AER1513: State Estimation for Aerospace Vehicles
Assignment 2: Lost in the Woods Dataset
(2D pose estimation)

Ahmed Elkoushy

October 31, 2024

1 Modeling Covariance Matrices

Based on the histograms of the sensors, the assumption for zero-mean Gaussian noise is reasonable:

Range errors: The mean is centered at zero, however the mode (the bucket with the highest frequency) is slightly to the left of zero, but it is still close enough to resemble a zero-mean normal distribution expecially since the bucket making up the mode is the closest as can be to zero on the left side.

Bearing errors: A similar pattern is visible here, with a mean of zero but a mode shifted slightly to the left, but for the same reasons above we can approximate it as a zero mean normal distribution.

Translational speed errors: The mean here is shifted slightly to the right of zero, and so is the mode, however the bucket representing the mode is the closest as can be to zero from the right side, and the mean (0.000386 m/s) is very close to zero, making it reasonable to approximate it as a a zero-mean Gaussian.

Rotational speed errors: The mean here is at zero, with two equal buckets at maximum value centered exactly as zero, making this probably the safest distribution to approximate as a zero-mean Gaussian. Although the approximation is reasonable, the slight shifts in mean (in the translational speed case) could contribute to biases in the estimator and the distributions themselves not being exactly Gaussian can lead to inconsistencies since the covariance matrices we derived based off these Gaussians may not capture the exact error in the estimator. It is important to acknowledge this source especially since there will be more sources of error from linearizing the motion and observation models later as well.

Given these approximations, let us determine what we should use for $\boldsymbol{Q_k}$, the covariance of the process noise $\boldsymbol{w_k}$. Let us write $\boldsymbol{w_k} = [w_{v,k}, w_{\omega,k}]^T$ where $w_{v,k}$ and $w_{\omega,k}$ are the process noise related to the translational and rotational speed measurements respectively. They can be approximated as zero mean Gaussians with variances σ_v^2 and σ_ω^2 respectively. Assuming the errors in the two speeds are statistically independent, the covariance matrix is given as:

$$\boldsymbol{Q_k} = \begin{bmatrix} \sigma_v^2 & 0 \\ 0 & \sigma_\omega^2 \end{bmatrix}$$

Next is to figure out what values we should use for the variances. We take σ_v and σ_ω as the specified standard deviations of the errors in the translational and rotational speeds respectively.

A symmetrical argument applies to $\boldsymbol{R_k^l}$, the covariance of the exteroceptive sensor noise $\boldsymbol{n_k^l}$ coming from the range and bearing measurements of a specific landmark l. We rewrite $\boldsymbol{n_k^l} = [n_{k,r}^l, n_{k,\phi}^l]^T$, with $n_{k,r}^l$ and $n_{k,\phi}^l$ being the exteroceptive sensor noise associated with the range and bearing readings for landmark l respectively. Assuming those two are statistically independent, we then have:

$$\boldsymbol{R_k^l} = \begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\phi^2 \end{bmatrix}$$

Then, for the variances we take σ_r to be the standard deviation of the range error and σ_ϕ to be the standard deviation of the bearing error.

2 Deriving Jacobians

Recall the motion model:

$$\begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \theta_{k-1} \end{bmatrix} + T \begin{bmatrix} cos\theta_{k-1} & 0 \\ sin\theta_{k-1} & 0 \\ 0 & 1 \end{bmatrix} \left(\begin{bmatrix} v_k \\ \omega_k \end{bmatrix} + \boldsymbol{w_k} \right)$$

which is in the form:

$$\boldsymbol{x_k} = \boldsymbol{h}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k, \boldsymbol{w}_k), \boldsymbol{x_k} = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix}, \boldsymbol{u_k} = \begin{bmatrix} v_k \\ \omega_k \end{bmatrix}$$

and the observation model:

$$\begin{bmatrix} r_k^l \\ \phi_k^l \end{bmatrix} = \begin{bmatrix} \sqrt{(x_l - x_k - dcos\theta_k)^2 + (y_l - y_k - dsin\theta_k)^2} \\ atan2(y_l - y_k - dsin\theta_k, x_l - x_k - dcos\theta_k) - \theta_k \end{bmatrix} + \boldsymbol{n_k^l}$$

which is in the form:

$$\boldsymbol{y_k^l} = \boldsymbol{g_l(x_k, n_k^l)}$$

Based on these models, the Extended Kalman filter requires four Jacobians to linearize the models. These Jacobians were verified symbolically in MATLAB (especially the more complicated ones in the observation model) as seen in the Appendix. The Jacobians are:

- The motion model with respect to the state:

$$\boldsymbol{H_{x,k}} = \left. \frac{\partial \boldsymbol{h(x_{k-1}, u_k, w_k)}}{\partial \boldsymbol{x_{k-1}}} \right|_{\boldsymbol{\hat{x}_{k-1}, u_k, 0}} = \begin{bmatrix} 1 & 0 & -Tv_k sin\hat{\theta}_{k-1} \\ 0 & 1 & Tv_k cos\hat{\theta}_{k-1} \\ 0 & 0 & 1 \end{bmatrix}$$

- The motion model with respect to the process noise:

$$\boldsymbol{H_{w,k}} = \left. \frac{\partial \boldsymbol{h(x_{k-1}, u_k, w_k)}}{\partial \boldsymbol{w_k}} \right|_{\boldsymbol{\hat{x}_{k-1}, u_k, 0}} = \begin{bmatrix} Tcos\hat{\theta}_{k-1} & 0 \\ Tsin\hat{\theta}_{k-1} & 0 \\ 0 & T \end{bmatrix}$$

- The observation model with respect to the state for a given landmark l (the messiest one due to $atan2$ and taking the derivative with respect to θ_k!):

$$\boldsymbol{G_{x,k}^l} = \left. \frac{\partial \boldsymbol{g_l(x_k, n_k^l)}}{\partial \boldsymbol{x_k}} \right|_{\boldsymbol{\check{x}_k, 0}}$$

$$= \begin{bmatrix} \frac{\check{x}_k - x_l + dcos\check{\theta}_k}{\sqrt{(x_l - \check{x}_k - dcos\check{\theta}_k)^2 + (y_l - \check{y}_k - dsin\check{\theta}_k)^2}} & \frac{\check{y}_k - y_l + dsin\check{\theta}_k}{\sqrt{(x_l - \check{x}_k - dcos\check{\theta}_k)^2 + (y_l - \check{y}_k - dsin\check{\theta}_k)^2}} & \frac{dsin\check{\theta}_k (x_l - \check{x}_k - dcos\check{\theta}_k) - dcos\check{\theta}_k (y_l - \check{y}_k - dsin\check{\theta}_k)}{\sqrt{(x_l - \check{x}_k - dcos\check{\theta}_k)^2 + (y_l - \check{y}_k - dsin\check{\theta}_k)^2}} \\ \frac{y_l - \check{y}_k - dsin\check{\theta}_k}{(x_l - \check{x}_k - dcos\check{\theta}_k)^2 + (y_l - \check{y}_k - dsin\check{\theta}_k)^2} & \frac{-(x_l - \check{x}_k - dcos\check{\theta}_k)}{(x_l - \check{x}_k - dcos\check{\theta}_k)^2 + (y_l - \check{y}_k - dsin\check{\theta}_k)^2} & \frac{-dcos\check{\theta}_k (x_l - \check{x}_k - dcos\check{\theta}_k) - dsin\check{\theta}_k (y_l - \check{y}_k - dsin\check{\theta}_k)}{(x_l - \check{x}_k - dcos\check{\theta}_k)^2 + (y_l - \check{y}_k - dsin\check{\theta}_k)^2} \end{bmatrix}$$

- The observation model with respect to the exteroceptive sensor noise for a given landmark l:

$$\boldsymbol{G_{n,k}^l} = \left. \frac{\partial \boldsymbol{g_l(x_k, n_k^l)}}{\partial \boldsymbol{n_k^l}} \right|_{\boldsymbol{\check{x}_k, 0}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \boldsymbol{1}_{2\times 2}$$

With these Jacobians, we can derive all the equations of the Kalman Filter by relating $\boldsymbol{H_{w,k}}$ to $\boldsymbol{Q_k'}$ and $\boldsymbol{G_{n,k}^l}$ to $\boldsymbol{R_k^l}'$as well as coming up with a way to concatenate $\boldsymbol{y_k^l}$, $\boldsymbol{G_{x,k}^l}$, and $\boldsymbol{R_k^l}'$ for the variable number of landmarks seen at each timestep. These relationships are derived in the next section.

3 EKF Modification Taking Into Account Variable Landmarks at Each Timestep

First, let us relate $\boldsymbol{H_{w,k}}$ to $\boldsymbol{Q_k'}$ and $\boldsymbol{G_{n,k}^l}$ to $\boldsymbol{R_k^l}'$. The Extended Kalman Filter transforms the covariance matrices $\boldsymbol{Q_k}$ and $\boldsymbol{R_k^l}$ to $\boldsymbol{Q_k'}$ and $\boldsymbol{R_k^l}'$ in order to take into account the Jacobians of the linearized model. The specific relationship is given by:

$$\boldsymbol{Q_k'} = E[\boldsymbol{w_k' w_k'^T}]$$
$$\boldsymbol{R_k^l}' = E[\boldsymbol{n_k^{l'} n_k^{l'T}}]$$

where $\boldsymbol{w_k' = H_{w,k} w_k}$, $\boldsymbol{n_k^{l'} = G_{n,k}^l n_k^l}$. We then obtain:

$$\begin{aligned}\boldsymbol{Q_k'} &= E[\boldsymbol{H_{w,k} w_k (H_{w,k} w_k)^T}] \\ &= E[\boldsymbol{H_{w,k} w_k w_k^T H_{w,k}^T}] \\ &= \boldsymbol{H_{w,k}} E[\boldsymbol{w_k w_k^T}] \boldsymbol{H_{w,k}^T} \\ &= \boldsymbol{H_{w,k} Q_k H_{w,k}^T}\end{aligned}$$

A symmetrical argument applies for the exteroceptive sensor noise covariance, giving:

$$\boldsymbol{R_k^l}' = \boldsymbol{G_{n,k}^l R_k^l (G_{n,k}^l)}^T$$

Next we need to find a way to combine these exteroceptive measurements at each timestep into the general matrices $\boldsymbol{R_k'}$, $\boldsymbol{G_{x,k}}$ and the general measurement vector $\boldsymbol{y_k}$ to be used in the EKF based on the number of visible landmarks in that timestep. Say during a certain timestep k there are L visible landmarks, where a landmark l is considered visible if $r_k^l \neq 0$. Let the set of visible landmarks be $\{l_1, l_2, ..., l_L\}$. Then we can stack the measurement vectors like so:

$$\begin{bmatrix} \boldsymbol{y_k^{l_1}} \\ \boldsymbol{y_k^{l_2}} \\ \cdots \\ \boldsymbol{y_k^{l_L}} \end{bmatrix} = \begin{bmatrix} r_k^{l_1} \\ \phi_k^{l_1} \\ r_k^{l_2} \\ \phi_k^{l_2} \\ \cdots \\ r_k^{l_L} \\ \phi_k^{l_L} \end{bmatrix} = \begin{bmatrix} \boldsymbol{g_{l_1}(x_k, n_k^{l_1})} \\ \boldsymbol{g_{l_2}(x_k, n_k^{l_2})} \\ \cdots \\ \boldsymbol{g_{l_L}(x_k, n_k^{l_L})} \end{bmatrix}$$

We see that this forms a new general function of the exact same state but with a concatenated noise vector

$$\boldsymbol{y_k = g(x_k, n_k), n_k} = \begin{bmatrix} \boldsymbol{n_k^{l_1}} \\ \boldsymbol{n_k^{l_2}} \\ \cdots \\ \boldsymbol{n_k^{l_L}} \end{bmatrix}$$

Since it is a function of the exact same state, the Jacobian of this function with respect to the state will be the per-landmark Jacobians stacked in the same manner since the number of columns (aka the number of states) is the same:

$$\boldsymbol{G_{x,k}} = \begin{bmatrix} \boldsymbol{G_{x,k}^{l_1}} \\ \boldsymbol{G_{x,k}^{l_2}} \\ \cdots \\ \boldsymbol{G_{x,k}^{l_L}} \end{bmatrix}$$

On the other hand, the noise vector of this function is now higher in dimension, The Jacobian with respect to the exteroceptive sensor noise will see an equal increase in rows and columns L times, but will follow the same pattern as the per-landmark form since we are stacking along the diagonal. This means that given that

$$\boldsymbol{G^l_{n,k}} = \boldsymbol{1_{2\times 2}},$$

$$\boldsymbol{G_{n,k}} = \boldsymbol{1_{2L\times 2L}}$$

Similarly, given that the per-landmark exteroceptive sensor error covariance is

$$\boldsymbol{R^l_k} = \begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\phi^2 \end{bmatrix} \in \mathbb{R}^{2\times 2},$$

$$\boldsymbol{R_k} = diag(\boldsymbol{R_k^{l_1}, R_k^{l_2}, ..., R_k^{l_L}}) = diag(\sigma_r^2, \sigma_\phi^2, \sigma_r^2, \sigma_\phi^2, ..., \sigma_r^2, \sigma_\phi^2) \in \mathbb{R}^{2L\times 2L}$$

And the same relationships hold when taking into account the Jacobian $\boldsymbol{G_{n,k}}$ to obtain $\boldsymbol{R_k^{'}}$:

$$\boldsymbol{R_k^{'}} = \boldsymbol{G_{n,k} R_k G_{n,k}^T} = \boldsymbol{R_k}$$

Note that now we are not only assuming that the range errors and bearing errors are statistically independent, but that the errors for different landmarks are also statistically indepdendent, which is still reasonable since we already assume that the sensor measurements at different timesteps are statistically independent. Now we can derive the modified EKF equations for a given timestep with L visible landmarks:

Prediction:
$\boldsymbol{\check{P}_k} = \boldsymbol{H_{x,k-1}\hat{P}_{k-1}H_{x,k-1}^T} + \boldsymbol{Q_k^{'}}$
$\boldsymbol{\check{x}_k} = \boldsymbol{h(\hat{x}_{k-1}, u_k, 0)}$

Kalman Gain update:
$\boldsymbol{K_k} = \boldsymbol{\check{P}_k G_{x,k}^T (G_{x,k}\check{P}_k G_{x,k}^T} + \boldsymbol{R_k^{'}})^{-1}$

Correction:
$\boldsymbol{\hat{P}_k} = \boldsymbol{(1 - K_k G_{x,k})\check{P}_k}$
$\boldsymbol{\hat{x}_k} = \boldsymbol{\check{x}_k + K_k(y_k - g(\check{x}_k, 0))}$

When iterating from $k = 2...K$ timesteps, we initialize with $\boldsymbol{\hat{x}_1} = \boldsymbol{x_1}$, $\boldsymbol{\hat{P}_1} = diag(1, 1, 0.1)$ which will be changed to assign a more poor initial condition as will be seen later. From this initialization, all the steps stated above from 2 to K can be executed. Note that in implementation, states and measurements relating to angles will be normalized to remain in the interval $[-\pi, \pi]$ as seen in the MATLAB code attached in the Appendix.

4 Plots and Discussion

a.

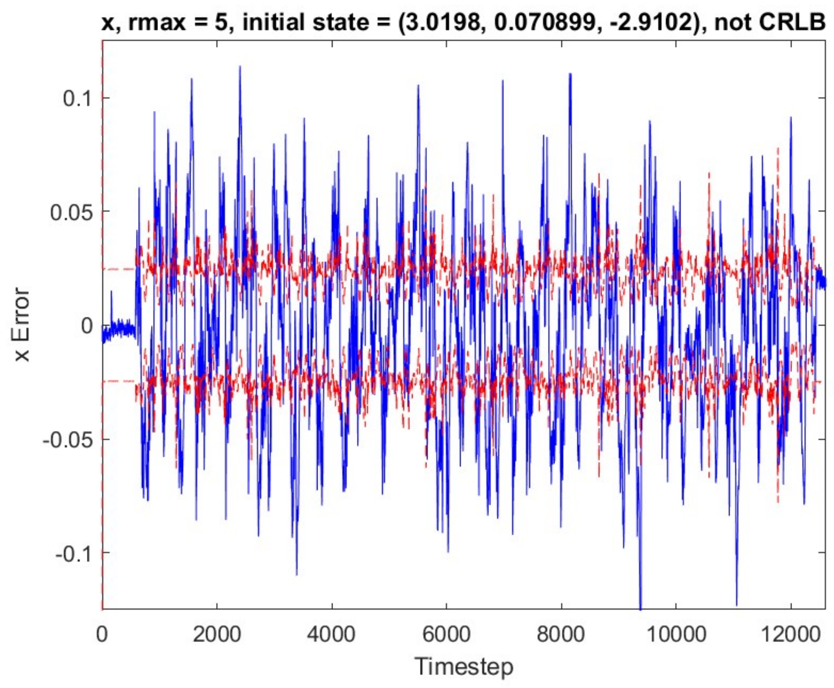

Figure 1: Error in x position vs. timestep with $r_{max} = 5$

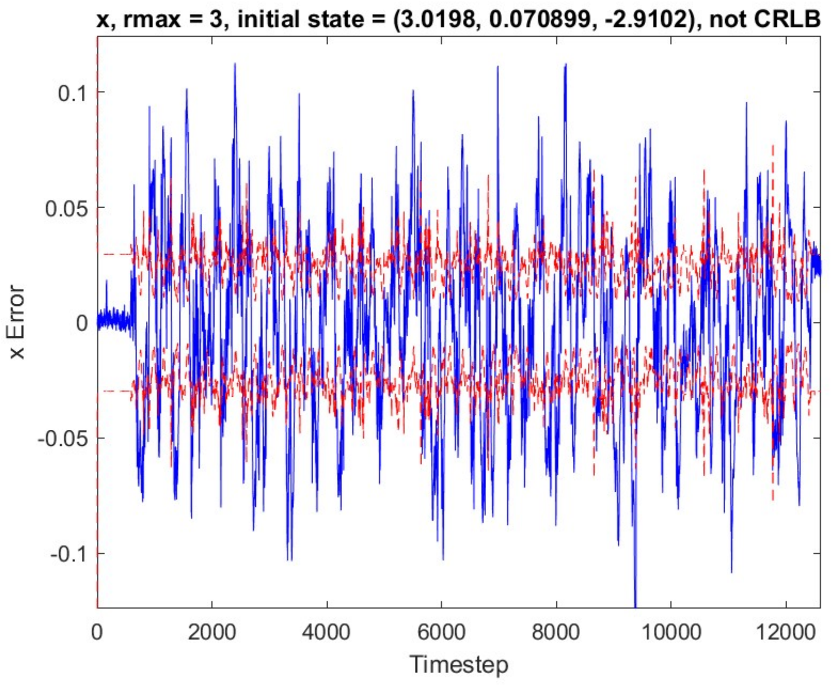

Figure 2: Error in x position vs. timestep with $r_{max} = 3$

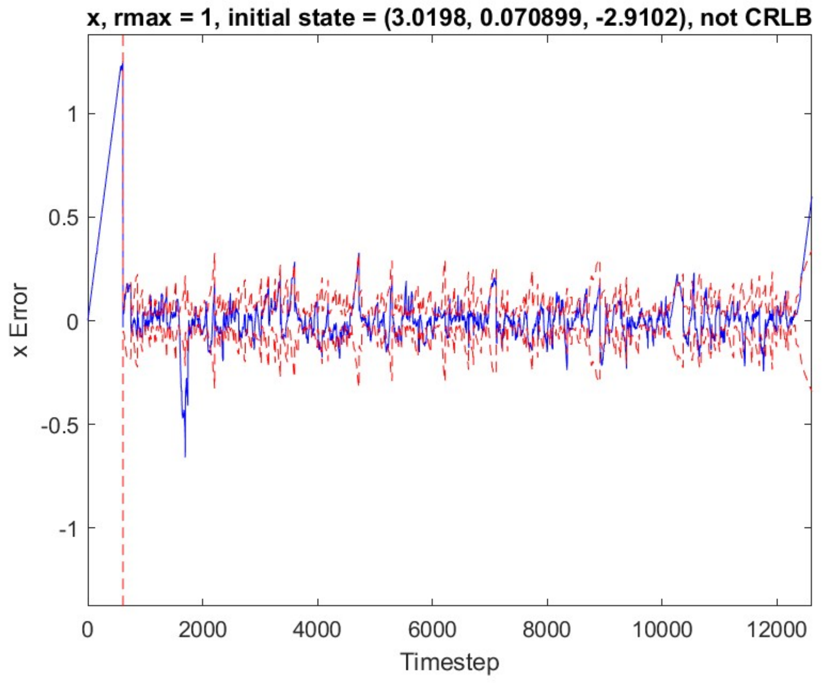

Figure 3: Error in x position vs. timestep with $r_{max} = 1$

The error hovers around zero, which is a good sign when aiming for an unbiased estimator. It also indicates that the estimator is indeed tracking the real x position to a good extent. However, at most timesteps the uncertainty envelope fails to contain the error, which indicates that the estimator is overconfident in its output, and that our noise covariance matrices do not encapsulate all the error in the experiment. Finally, as r_{max} is varied, there is not a big difference going from 5 to 3, but when going down to 1, there is a large spike in initial error before convergence. This is likely because this threshold is very strict, resulting in using very little landmarks to correct the position. This likely makes the estimator take longer to converge for more poor initial conditions as well (will be seen later).

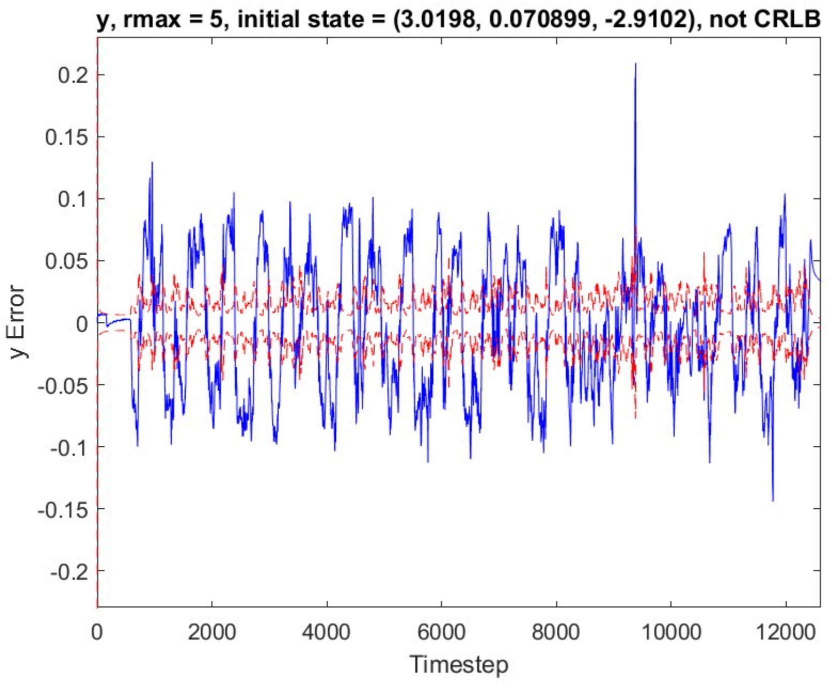

Figure 4: Error in y position vs. timestep with $r_{max} = 5$

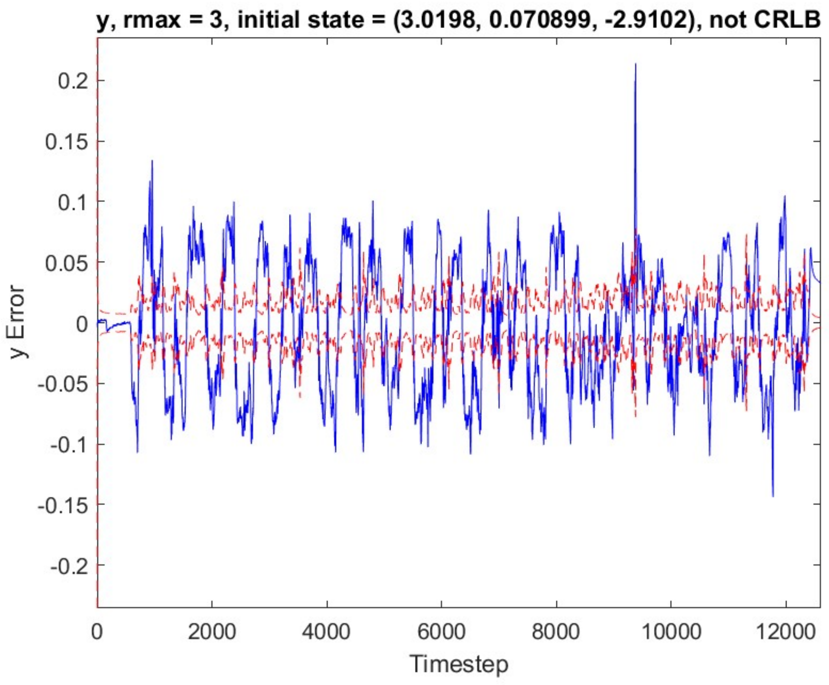

Figure 5: Error in y position vs. timestep with $r_{max} = 3$

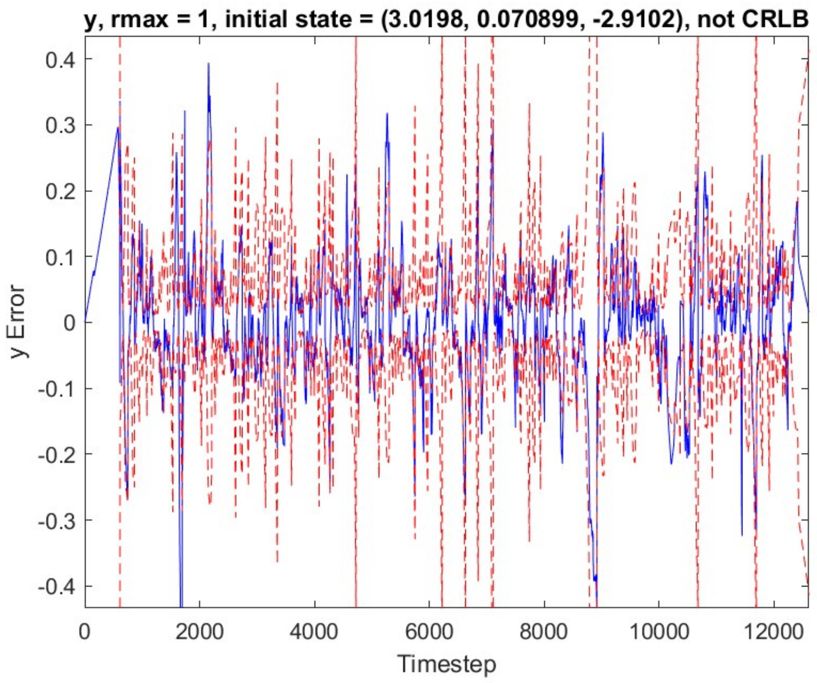

Figure 6: Error in y position vs. timestep with $r_{max} = 1$

Generally speaking, the y position follows the same trends as the x position, but with faster convergence especially in the $r_{max} = 1$ case. Furthermore, in the $r_{max} = 1$ case, the uncertainty envelope is doing a better job of encapsulating the error. Since lower r_{max} results in less correction from landmarks, this could indicate that the covariance related to the exteroceptive sensor noise models the dynamics of the system more poorly in comparison to the chosen covariance for the process noise.

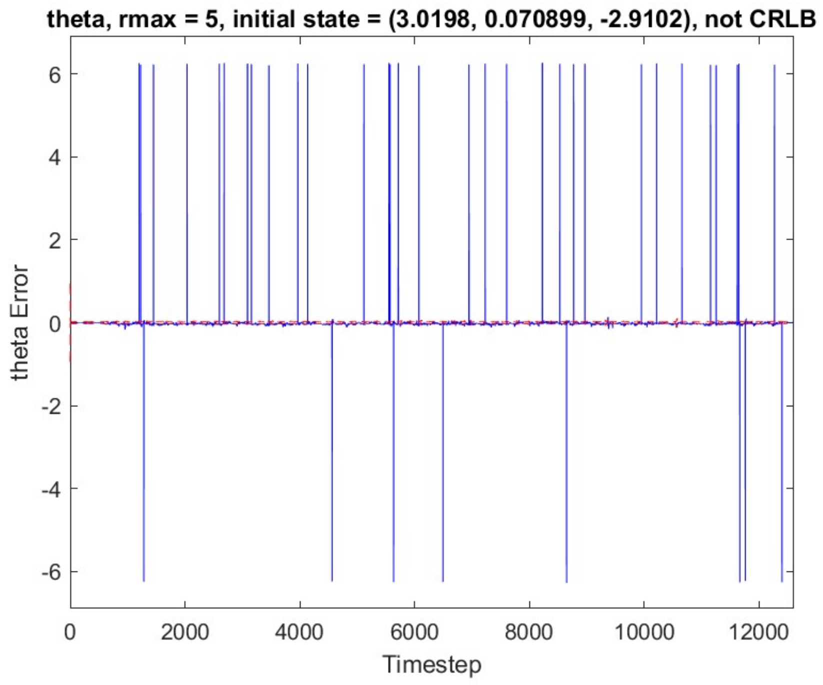

Figure 7: Error in θ vs. timestep with $r_{max} = 5$

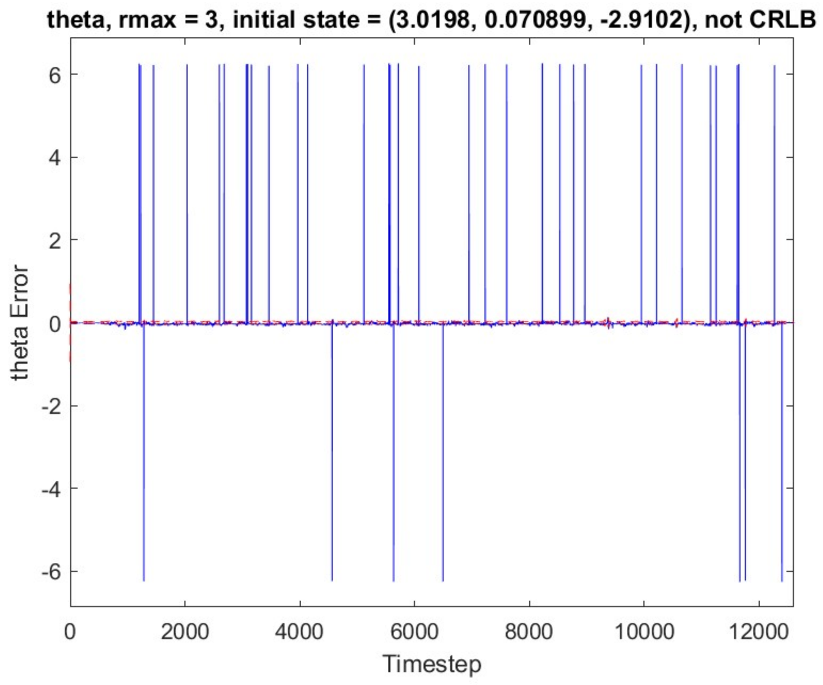

Figure 8: Error in θ vs. timestep with $r_{max} = 3$

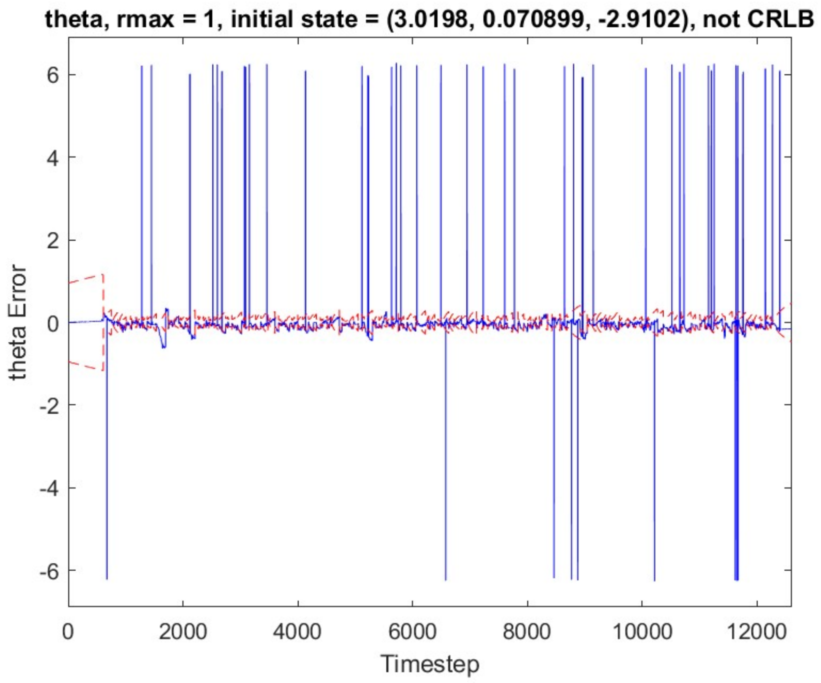

Figure 9: Error in θ vs. timestep with $r_{max} = 1$

θ also follows the same general trend, but with the addition that there are extremely large but short-duration periodic spikes in error despite the angles being normalized (the error is much higher across the board when the angles are unnormalized). These spikes shoot far above the uncertainty envelope, but only last for one or two timesteps. These spikes are likely due to the system's states and Jacobians being the most sensitive to θ by far, meaning a small change in θ could cause a large change in system dynamics that the estimator needs to react to and converge. The large change in state also introduces linearization error because the approximation becomes less accurate as we deviate from the operating point. Perhaps the Iterated Extended Kalman Filter (IEKF) could help mitigate this issue.

b.

Repeating the previous part with a poor initial condition, we have:

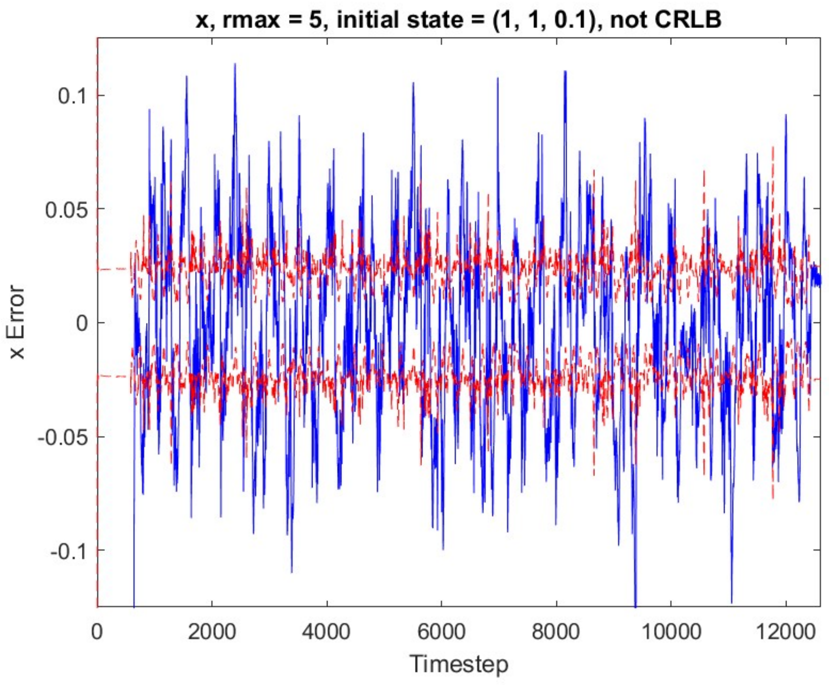

Figure 10: Error in x position vs. timestep with $r_{max} = 5$ (poor initial condition)

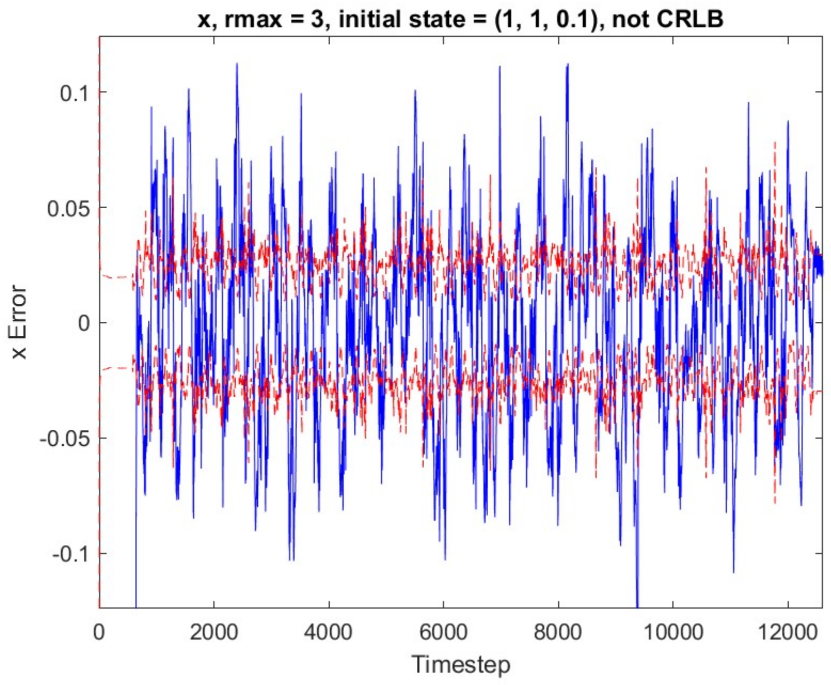

Figure 11: Error in x position vs. timestep with $r_{max} = 3$ (poor initial condition)

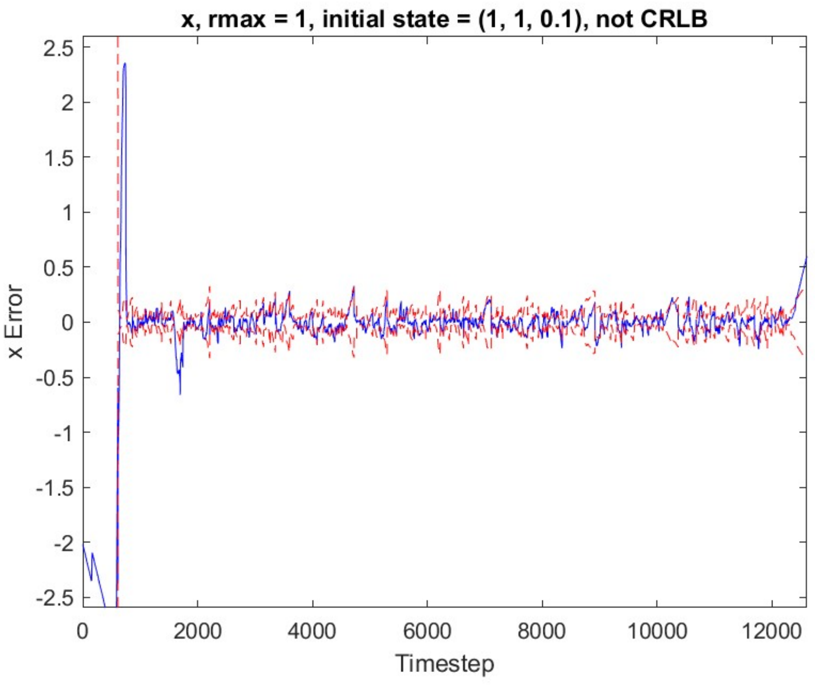

Figure 12: Error in x position vs. timestep with $r_{max} = 1$ (poor initial condition)

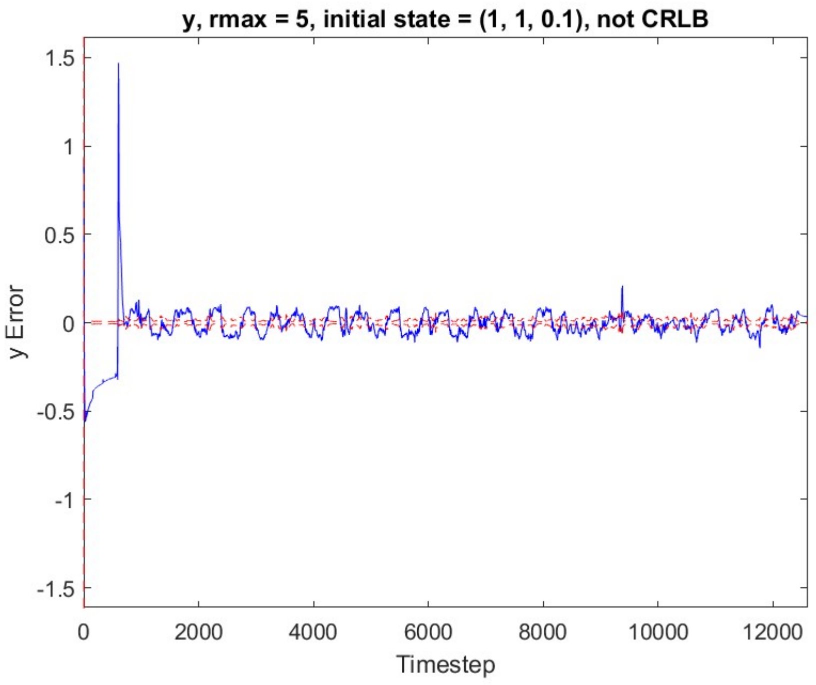

Figure 13: Error in y position vs. timestep with $r_{max} = 5$ (poor initial condition)

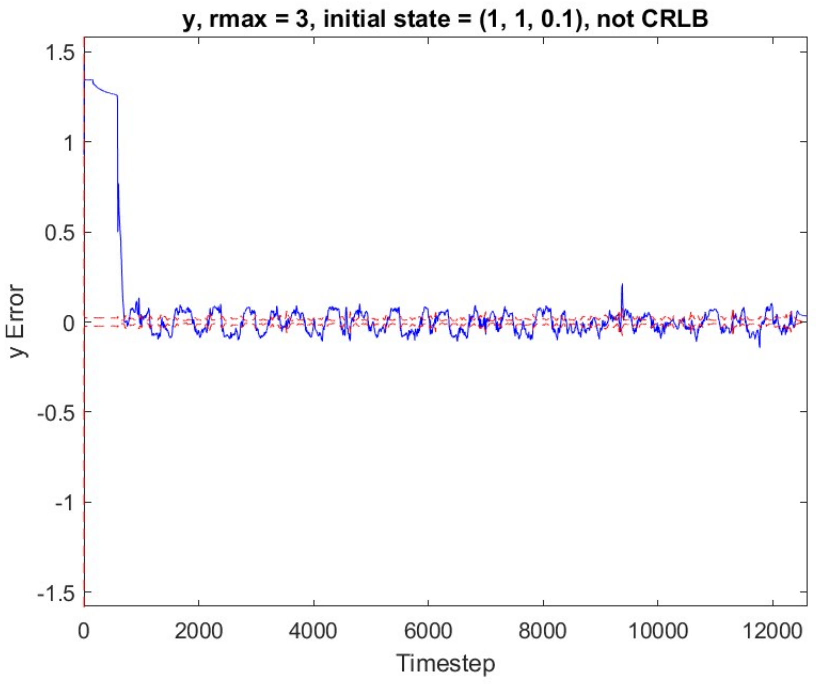

Figure 14: Error in y position vs. timestep with $r_{max} = 3$ (poor initial condition)

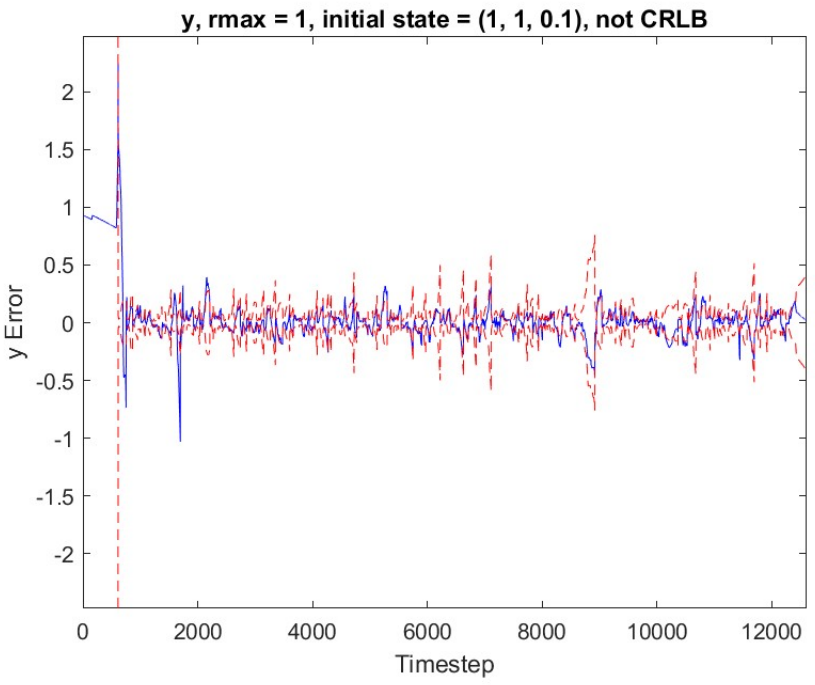

Figure 15: Error in y position vs. timestep with $r_{max} = 1$ (poor initial condition)

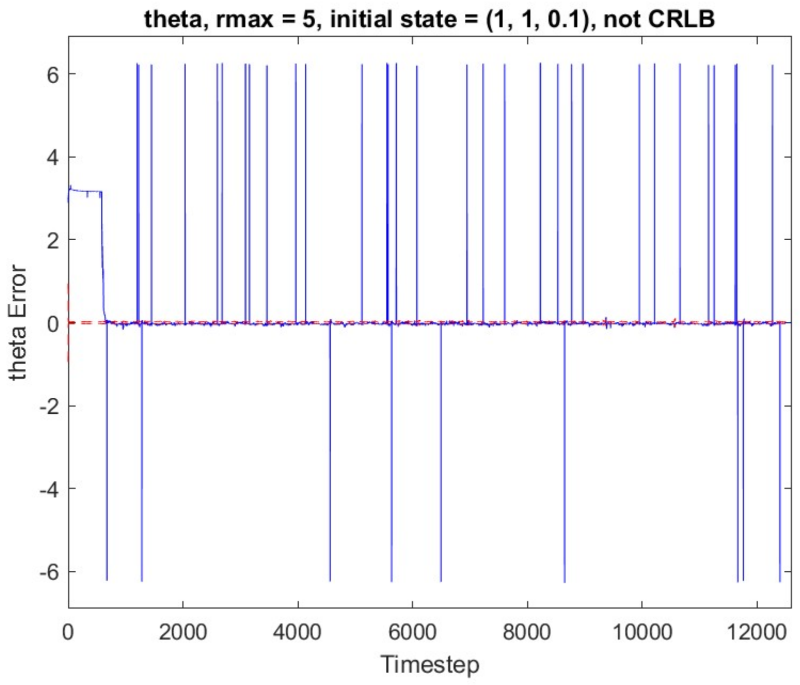

Figure 16: Error in θ vs. timestep with $r_{max} = 5$ (poor initial condition)

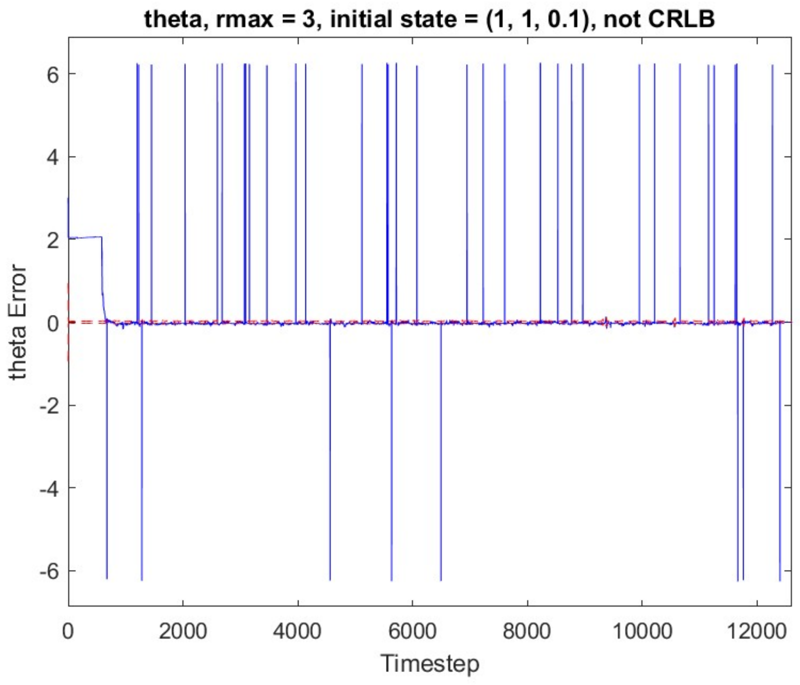

Figure 17: Error in θ vs. timestep with $r_{max} = 3$ (poor initial condition)

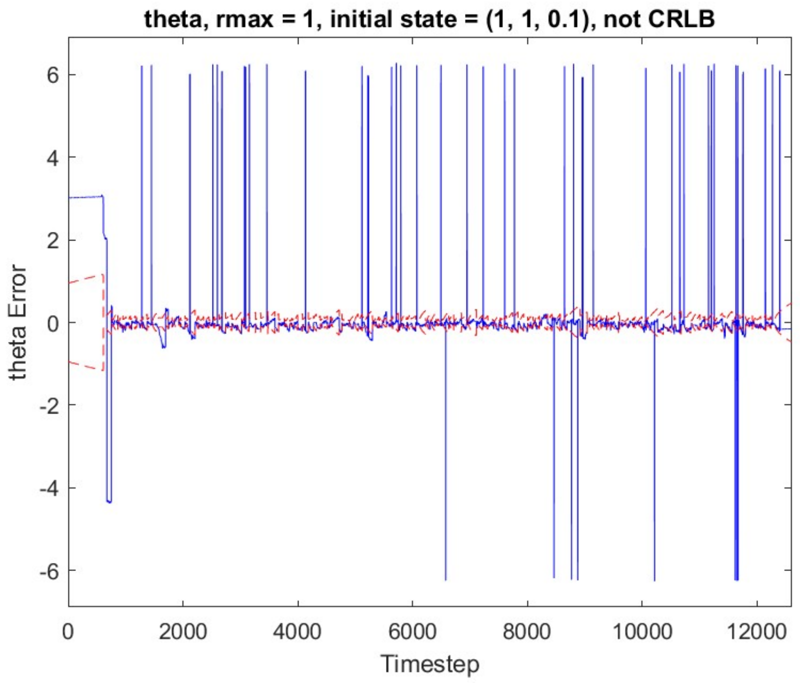

Figure 18: Error in θ vs. timestep with $r_{max} = 1$ (poor initial condition)

The main difference we see here from the previous section is significantly larger error in the earlier timesteps, however the estimator eventually converges. We also see the largest error in the early timesteps when $r_{max} = 1$ as hypothesized in the previous part, since using less landmarks for correction can lead to more sensitivity to initial condition. The uncertainty envelope still struggles to encase the error when $r_{max} \neq 1$ and the spikes with θ still persist. The easiest way to deviate from the best initial condition is to change θ. Trying the initial condition $[1, 1, -\pi]$ actually still yields convergence.

c.

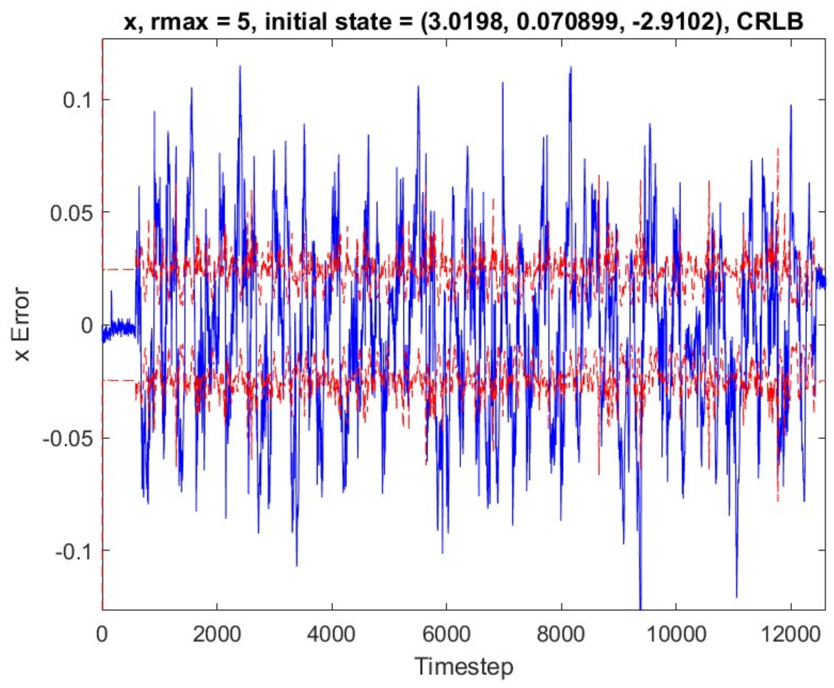

Figure 19: Error in x position vs. timestep with $r_{max} = 5$ (CRLB)

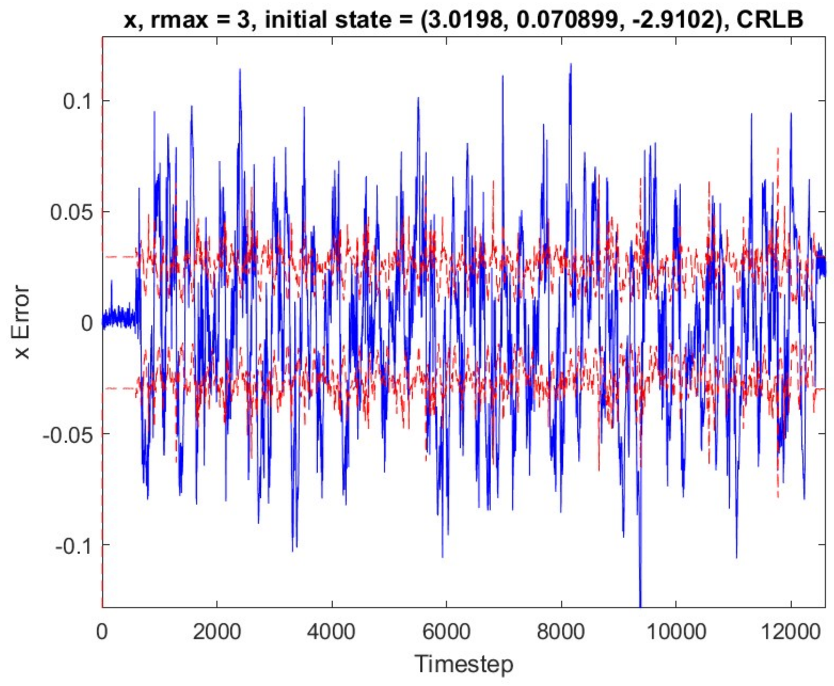

Figure 20: Error in x position vs. timestep with $r_{max} = 3$ (CRLB)

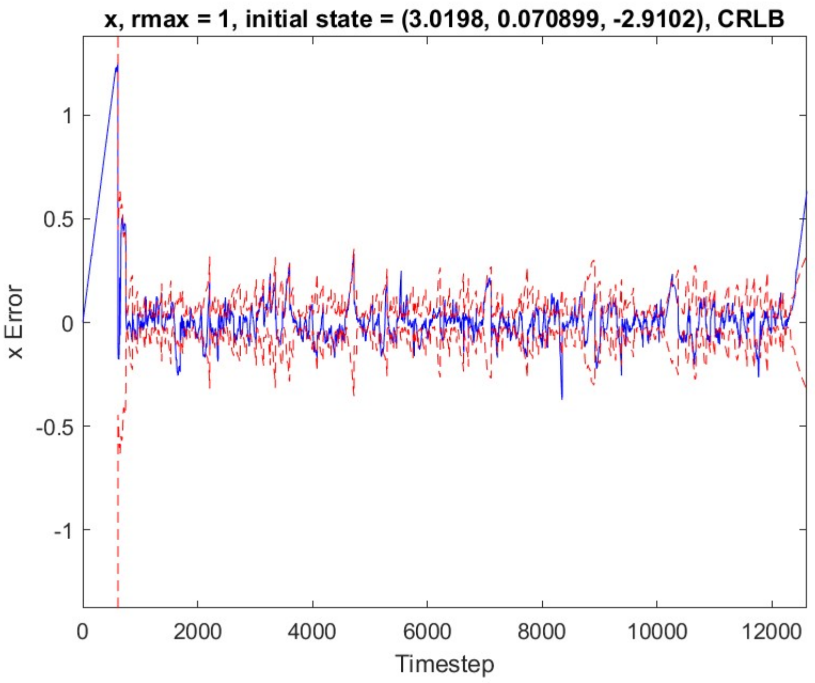

Figure 21: Error in x position vs. timestep with $r_{max} = 1$ (CRLB)

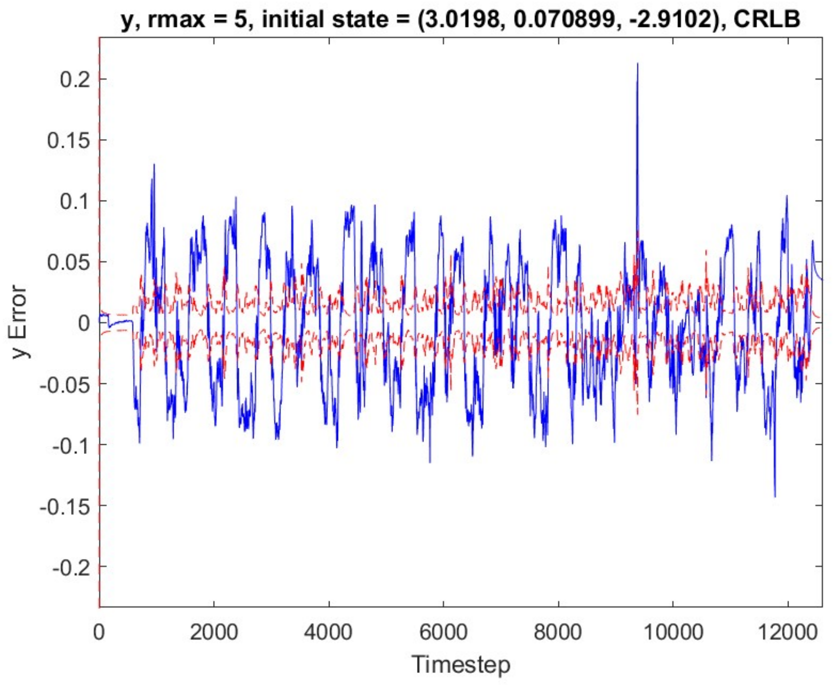

Figure 22: Error in y position vs. timestep with $r_{max} = 5$ (CRLB)

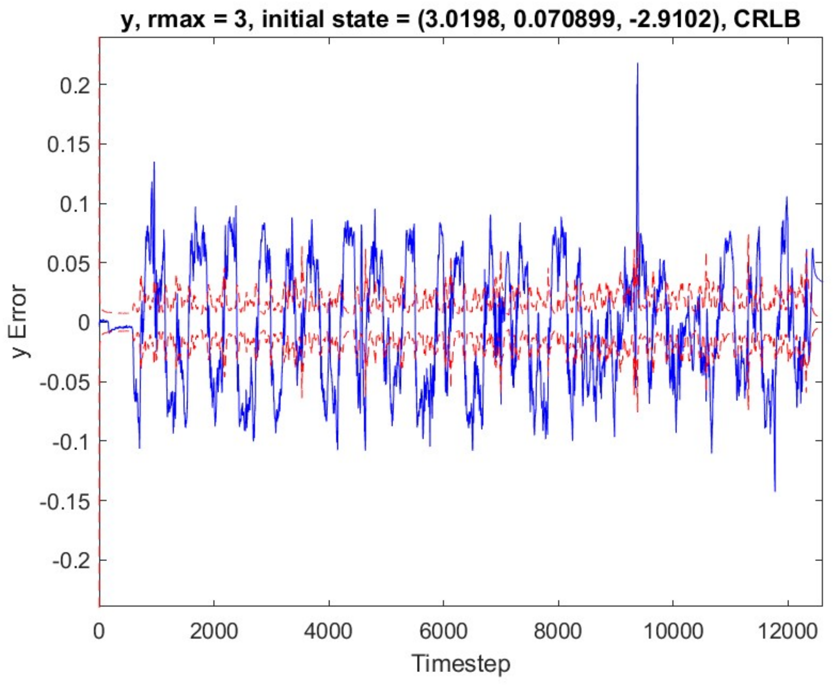

Figure 23: Error in y position vs. timestep with $r_{max} = 3$ (CRLB)

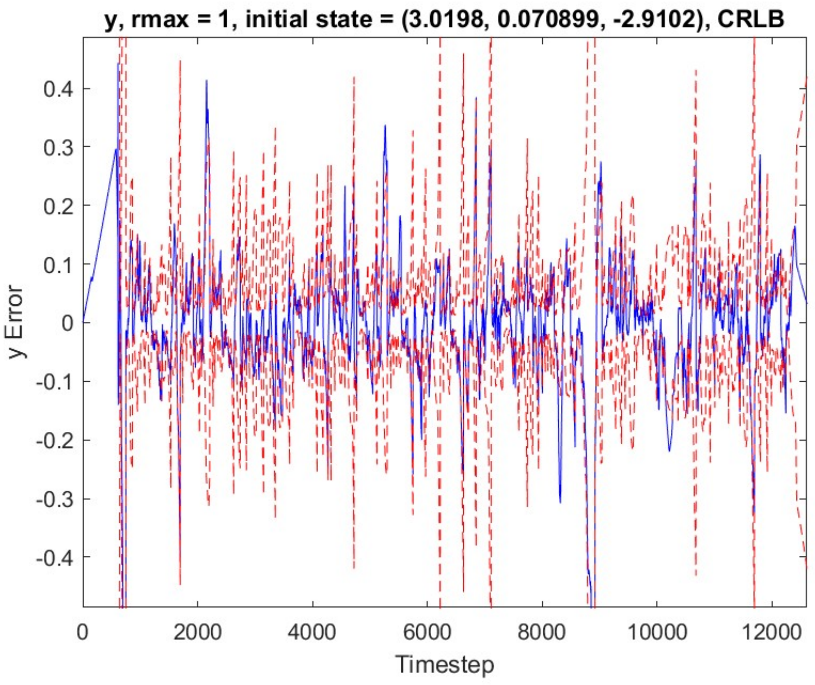

Figure 24: Error in y position vs. timestep with $r_{max} = 1$ (CRLB)

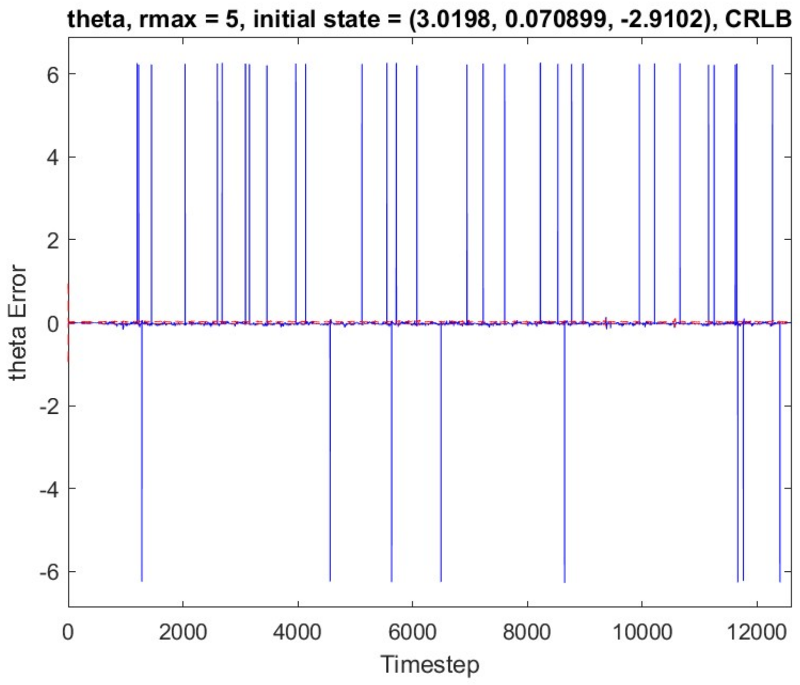

Figure 25: Error in θ vs. timestep with $r_{max} = 5$ (CRLB)

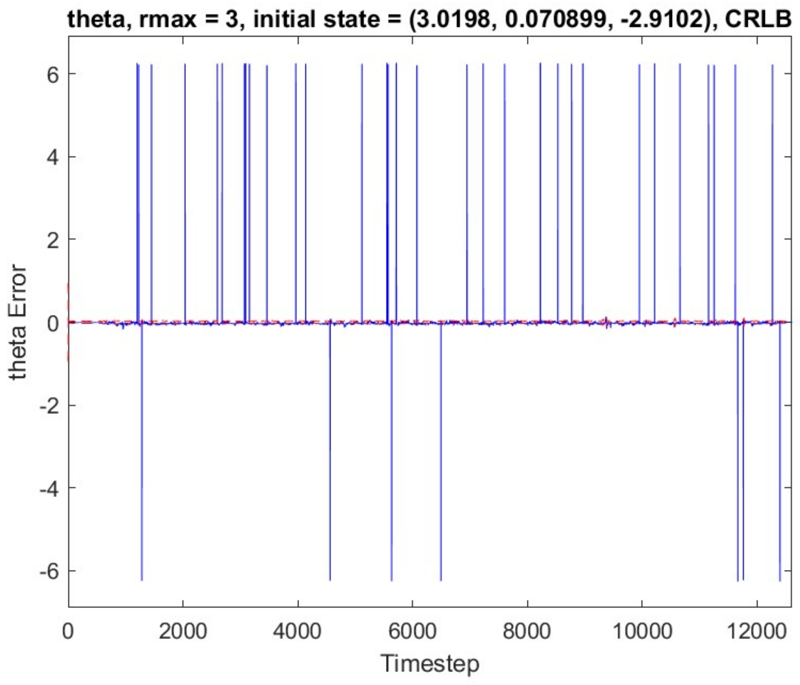

Figure 26: Error in θ vs. timestep with $r_{max} = 3$ (CRLB)

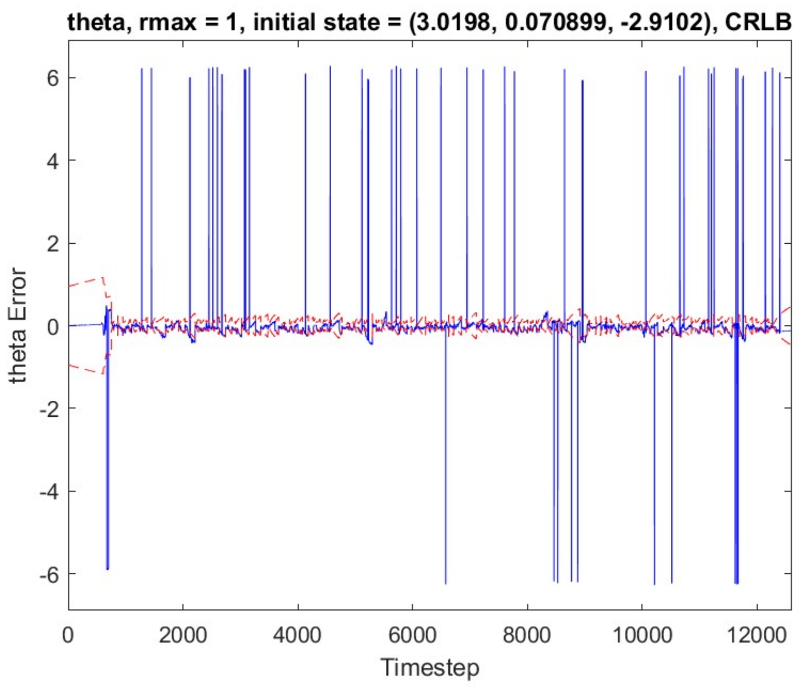

Figure 27: Error in θ vs. timestep with $r_{max} = 1$ (CRLB)

This is not too far off from the original results, indicating that the EKF is already performing near the CRLB when converging in this specific system.

5 Animation

The animation is attached separately. Please note I used a green ellipse for the covariance ellipse to more clearly illustrate the times at which the estimator is overconfident and the ellipse is smaller than the robot's marker. Please note that ChatGPT was used in the animation code in order to do plot labeling and modify the refresh rate of the animation to properly encapsulate the key timesteps, however the code was also modified and polished to meet assignment instructions.

A MATLAB CODE

```
% Load dataset
load dataset2.mat

% Symbolically Obtaining Jacobians
sympref('AbbreviateOutput',false);
% Motion Model Variables
x_prev_sym = sym("x_k_minus_1");
y_prev_sym = sym("y_k_minus_1");
theta_prev_sym = sym("theta_k_minus_1");
x_prev_hat_sym = sym("x__hat_k_minus_1");
y_prev_hat_sym = sym("y__hat_k_minus_1");
theta_prev_hat_sym = sym("theta__hat_k_minus_1");
T_sym = sym("T");
v_sym = sym("v_k");
omega_sym = sym("omega_k");
w_v_sym = sym("w_k_v");
w_omega_sym = sym("w_k_omega");

% Observation Model Variables
x_l_sym = sym("x_l", "real");
y_l_sym = sym("y_l", "real");
x_sym = sym("x_k", "real");
y_sym = sym("y_k", "real");
theta_sym = sym("theta_k", "real");
x_check_sym = sym("x__v_k", "real");
y_check_sym = sym("y__v_k", "real");
theta_check_sym = sym("theta__v_k", "real");
d_sym = sym("d", "real");
n_r_sym = sym("n__l_k_r", "real");
n_phi_sym = sym("n__l_k_phi", "real");


% Motion Model
h = [x_prev_sym; y_prev_sym; theta_prev_sym] + ...
    T_sym * [cos(theta_prev_sym) 0; sin(theta_prev_sym) 0; 0 1] ...
    * ([v_sym; omega_sym] + [w_v_sym; w_omega_sym])

H_x_pure = jacobian(h, [x_prev_sym;y_prev_sym;theta_prev_sym])
H_w_pure = jacobian(h, [w_v_sym;w_omega_sym])

old_variables_h = [x_prev_sym;y_prev_sym;theta_prev_sym; ...
    w_v_sym;w_omega_sym];
new_variables_h = [x_prev_hat_sym;y_prev_hat_sym; ...
    theta_prev_hat_sym;0;0];

H_x = subs(H_x_pure, old_variables_h, new_variables_h)
H_w = subs(H_w_pure, old_variables_h, new_variables_h)

% Observation Model
g1 = sqrt((x_l_sym - x_sym - d_sym * cos(theta_sym))^2 + ...
    (y_l_sym - y_sym - d_sym * sin(theta_sym))^2) + n_r_sym;
g2 = atan2(y_l_sym - y_sym - d_sym * sin(theta_sym), ...
    x_l_sym - x_sym - d_sym * cos(theta_sym)) - theta_sym + n_phi_sym;
g = [g1;g2]

G_x_pure = jacobian(g, [x_sym;y_sym;theta_sym])
G_n_pure = jacobian(g, [n_r_sym;n_phi_sym])

old_variables_g = [x_sym;y_sym;theta_sym;n_r_sym;n_phi_sym];
new_variables_g = [x_check_sym;y_check_sym;theta_check_sym;0;0];

G_x = subs(G_x_pure, old_variables_g, new_variables_g)
G_n = subs(G_n_pure, old_variables_g, new_variables_g)

% Process data
K = size(t, 1);
T = 1 / 10;
Q_k = diag([v_var, om_var]);
R_k = diag([r_var, b_var]);

% Used for evaluating Jacobians and motion/observation models
% No need for noise jacobian for observation model since it is always
% the identity matrix (the noise is already linear)
h_func = matlabFunction(h);
H_x_func = matlabFunction(H_x_pure);
H_w_func = matlabFunction(H_w_pure);

g_func = matlabFunction(g);
G_x_func = matlabFunction(G_x_pure);

% Running trials and plotting
time = 1:K;
r_max_vals = [5, 3, 1];
initial_states = zeros(3,1,4);
initial_states(:,:,1) = [x_true(1);y_true(1);th_true(1)];
initial_states(:,:,2) = [1;1;0.1];
initial_states(:,:,3) = [x_true(1);y_true(1);th_true(1)];
% EXTREMELY poor initial condition
initial_states(:,:,4) = [1;1;-pi];
initial_covs = zeros(3,3,3);
initial_covs(:,:,1) = diag([1;1;0.1]);
initial_covs(:,:,2) = diag([1;1;0.1]);
initial_covs(:,:,3) = diag([1;1;0.1]);
initial_covs(:,:,4) = diag([1;1;0.1]);

idx = 1;
for i = 1:size(initial_states,3)
    for j = 1:size(r_max_vals,2)
        if i == size(initial_states,3) - 1
            use_CRLB=true;
        else
            use_CRLB=false;
        end
         [x_hat, y_hat, theta_hat, x_var, y_var, theta_var] = EKF( ...
            x_true,y_true,th_true, K, T, d, v, om, l, r, b, Q_k, R_k, h_func, ...
            H_x_func, H_w_func, g_func, G_x_func, r_max_vals(j), ...
```

```
   initial_states(:,:,i), initial_covs(:,:,i), use_CRLB);
x_error = x_hat - x_true;
y_error = y_hat - y_true;
theta_error = theta_hat - th_true;

% Plot x
figure;
plot(time, x_error, 'b-');
hold on;
plot(time, 3 * sqrt(x_var), 'r--');
plot(time, -3 * sqrt(x_var), 'r--');
if use_CRLB
    title( ['x, rmax = ' num2str(r_max_vals(j)) ...
        ', initial state = (' num2str(initial_states(1,1,i)) ...
        ', ' num2str(initial_states(2,1,i)), ', ' ...
        num2str(initial_states(3,1,i)) '), CRLB'])
else
    title( ['x, rmax = ' num2str(r_max_vals(j)) ...
        ', initial state = (' num2str(initial_states(1,1,i)) ...
        ', ' num2str(initial_states(2,1,i)), ', ' ...
        num2str(initial_states(3,1,i)) '), not CRLB'])
end
xlabel("Timestep")
ylabel("x Error")
y_boundary = max(x_error);
ylim([1.1 * -y_boundary, 1.1 * y_boundary])
xlim([0, K])
saveas(gcf, ['f_x_' num2str(idx) '.jpg']);
hold off;

% Plot y
figure;
plot(time, y_error, 'b-');
hold on;
plot(time, 3 * sqrt(y_var), 'r--');
plot(time, -3 * sqrt(y_var), 'r--');
if use_CRLB
    title( ['y, rmax = ' num2str(r_max_vals(j)) ...
        ', initial state = (' num2str(initial_states(1,1,i)) ...
        ', ' num2str(initial_states(2,1,i)), ', ' ...
        num2str(initial_states(3,1,i)) '), CRLB'])
else
    title( ['y, rmax = ' num2str(r_max_vals(j)) ...
        ', initial state = (' num2str(initial_states(1,1,i)) ...
        ', ' num2str(initial_states(2,1,i)), ', ' ...
        num2str(initial_states(3,1,i)) '), not CRLB'])
end
xlabel("Timestep")
ylabel("y Error")
y_boundary = max(y_error);
ylim([1.1 * -y_boundary, 1.1 * y_boundary])
xlim([0, K])
saveas(gcf, ['f_y_' num2str(idx) '.jpg']);
hold off;
```

```
        % Plot theta
        figure;
        plot(time, theta_error, 'b-');
        hold on;
        plot(time, 3 * sqrt(theta_var), 'r--');
        plot(time, -3 * sqrt(theta_var), 'r--');
        if use_CRLB
            title( ['theta, rmax = ' num2str(r_max_vals(j)) ...
                ', initial state = (' num2str(initial_states(1,1,i)) ...
                ', ' num2str(initial_states(2,1,i)), ', ' ...
                num2str(initial_states(3,1,i)) '), CRLB'])
        else
            title( ['theta, rmax = ' num2str(r_max_vals(j)) ...
                ', initial state = (' num2str(initial_states(1,1,i)) ...
                ', ' num2str(initial_states(2,1,i)), ', ' ...
                num2str(initial_states(3,1,i)) '), not CRLB'])
        end
        xlabel("Timestep")
        ylabel("theta Error")
        y_boundary = max(theta_error);
        ylim([1.1 * -y_boundary, 1.1 * y_boundary])
        xlim([0, K])
        saveas(gcf, ['f_theta_' num2str(idx) '.jpg']);
        idx = idx + 1;
        hold off;
    end
end



% Animation
[x_hat, y_hat, theta_hat, x_var, y_var, theta_var] = EKF( ...
    x_true,y_true,th_true, K, T, d, v, om, l, r, b, Q_k, R_k, h_func, ...
    H_x_func, H_w_func, g_func, G_x_func, 1, initial_states(:,:,1), ...
    initial_covs(:,:,1), false);

dot_size = 25;
pause_time = 0.0005;
sampling_factor = 13;

figure('Position', [100, 100, 800, 600]);
hold on;
axis equal;
xlim([min(x_true) - 1, max(x_true) + 1]);
ylim([min(y_true) - 1, max(y_true) + 1]);
title('Robot Movement: Estimated vs. Ground Truth');
xlabel('X Position');
ylabel('Y Position');

% Plot landmarks
scatter(l(:,1), l(:,2), dot_size, 'k', 'filled', 'DisplayName', 'Landmarks');

% Scatter plots for the current position
```

```
true_pos = scatter(x_true(1), y_true(1), dot_size, 'b', 'filled', ...
    'DisplayName', 'True Position');
est_pos = scatter(x_hat(1), y_hat(1), dot_size, 'r', 'filled', 'DisplayName', ...
    'Estimated Position');

% Initialize covariance ellipse
cov_ellipse = plot(0, 0, 'g', 'LineWidth', 1.5, 'DisplayName', ...
    '3-Sigma Covariance Ellipse');

legend('Location', 'northeastoutside');

% Animation loop
for k = 1:sampling_factor:K
    % Update positions
    true_pos.XData = x_true(k);
    true_pos.YData = y_true(k);
    est_pos.XData = x_hat(k);
    est_pos.YData = y_hat(k);

    % Update covariance ellipse parameters
    cov_matrix = [x_var(k) 0; 0 y_var(k)] * 9;
    % 3-sigma is 9 times the variance
    theta = linspace(0, 2*pi, 100);
    % Transform circle to ellipse
    ellipse_points = sqrtm(cov_matrix) * [cos(theta); sin(theta)];
    cov_ellipse.XData = x_hat(k) + ellipse_points(1, :);
    cov_ellipse.YData = y_hat(k) + ellipse_points(2, :);

    % Refresh the plot
    drawnow;
    pause(pause_time);
end

% EKF IMPLEMENTATION
% CRLB is a boolean that dictates whether we evaluate the jacobians at the
% true state (true) or the previous estimated state (false)
function [x_hat, y_hat, theta_hat, x_var, y_var, theta_var] = EKF( ...
    x_true,y_true,th_true, K, T, d, v, om, l, r, b, Q_k, R_k, h_func, ...
    H_x_func, H_w_func, g_func, G_x_func, r_max, initial_state, initial_cov, ...
    CRLB)

    % Initializing EKF
    predicted_cov = zeros(3,3,K);
    predicted_state = zeros(3,1,K);
    estimated_cov = zeros(3,3,K);
    estimated_state = zeros(3,1,K);
    x_hat = zeros(K, 1);
    x_var = zeros(K, 1);
    y_hat = zeros(K, 1);
    y_var = zeros(K, 1);
    theta_hat = zeros(K, 1);
    theta_var = zeros(K, 1);

    % Initial condition
```

```
estimated_cov(:,:,1) = initial_cov;
estimated_state(:,:,1) = initial_state;
predicted_cov(:,:,1) = estimated_cov(1);
predicted_state(:,:,1) = estimated_state(1);

x_hat(1) = estimated_state(1,1,1);
y_hat(1) = estimated_state(2,1,1);
theta_hat(1) = estimated_state(3,1,1);
x_var(1) = estimated_cov(1,1,1);
y_var(1) = estimated_cov(2,2,1);
theta_var(1) = estimated_cov(3,3,1);

% EKF
for k = 2:K
    % Evaluate Jacobian and motion model at previous estimated state
    % and current inputs with 0 noise
    x_prev = estimated_state(1,1,k-1);
    y_prev = estimated_state(2,1,k-1);
    theta_prev = estimated_state(3,1,k-1);
    % Predictor
    if CRLB
        H_x = H_x_func(T,normalize_angle(th_true(k-1)),v(k),0);
        H_w = H_w_func(T,normalize_angle(th_true(k-1)));
    else
        H_x = H_x_func(T,theta_prev,v(k),0);
        H_w = H_w_func(T,theta_prev);
    end
    Q = H_w * Q_k * H_w.';
    predicted_cov(:,:,k) = H_x * estimated_cov(:,:,k-1) * H_x.' + Q;
    predicted_state(:,:,k) = h_func(T, om(k), theta_prev, v(k), 0, 0, ...
        x_prev, y_prev);
    % normalize theta between -pi and pi
    predicted_state(3,1,k) = normalize_angle(predicted_state(3,1,k));
    % Building up y, G, and R based on visible landmarks
    x_curr = predicted_state(1,1,k);
    y_curr = predicted_state(2,1,k);
    theta_curr = predicted_state(3,1,k);
    L = 0;
    % Finding number of visible landmarks
    for i = 1:size(r, 2)
        % landmark seen
        if r(k,i) ~= 0 && r(k,i) < r_max
            L = L + 1;
        end
    end
    % Building y, y_pred, and G_x (State Jacobian)
    idx = 1;
    G_x = zeros(2*L, 3);
    y = zeros(2*L, 1);
    y_pred = zeros(2*L, 1);
    for i = 1:size(r, 2)
        % landmark seen
        if r(k,i) ~= 0 && r(k,i) < r_max
            % y
```

```
                y(2*idx-1) = r(k,i);
                % normalize phi between -pi and pi
                y(2*idx) = normalize_angle(b(k,i));
                % current landmark position
                x_l_curr = l(i,1);
                y_l_curr = l(i,2);
                % y_pred
                prediction = g_func(d,0,0,theta_curr,x_curr,x_l_curr,y_curr, ...
                    y_l_curr);
                y_pred(2*idx-1) = prediction(1);
                % normalize predicted phi between -pi and pi
                y_pred(2*idx) = normalize_angle(prediction(2));
                % State Jacobian
                if CRLB
                    G_x_l = G_x_func(d,normalize_angle(th_true(k)),x_true(k), ...
                        x_l_curr,y_true(k),y_l_curr);
                else
                    G_x_l = G_x_func(d,theta_curr,x_curr,x_l_curr,y_curr, ...
                        y_l_curr);
                end
                G_x(2*idx - 1, :) = G_x_l(1, :);
                G_x(2*idx,:) = G_x_l(2, :);
                % increment
                idx = idx + 1;
            end
        end
        % Building R (covariance matrix)
        R = zeros(2*L);
        for i = 1:L
            R(2*i-1, 2*i-1) = R_k(1,1);
            R(2*i, 2*i) = R_k(2,2);
        end
        % Kalman Gain
        K_gain = predicted_cov(:,:,k) * G_x.' / (G_x * predicted_cov(:,:,k) * ...
            G_x.' + R);
        % Corrector
        estimated_cov(:,:,k) = (eye(size(K_gain * G_x)) - K_gain * G_x) ...
            * predicted_cov(:,:,k);
        estimated_state(:,:,k) = predicted_state(:,:,k) + K_gain * (y - y_pred);
        % normalize estimated theta between -pi and pi
        estimated_state(3,1,k) = normalize_angle(estimated_state(3,1,k));
        % Partitioning findings into final arrays
        x_hat(k) = estimated_state(1,1,k);
        y_hat(k) = estimated_state(2,1,k);
        theta_hat(k) = estimated_state(3,1,k);

        estimation_var = diag(estimated_cov(:,:,k));
        x_var(k) = estimation_var(1);
        y_var(k) = estimation_var(2);
        theta_var(k) = estimation_var(3);
    end
end

% Used to ensure landmark and position bearings remain within the range -pi
% to pi
function theta = normalize_angle(in)
    theta = in;
    if theta > pi
        theta = theta - 2*pi;
    elseif theta < -pi
        theta = theta + 2*pi;
    end
end
```