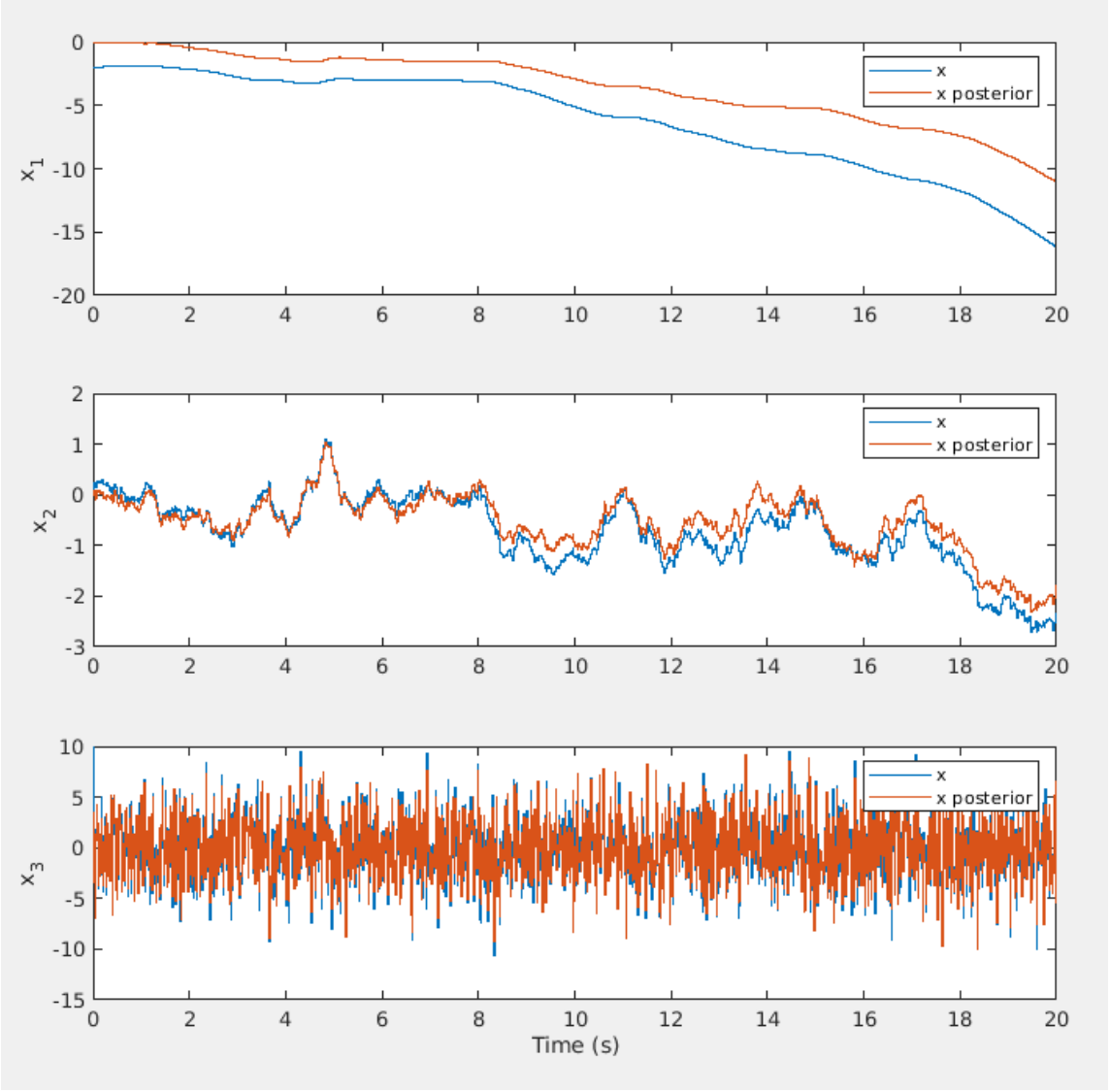
(a)

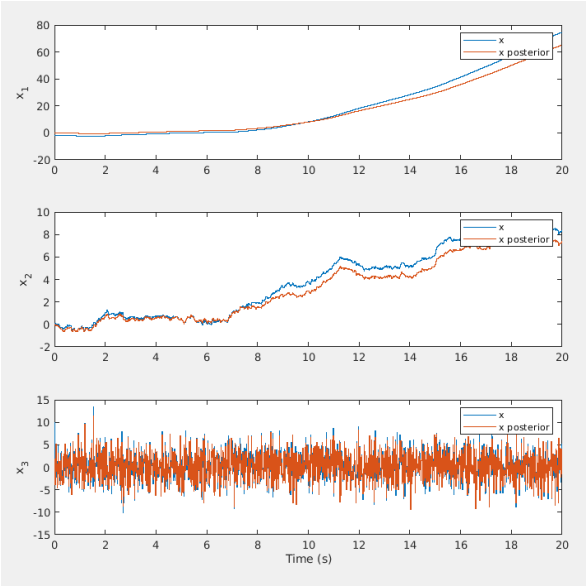
Choose appropriate initial conditions x(0|0) and P (0|0) for the filter. At t = 0 the robot head rests against a stop, which corresponds to a starting angle of (0 ± 5)◦ .

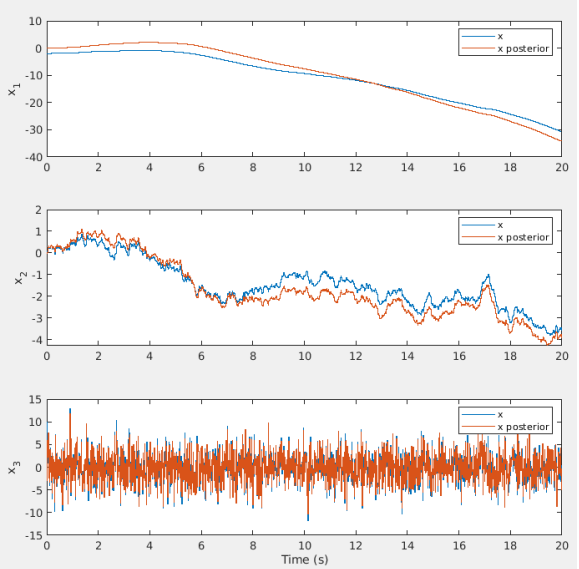
Given that the head is at rest, it is safe to assume that the angle is between -5 and 5 degrees. 0 was assumed for this.
Given that it is at rest, it is also reasonable to assume that the angular velocity is 0, given the sensor has a variance of 0.5, a value between -0.707 and 0.707 is reasonable.
For the acceleration, given that it's at rest, a reasonable assumption is that there is no net acceleration on the head. So 0 was assumed for this also.

In terms of the variance estimates, a small one of all 0's was used to demonstrate total trust in the guess along with a higher one of all 5's to show no trust in the guess.
The below figure shows the operation with one sensor.



While one sensor allows the kalman filter to follow the measured value, it is not accurate and deviates further as time progresses



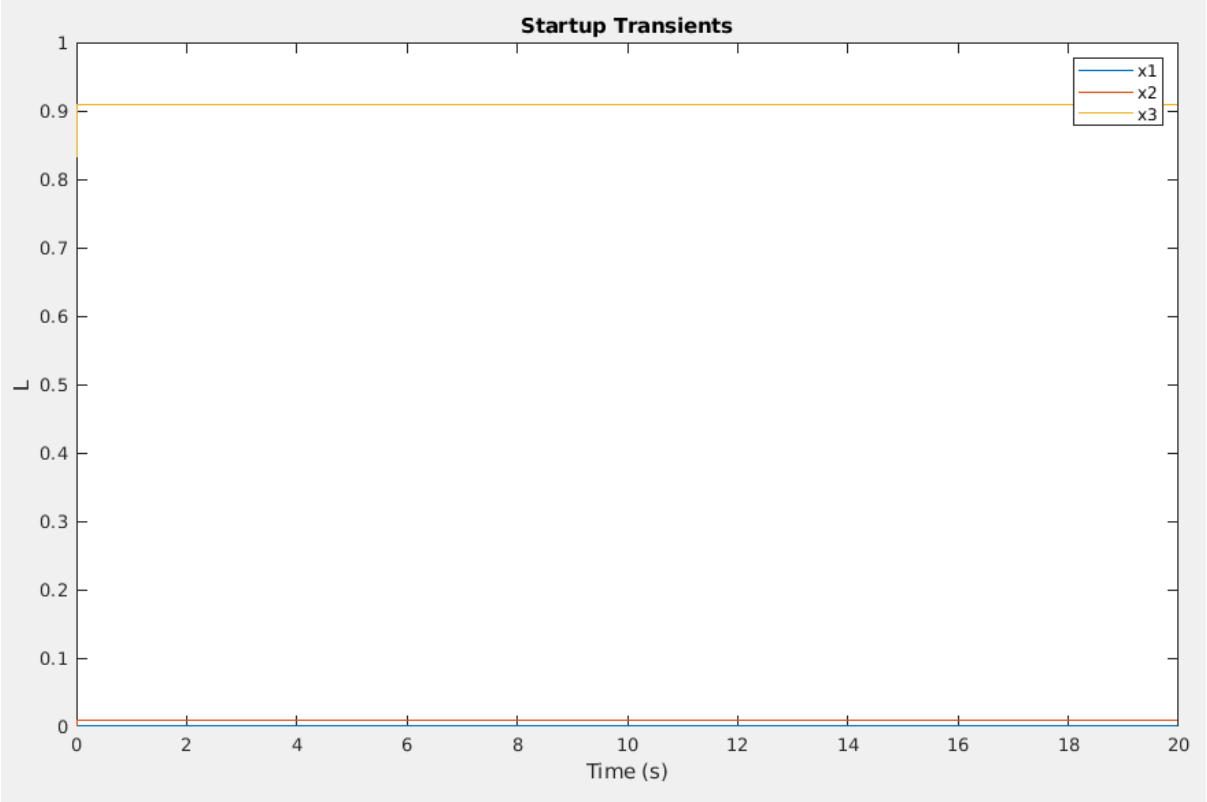


These two figures show how the filter is capable of getting occasionally very close to the measured value for a short period, before deviating again.

The filter with one sensor exhibits no consistent following or response and gets less accurate over time.

(b)

Demonstrate the start up transient of the Kalman filter by plotting L as a function of time.



From this plot we can see there are start up transients present for x2 and x3, however they only last 1 time step, so occur very quickly.

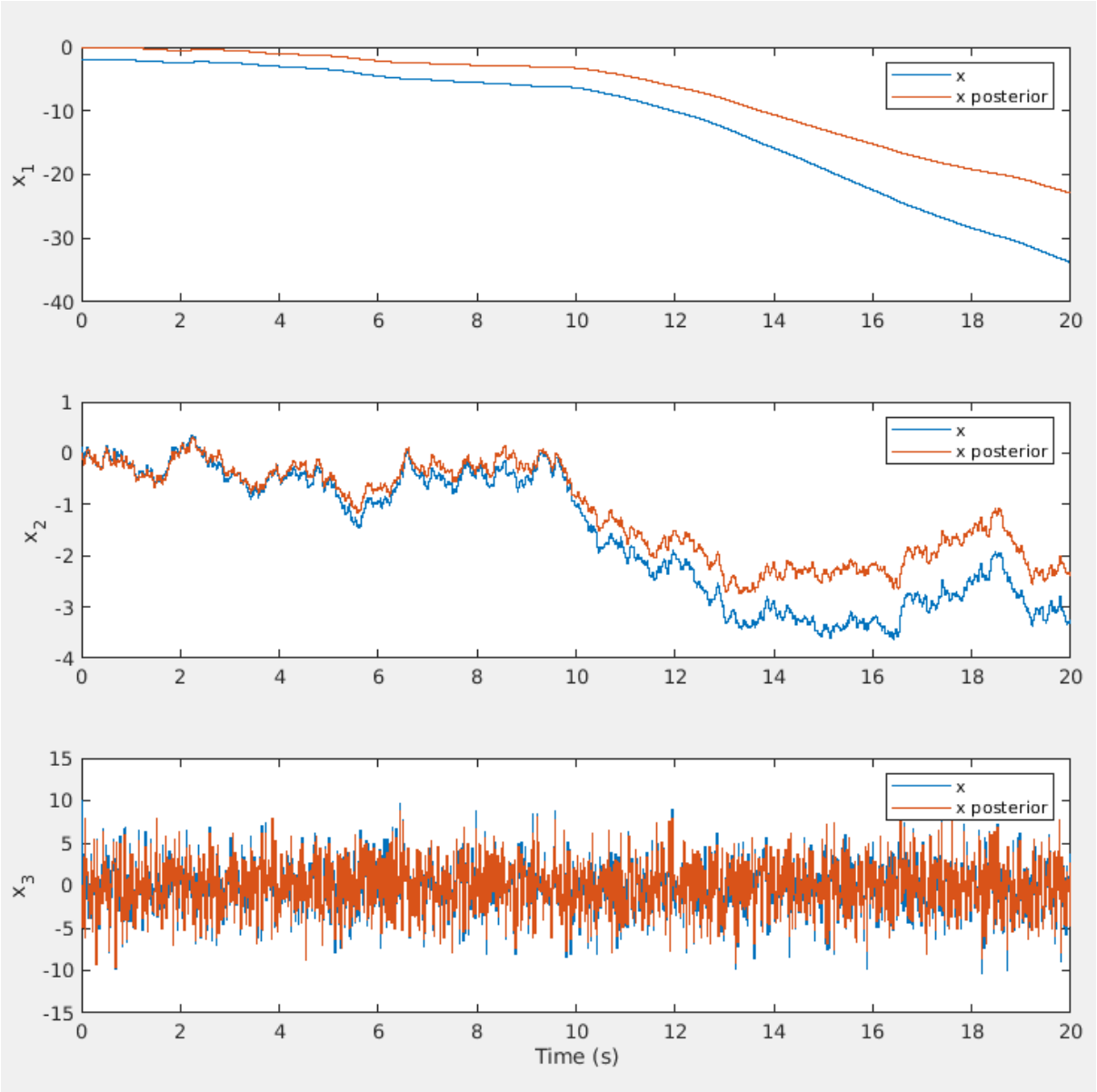
This was found to be the case for differing values of P_post estimate, the only difference being the magnitude of the initial value for x3. As was the case with differing values for the initial estimate of x0. So it was relatively unexciting.

(c)

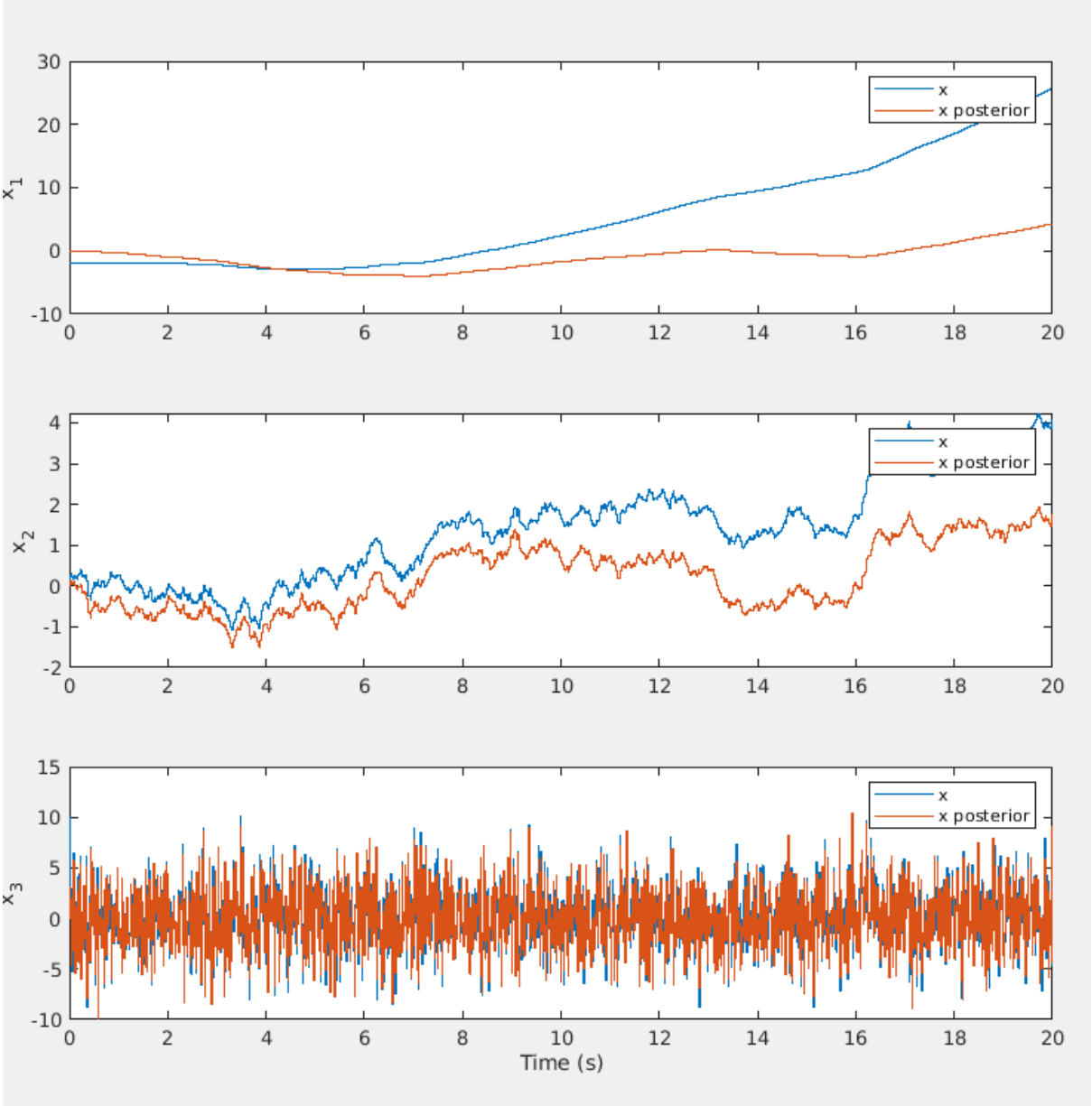
Contrast the performance of the filter with a steady state Kalman filter. You should use Matlab to design the steady state Kalman gain.

The steady state L used was as follows:
4.54545454545455e-05
0.00909090909090909
0.909090909090909

Which were the values settled on in the above graph
The same initial guesses were used.



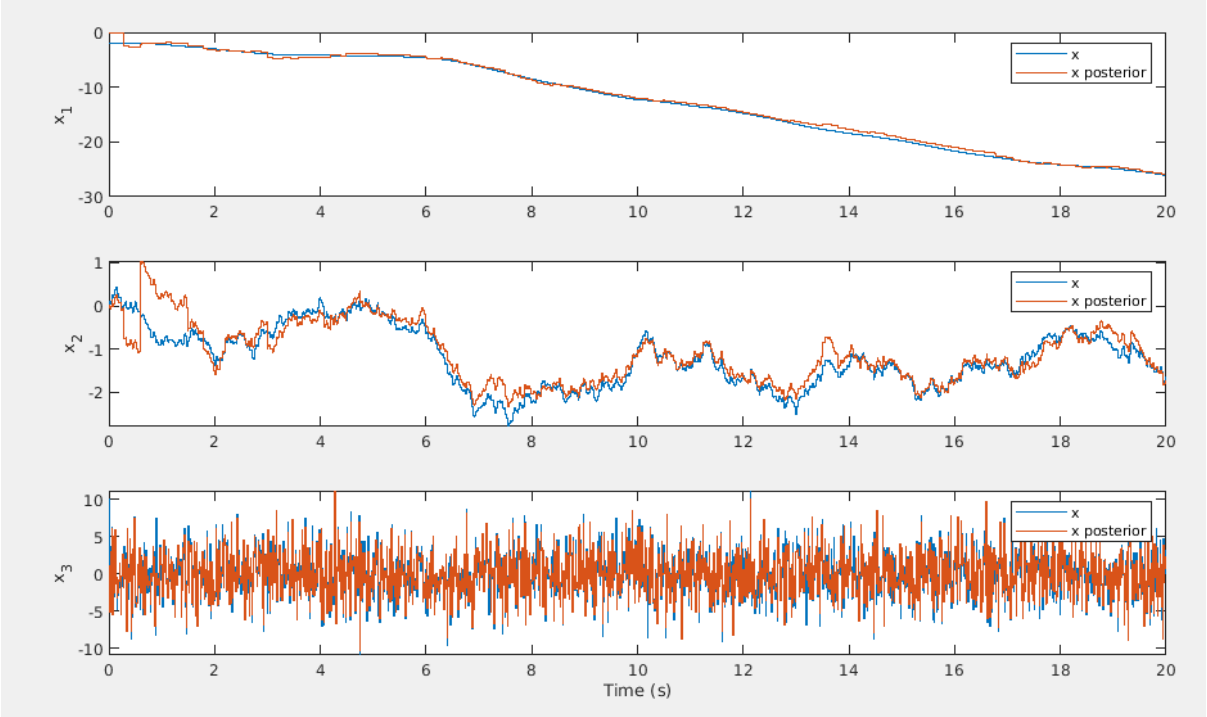
Given the value of L being settled on extremely quickly, there is expected ot be no major difference in the kalman filters with one sensor as the original filter is relatively steady state to begin with. And from the above plot this can be seen to be the case. It is a different plot from the first shown in (a) even though the position appears identical. The difference in velocity is more noticeable.



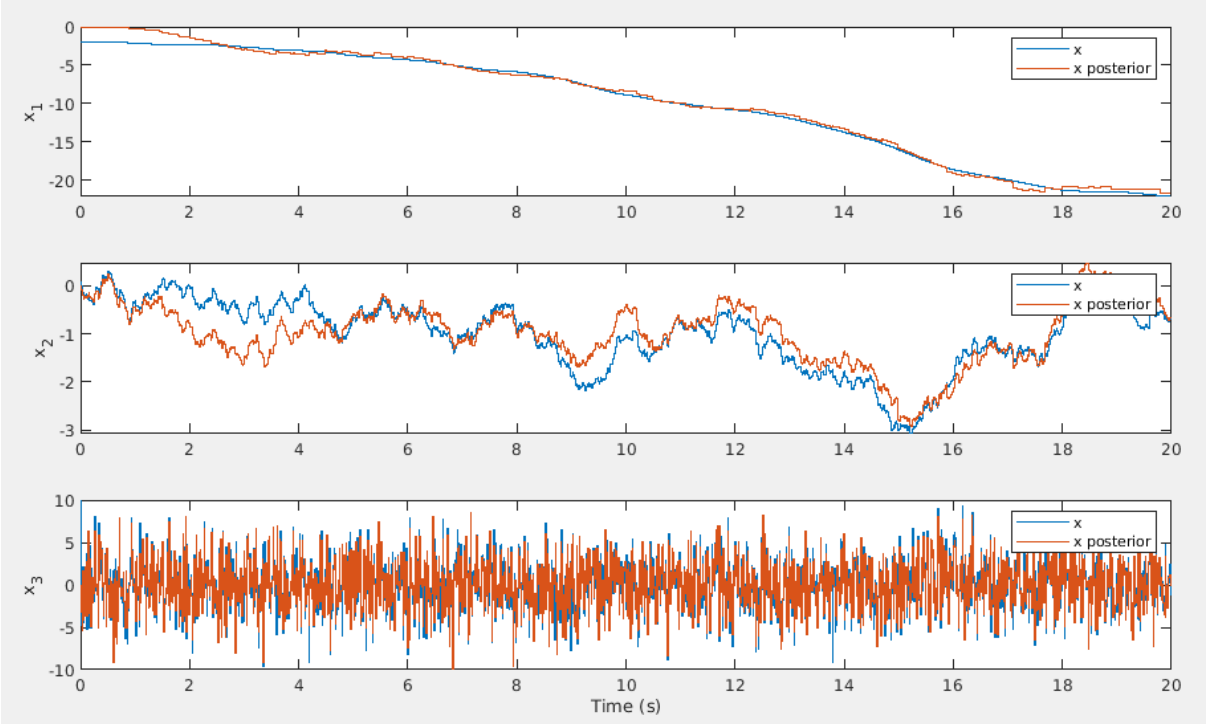
We can see similar behaviour to the non-steady state kalman filter also, when the estimate is close to the measured and the deviates away again as time progresses. So this system contains no major differences from linearisation.

(d)

```
x_post_sen(:,1) = [0,0,0]';     % Put your initial estimate here.
P_post_sen = diag([100,100,100]);     % Put your initial estimate here.
```



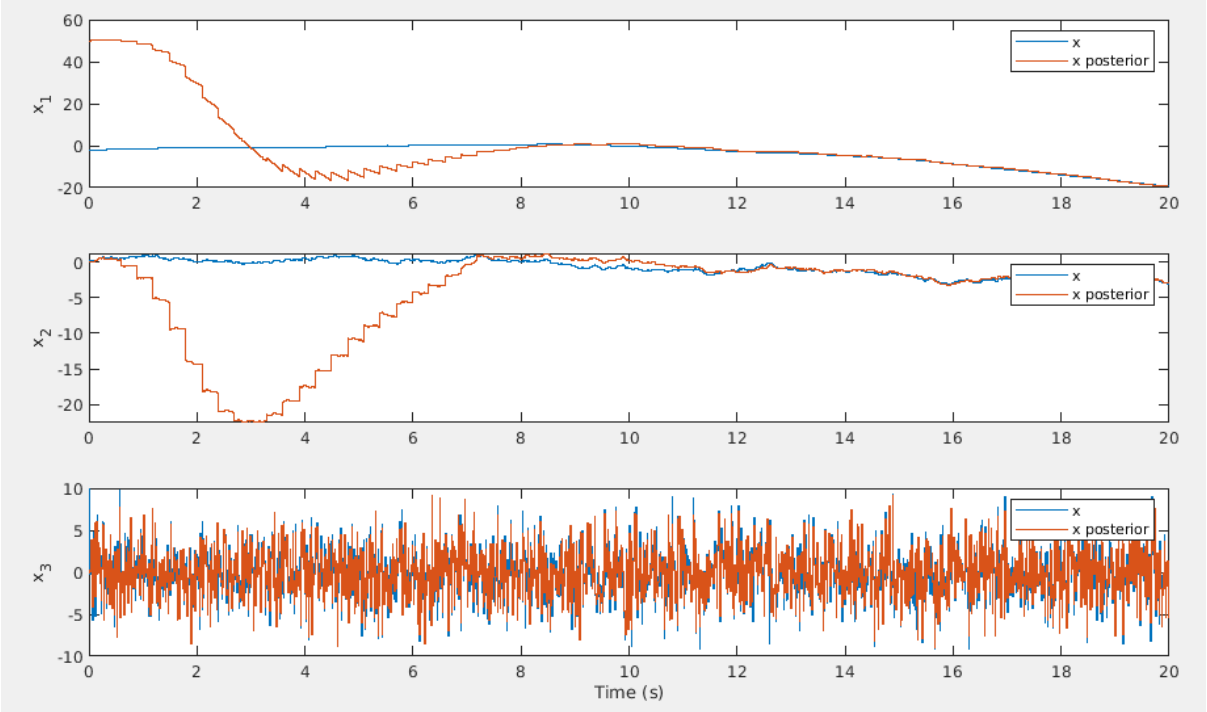

```
x_post_sen(:,1) = [0,0,0]';     % Put your initial estimate here.
P_post_sen = diag([0,0,0]);     % Put your initial estimate here.
```



The graphs on the previous page show that when using 2 sensors the kalman filter is capable of significantly better estimation, being much more accurate to the measured value. Additionally they show that by being super uncertain at the start of estimations results in faster settling time to the measured values as the filter believes the measured value more than the estimate.

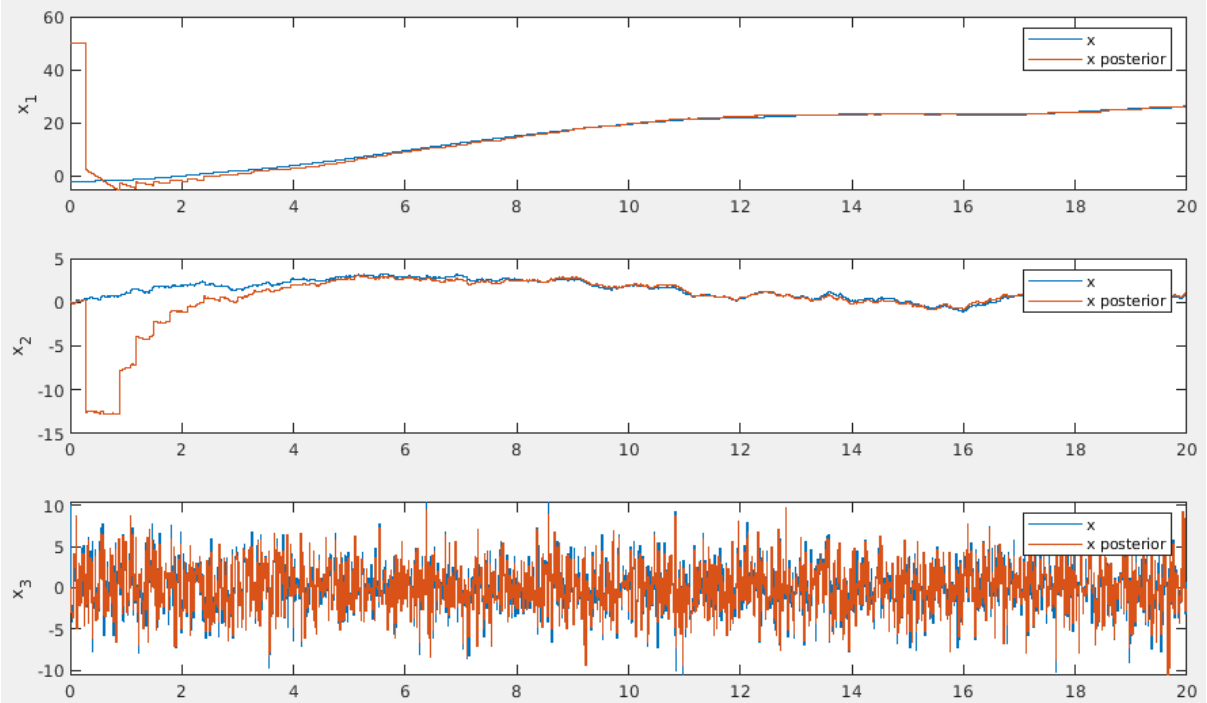
Not only does it have improved position estimation, but the velocity estimation has a noticeable improvement also after the initial transience.

```
x_post_sen(:,1) = [50,0,0]';     % Put your initial estimate here.
P_post_sen = diag([0,0,0]);      % Put your initial estimate here.
```

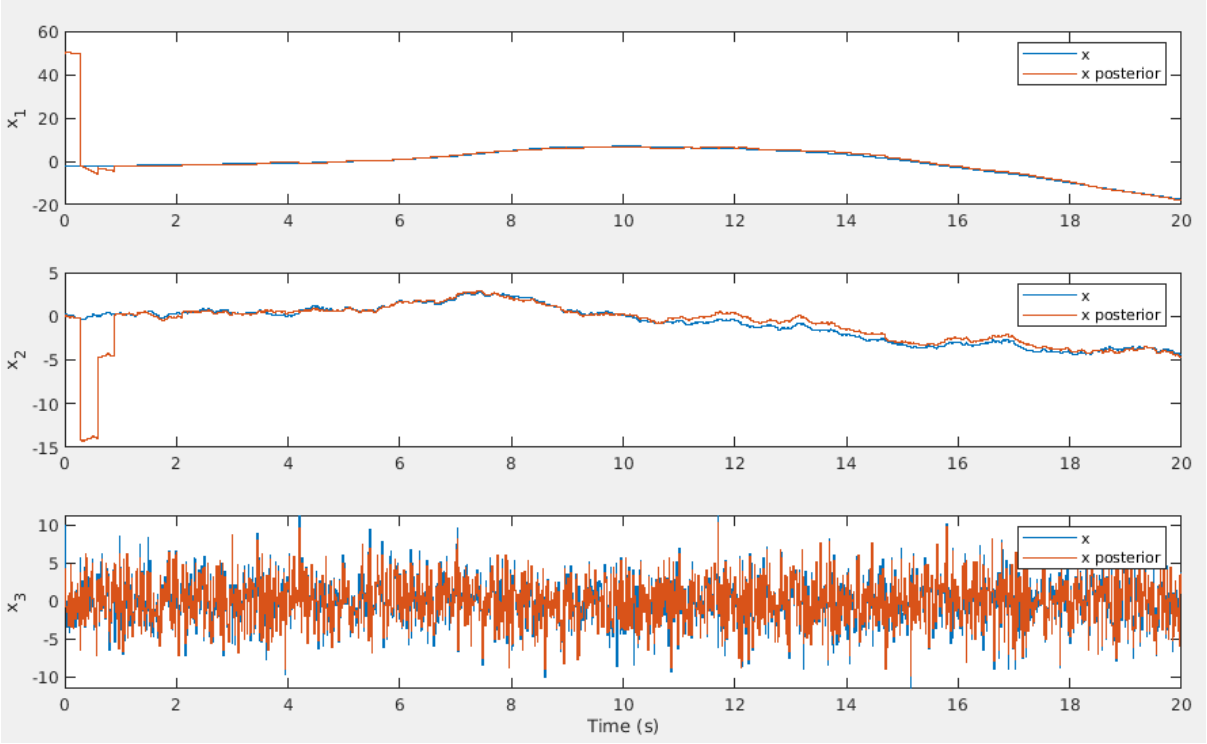


This plot emphasises the importance of a high initial variance as this run was extremely confident of the wrong value, which took 8 seconds to correct. By that point the robot may be lost if the sensors aren't good enough.

x_post_sen(:,1) = [50,0,0]'; % Put your initial estimate here.
P_post_sen = diag([5,5,5]); % Put your initial estimate here.



x_post_sen(:,1) = [50,0,0]'; % Put your initial estimate here.
P_post_sen = diag([100,100,100]); % Put your initial estimate here.



These graphs show how quickly it can respond when a high variance is initially set, as it quickly recovers from the very incorrect initial guess and the proceeds to follow the measured value incredibly well.

Appendix

Code used for this assignment

```
%==========================================================================
==========
% ECEN426 Advanced Mechatronic Systems
% Assignment One 2021
%
% This file is skeleton code that implements a Kalman filter to estimate the
% pose of a robot head/neck in the roll dimension.
%
% C.Hollitt
% 20th September 2021
%==========================================================================
==========

%--------------------------------------------------------------------------
% Form the model.

dt = 10e-3; % 10 ms time interval.
T_max = 20; % Duration of the simulation.
t=[0:1:T_max/dt]'.*dt;

% State vector
% [theta, omega, alpha]

A= [1 dt 0.5*dt^2;
   0 1 dt;
   0 0 0];
B= [0 0 1]';
C_1= [0 0 1];
C_2= [1 0 0];
D= 0;

% Noise model ------------------

G = [0.5*dt^2 ; dt ; 1];
var_acceleration = 10; % Variance of the acceleration noise on the head.

Q = G * var_acceleration * (G');

R_1 = 1;   % Variance in the acceleration measurement (sensor 1).
R_2 = 0.5; % Variance in the position measurement (sensor 2).

%--------------------------------------------------------------------------
```

```
% Initial  Conditions.

% Actual position that the neck begins the simulation.
x0 = [-2, 0, 10]'; % Don't change this - it is the true initial state that you
                % "don't know".

x(:,1) = x0; % Set the initial state for the simulation.

% Our initial guess at the neck position. a)
x_post(:,1) = [0,0,0]';     % Put your initial estimate here.
P_post = diag([5,5,5]);     % Put your initial estimate here.

% %-------------------------------------------------------------------------
L_acc = zeros(length(t),3);
L_acc(1,:) = [P_post * C_1' * inv( C_1 * P_post *C_1' + R_1)].';  % initialise the accumulated
L with initial guess
% % Run simulation b)

I=eye(3);

for k=2:length(t)
  % x(k) = Ax(k-1) + Bu(k) + w(k)
  w = G*sqrt(var_acceleration)*randn(1,1);

  x(:,k) = A*x(:,k-1) + w;   % u=0, so omit the B*u(k) term.

  x_prior(:,k) = A*x_post(:,k-1);
  P_prior = A*P_post*A' + Q;

  % Update the state using Sensor One at every step.
  % If you were to change S1_update_interval to 3 for example, then this block
  % of code would only run every third time through the outer loop.

  S1_update_interval = 1;
  if mod(k, S1_update_interval)==0
    v = sqrt(R_1)*randn(1,1); % Find noise to add to the measurement.
    y(:,k) = C_1*x(:,k) + v;         % Measure the state.

    L = P_prior * C_1' * inv( C_1 * P_prior *C_1' + R_1);
    L_acc(k,:) = L;
    x_post(:,k) = x_prior(:,k) + L*(y(:,k)-C_1*x_prior(:,k));
    P_post = (I-L*C_1)*P_prior;
    P_post = 0.5*(P_post + P_post'); % Make sure that the covariance remains symmetric.
  end

end

% Plot results ----------------------------------------------------------------
```

```
figure(1)
for plot_num = 1:3
  subplot(3,1,plot_num)
  stairs(t,[x(plot_num,:);x_post(plot_num,:)]')
  ylabel(['x_' num2str(plot_num)])
  legend("x","x posterior")
end
xlabel('Time (s)')

% B
figure(2)
plot(t,L_acc)
ylabel("L")
xlabel("Time (s)")
legend("x1","x2","x3")
title("Startup Transients")

%C Steady State Kalman Filter

%L is constant and set to the last value of L in the non-steady state run
%through

L_ss = L;
x_ss(:,1) = x0; % Set the initial state for the simulation.
x_post_ss(:,1) = [0,0,0]';     % Put your initial estimate here.

for k=2:length(t)
  % x(k) = Ax(k-1) + Bu(k) + w(k)
  w = G*sqrt(var_acceleration)*randn(1,1);

  x_ss(:,k) = A*x_ss(:,k-1) + w;   % u=0, so omit the B*u(k) term.

  x_prior_ss(:,k) = A*x_post_ss(:,k-1);

  % Update the state using Sensor One at every step.
  % If you were to change S1_update_interval to 3 for example, then this block
  % of code would only run every third time through the outer loop.

  S1_update_interval = 1;
  if mod(k, S1_update_interval)==0
    v = sqrt(R_1)*randn(1,1); % Find noise to add to the measurement.
    y(:,k) = C_1*x_ss(:,k) + v;         % Measure the state.

    x_post_ss(:,k) = x_prior_ss(:,k) + L_ss*(y(:,k)-C_1*x_prior_ss(:,k));
  end

end
```

```
% Plot results ----------------------------------------------------------------
figure(3)
for plot_num = 1:3
  subplot(3,1,plot_num)
  stairs(t,[x_ss(plot_num,:);x_post_ss(plot_num,:)]')
  ylabel(['x_' num2str(plot_num)])
  legend("x","x posterior")
end
xlabel('Time (s)')
%Is the same because the non-steady-state filter quickly settles on L

% D Add a second sensor

x_sen(:,1) = x0; % Set the initial state for the simulation.
% Our initial guess at the neck position. a)
x_post_sen(:,1) = [50,0,0]';     % Put your initial estimate here.
P_post_sen = diag([0,0,0]);     % Put your initial estimate here.

for k=2:length(t);
  % x(k) = Ax(k-1) + Bu(k) + w(k)
  w = G*sqrt(var_acceleration)*randn(1,1);

  x_sen(:,k) = A*x_sen(:,k-1) + w;   % u=0, so omit the B*u(k) term.

  x_prior_sen(:,k) = A*x_post_sen(:,k-1);
  P_prior_sen = A*P_post_sen*A' + Q;

  % Update the state using Sensor One at every step.
  % If you were to change S1_update_interval to 3 for example, then this block
  % of code would only run every third time through the outer loop.

  S1_update_interval = 1;
  S2_update_interval = 8;
  if mod(k, S1_update_interval)==0
    v = sqrt(R_1)*randn(1,1); % Find noise to add to the measurement.
    if mod(k, S2_update_interval) == 0
       v2 = sqrt(R_2)*randn(1,1);

       y(:,k) = C_2*x_sen(:,k) + v2;         % Measure the state.
       L_sen = P_prior_sen * C_2' * inv( C_2 * P_prior_sen *C_2' + R_2);
       L_acc_sen(k,:) = L_sen;
       x_post_sen(:,k) = x_prior_sen(:,k) + L_sen*(y(:,k)-C_2*x_prior_sen(:,k));
       P_post_sen = (I-L_sen*C_2)*P_prior_sen;
       P_post_sen = 0.5*(P_post_sen + P_post_sen'); % Make sure that the covariance
remains symmetric.

    else
       y(:,k) = C_1*x_sen(:,k) + v;         % Measure the state.
```

```
        L_sen = P_prior_sen * C_1' * inv( C_1 * P_prior_sen *C_1' + R_1);
        L_acc_sen(k,:) = L_sen;
        x_post_sen(:,k) = x_prior_sen(:,k) + L_sen*(y(:,k)-C_1*x_prior_sen(:,k));
        P_post_sen = (I-L_sen*C_1)*P_prior_sen;
        P_post_sen = 0.5*(P_post_sen + P_post_sen'); % Make sure that the covariance
remains symmetric.
    end
  end



end
figure(4)
for plot_num = 1:3
  subplot(3,1,plot_num)
  stairs(t,[x_sen(plot_num,:);x_post_sen(plot_num,:)]')
  ylabel(['x_' num2str(plot_num)])
  legend("x","x posterior")
end
xlabel('Time (s)')

% END
================================================================================
====
```