot using extended Kalman filter, which has been described in detail in previous section [ref section], has been solved using ROS. [Number of nodes] nodes have been developed respecting the aspects of modularity and exchangeability of software:

- Preprocessor node
- Control node
- Communication node
- Extended Kalman filter node
- Visualization

Analysis

Each node is supposed to fulfill one function, however there are several aspects that have been merged into nodes, that might be contrainuitive. Functionalities of the nodes are distributed the following way:

1. Preprocessor node:
2. Control node:

- (a) Translate input from keyboard into linear and angular velocities within unitary limits ($-100\% \div 100\%$)
- (b) Publish data

3. Communication node:

- (a) Translate data from control node into a 3 byte message for Rollo
- (b) Send data continuously at a given rate to Rollo
- (c) Perform emergency procedure if connection to control node is lost

ous options, B for this one

- (d) Square test of n-th order
 - i. Perform a cycle of forward movement and turn
 - ii. For higher than 1 order square test, turn around between runs

4. Extended Kalman filter node:

- (a) Preprocessing of acquired data to make comparison of relevant information
- (b) Estimate of state using EKF
- (c) Odometry data calculation
- (d) Publish results

5. Visualization node:

- (a) Visualize data from preprocessor and extended Kalman filter, which provides estimates of state and odometry data

Nodes have been written in C++ with the exception of visualization node, which is written in Python, because of easy access to the powerful Matplotlib [reference] library. There has been also a significant amount of testing and prototyping of software done in Matlab and Python, mainly for the odometry model and extended Kalman filter implementation and analysis. Additional scripts in Bash shell were also written, mostly for the purpose of easier repetition of tasks.

The basic software architecture model implemented is publish-subscribe given the architecture ROS uses represented by [reference to diagram].

More in depth documentation of the actual code and algorithms has been done using Doxygen in two formats: html version [reference to github] and local document form [reference to refman.pdf]. Only the most important points shall be described here.

A few aspects have been addressed while developing the software. One of those aspects is safety of operation. Should the control node loose connection to the master and at the same time be sending other than full stop (linear and angular velocities equal to zero), then in normal case the communication node would continue sending data with last values received from control node. This is the reason a safety or emergency procedure has been implemented, so that after a period of time, which can be specified from the command line, a full stop command is being send for 10 [verify] times. Should the control node reestablish connection, the communication node will continue to work in normal procedure. Since the robot has been tested in laboratory conditions, the algorithm implemented is not very robust and could be further improved. For example, the stop command could be send continuously, until control node is reconnected with ROS master or forcefully exited. Another improvement would be to send a special message that this event occurred, so that the user is informed about it without delay.

Next feature implemented into the communication node is the square test. In this particular test [provide more description] [provide reference]. The implemented algorithm lets the user specify the forward and turning times for the square test, which is necessary in case of not having access to odometry data. The square test can be performed in multiple runs, so that each cycle includes:

1. Moving forward
2. Turning 90 deg
3. Moving forward
4. Turning 90 deg
5. Moving forward
6. Turning 90 deg
7. Moving forward
8. Turning 90 deg

Should there be multiple runs specified, the robot would turn around, which is to rotate 180° and continue with step 1. In other case that would be the last run, ideally the robot would stop at exactly the same position it started with the initial orientation. Since there are systematic and random errors, this is only achievable for a given tolerance.

During tests runs it has been only confirmed, what the analysis of the logs performed for the extended Kalman filter node concluded, namely that the turning times differ significantly, which is most probably due to mechanical asymmetry. This makes the source code for this particular robot somewhat irrelevant. However it has been decided to leave the current implementation at this state, since it actually is sound from the theoretical point of view and most robots should have the similar turning speeds for both directions.

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 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 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

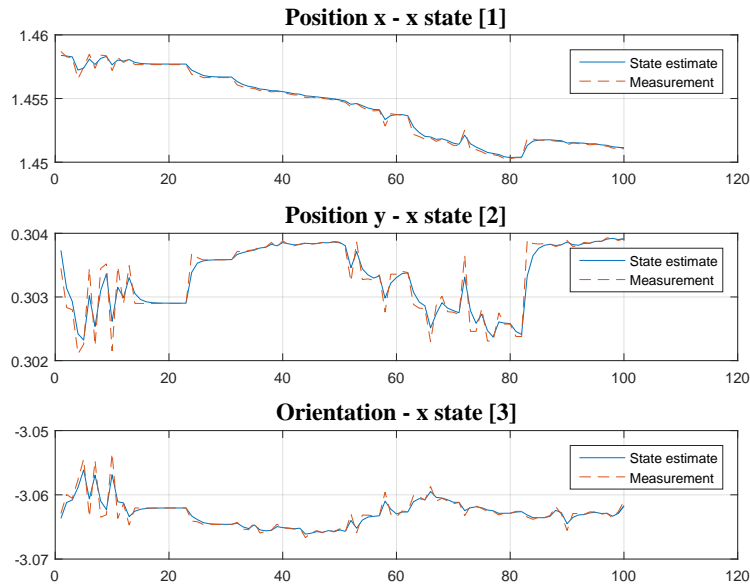


Figure 1: Results of EKF simulation a log file.

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 2. were tested.

In the start of each test run, the robot position and orientation read from the motion capture system

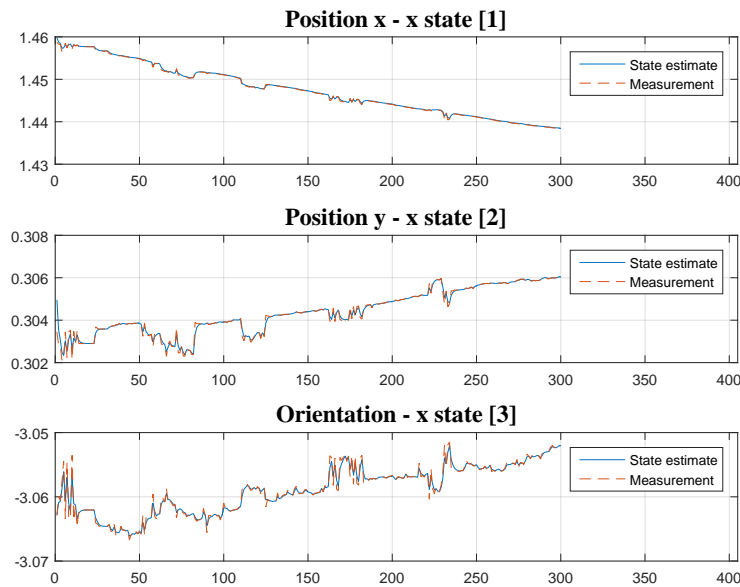


Figure 2: Results of EKF simulation a log file.

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 modelled 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 (50).

$$Q_k = \begin{bmatrix} q^2 & 0 & 0 \\ 0 & q^2 & 0 \\ 0 & 0 & q^2 \end{bmatrix} \quad (50)$$

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 (51).

$$R_k = \begin{bmatrix} r^2 & 0 & 0 \\ 0 & r^2 & 0 \\ 0 & 0 & r^2 \end{bmatrix} \quad (51)$$

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.

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

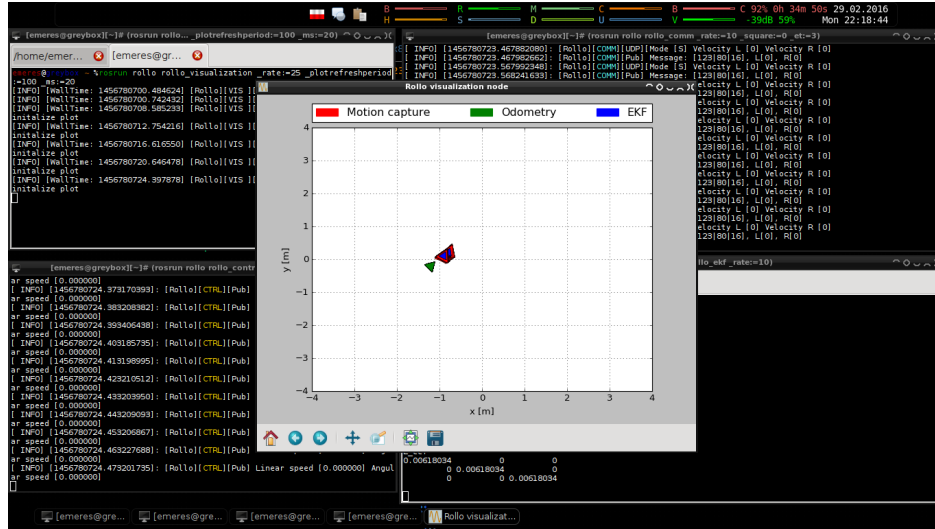


Figure 3: Pseudo real time visualization: Localization of the robot based on EKF, odometry and the measurement from motion capture.

drifted far from actual position of the robot with time. Since this was only crudely modelled 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 value of q , we are increasing the uncertainty in our process model. In comparison to the test analyzed 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 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 ,

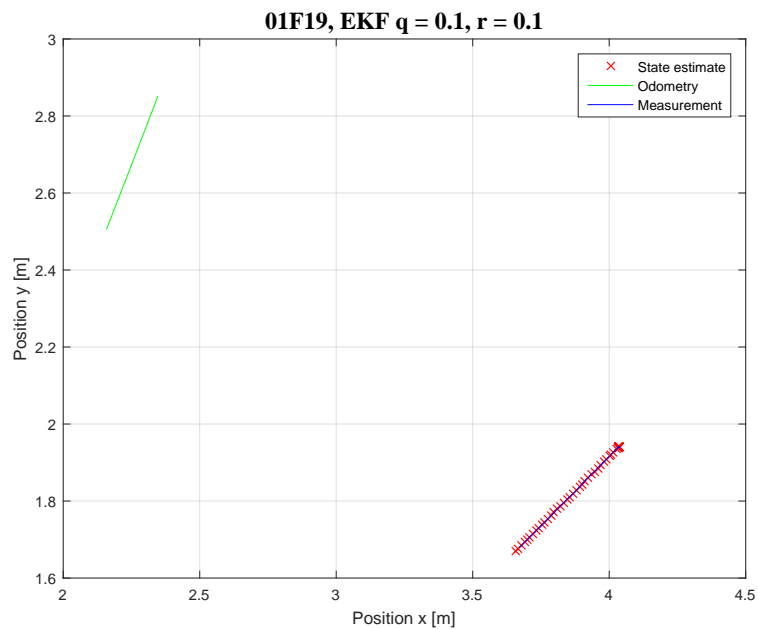


Figure 4: Test 1, velocity command 19% for both left and right wheels. $q = 0.1, r = 0.1$.

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.

Conclusion

Localization is the estimation of position and state of a system in the world frame of reference making use of limited information. Localization has been one of the primary challenges of working with mobile robots. It is very common in mobile robots to use odometry and inertial measurement unit (IMU) as relative position sensors and GPS as absolute position sensor to estimate state of the robot. Extended Kalman filter is a

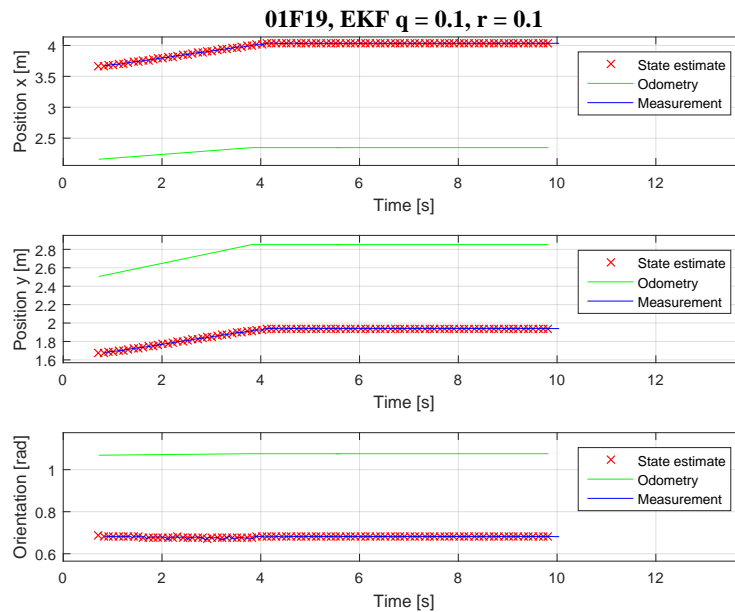


Figure 5: Test 1, velocity command 19% for both left and right wheels. $q = 0.1$, $r = 0.1$. Figure shows all three states with respect to time.

widely used filtering technique in estimating the localization of a robot that processes feedback from two or more sensors.

The presented project starts with preparing the odometry based physical model of Rollo, showing it's advantages and disadvantages with respect to the localization problem. Furthermore errors associated with odometry, divided into systematic and non systematic, were presented along with the UMBMark based square test technique to determine dominant odometry errors.

Next the preparation of the system and measurement models was conducted and the equations for the implementation of EKF were defined. EKF was first simulated and verified in Matlab. The final tests were conducted on Rollo and the performance of EKF was analyzed by varying process covariance matrix, Q and

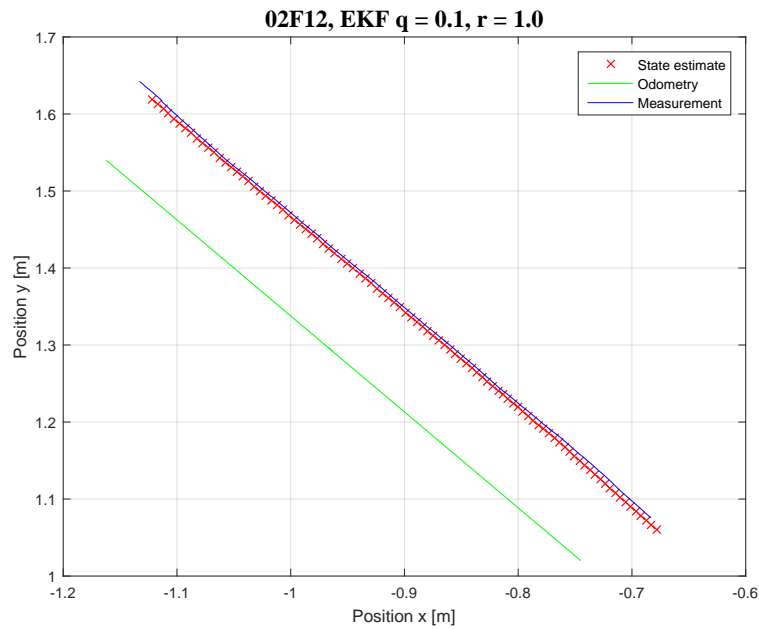


Figure 6: Test 2, velocity command 12% for both left and right wheels. $q = 0.1, r = 1.0$.

noise covariance matrix, R .

For future work, it is proposed to first implement encoders feedback in the system and validate the EKF with working odometry. Even though the encoders feedback is currently unavailable in Rollo, the modeling of the system and software was designed for a swift and easy transition to odometry feedback. Next, another sensor should be introduced in the system since the motion capture system will be unavailable in field environment of the robot.

In regards to the mechanical system of the robot, it is recommended that a different material for the wheels should be used to improve traction and reduce skidding of the robot. The wheels of the robot also need to be properly aligned so that the odometry errors are reduced.

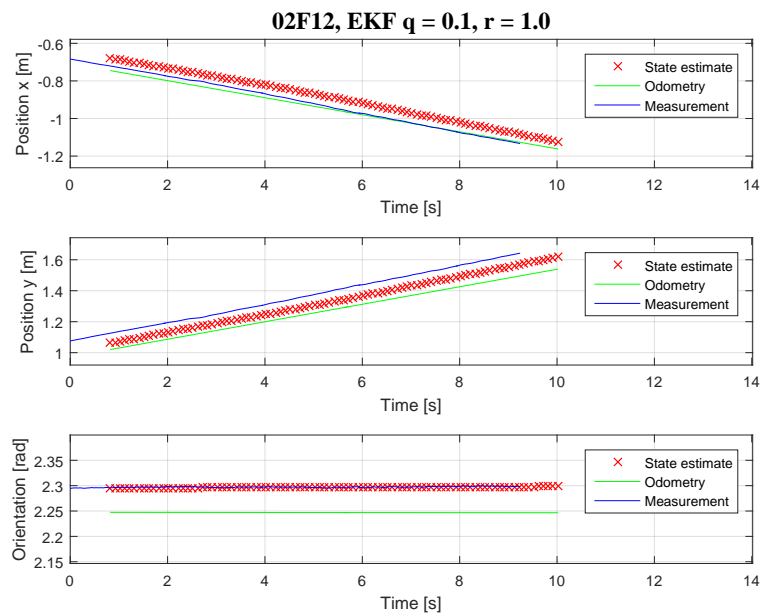


Figure 7: Test 2, velocity command 12% for both left and right wheels. $q = 0.1$, $r = 1.0$. Figure shows all three states with respect to time.

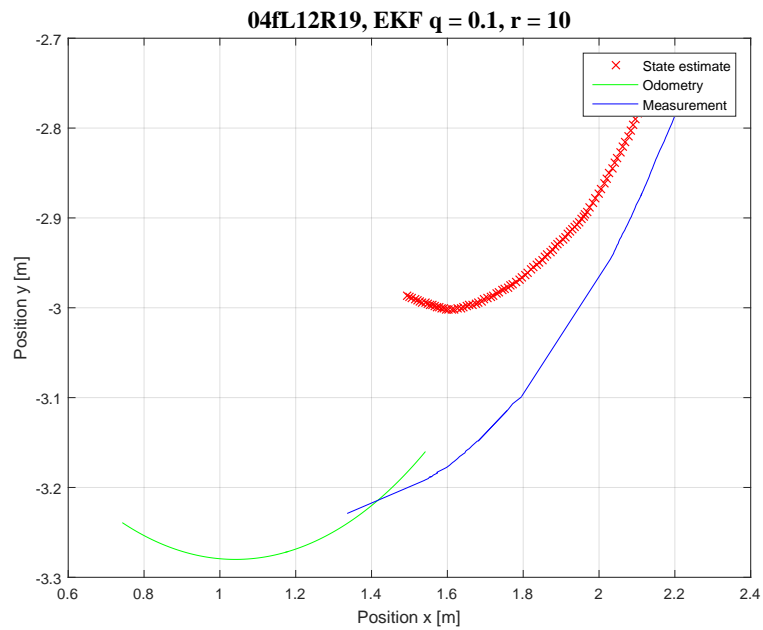


Figure 8: Test 3, velocity command 12% for left wheel and 19% for right wheel. $q = 0.1$, $r = 10$.

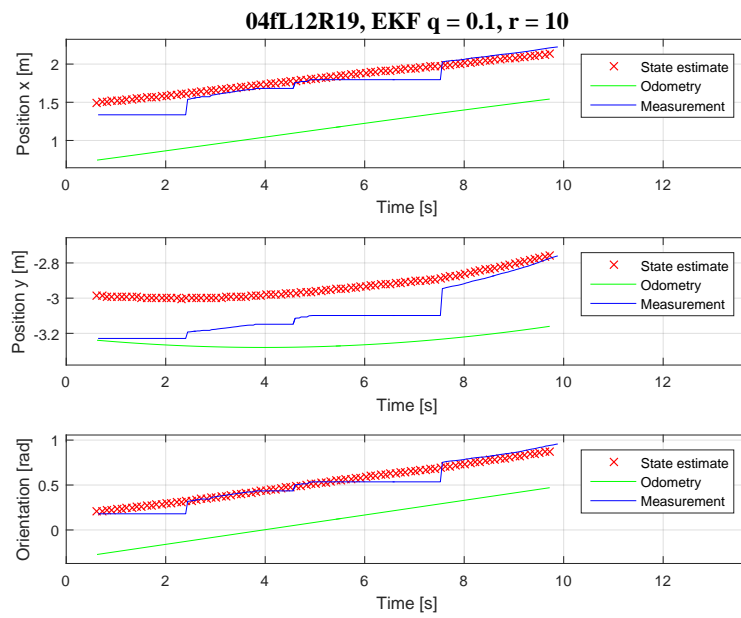


Figure 9: Test 3, velocity command 12% for left wheel and 19% for right wheel. $q = 0.1$, $r = 10$. Figure shows all three states with respect to time.

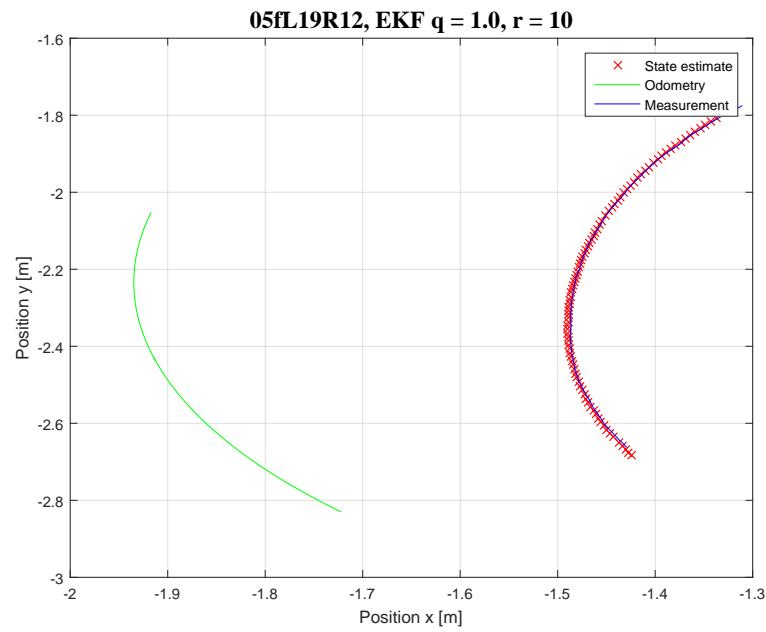


Figure 10: Test 4, velocity command 19% for left wheel and 12% for right wheel. $q = 1.0$, $r = 10$.

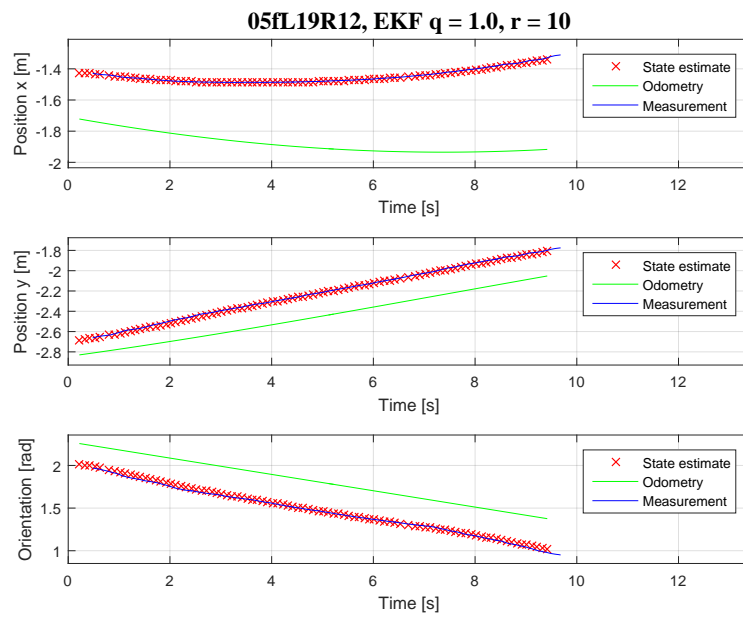


Figure 11: Test 4, velocity command 19% for left wheel and 12% for right wheel. $q = 1.0$, $r = 10$. Figure shows all three states with respect to time.

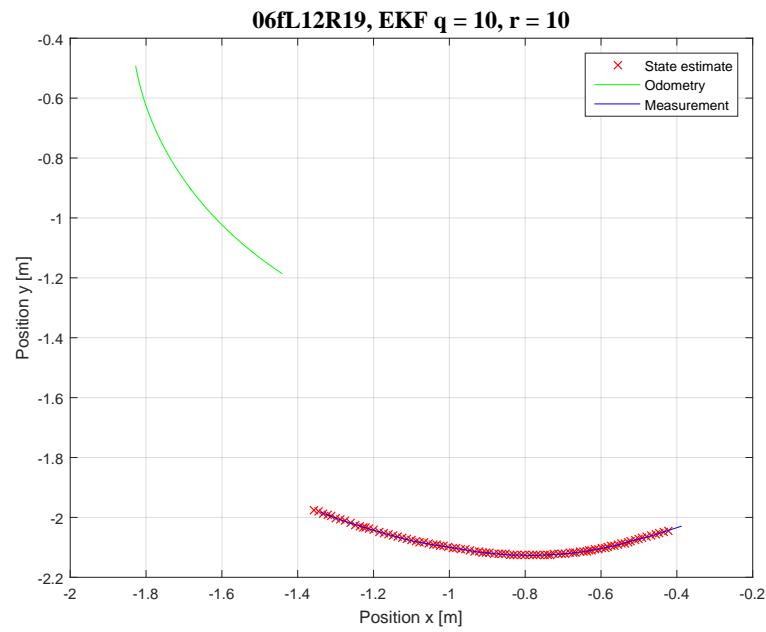


Figure 12: Test 5, velocity command 12% for left wheel and 19% for right wheel. $q = 10$, $r = 10$.

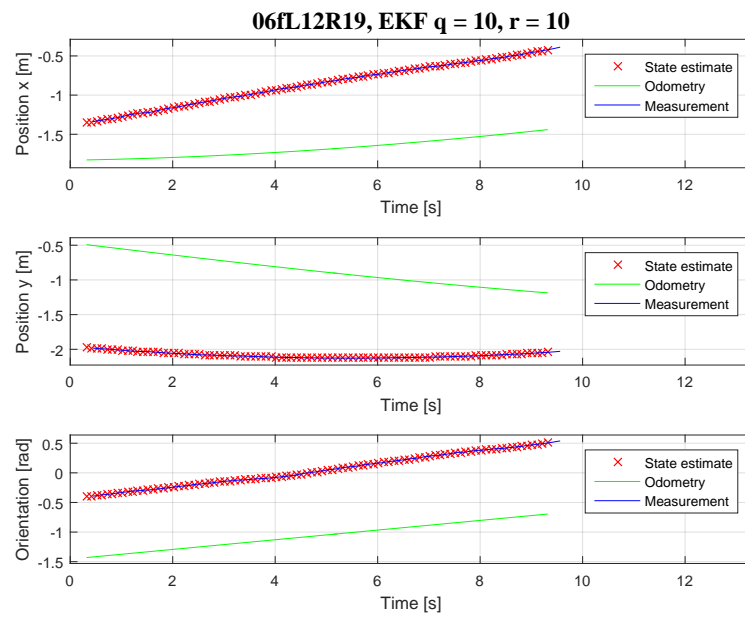


Figure 13: Test 5, velocity command 12% for left wheel and 19% for right wheel. $q = 10$, $r = 10$. Figure shows all three states with respect to time.

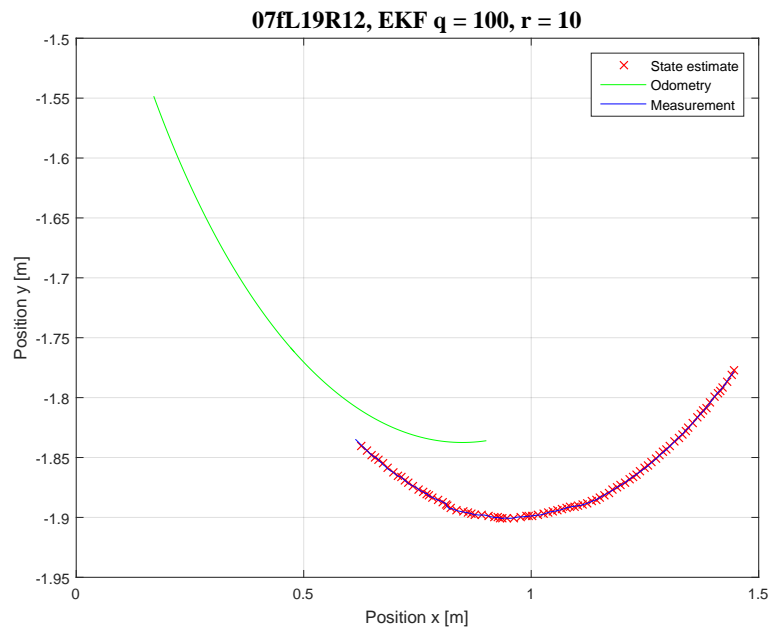


Figure 14: Test 6, velocity command 19% for left wheel and 12% for right wheel. $q = 100$, $r = 10$.

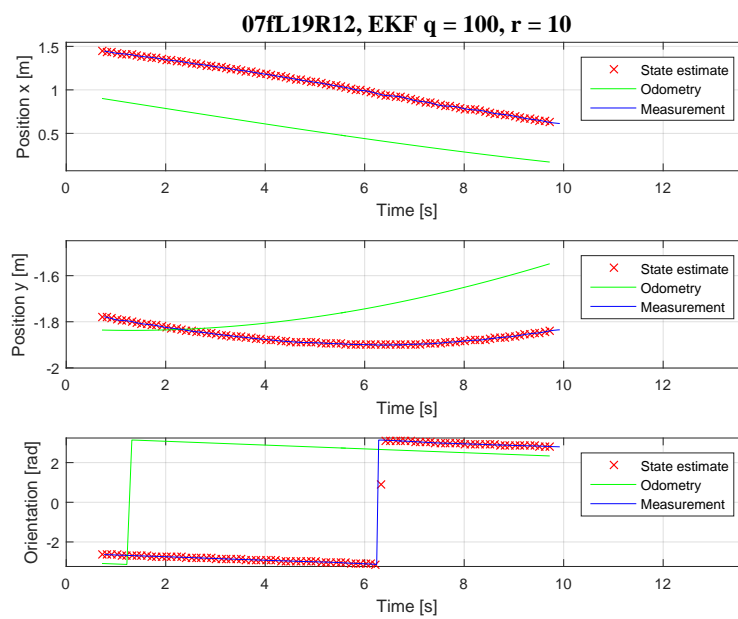


Figure 15: Test 6, velocity command 19% for left wheel and 12% for right wheel. $q = 100$, $r = 10$. Figure shows all three states with respect to time.

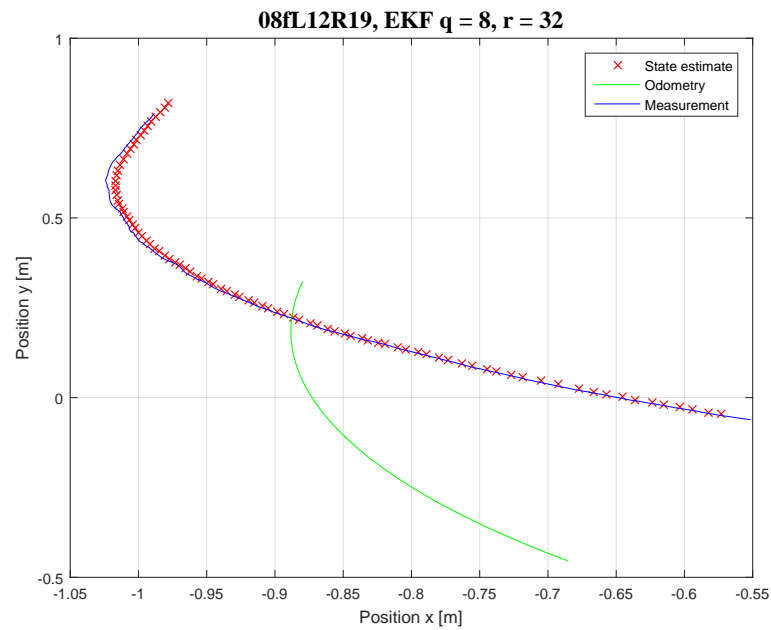


Figure 16: Test 7, velocity command 12% for left wheel and 19% for right wheel. $q = 8$, $r = 32$.

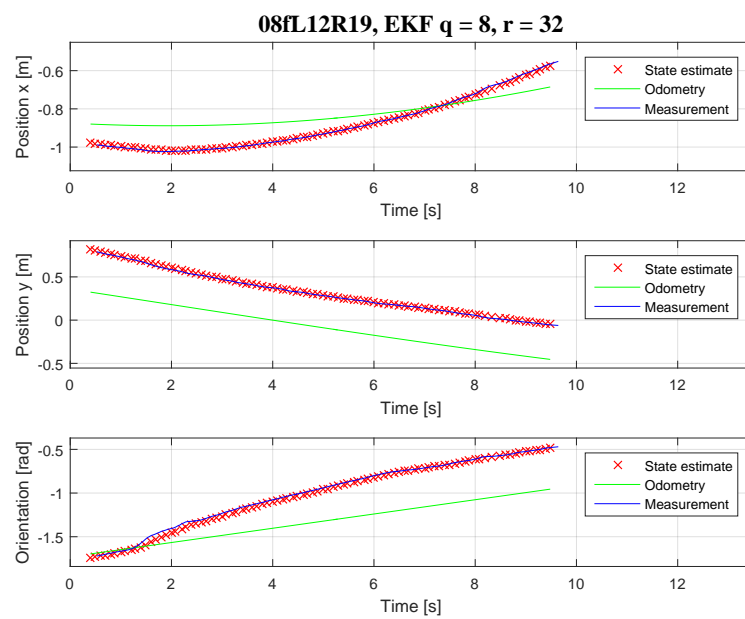


Figure 17: Test 8, velocity command 12% for left wheel and 19% for right wheel. $q = 8, r = 32$. Figure shows all three states with respect to time.