Moving Horizon Estimation

Richard M. Murray, 24 Feb 2023

In this notebook we illustrate the implementation of moving horizon estimation (MHE)

In [1]:
```
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
import control as ct

import control.optimal as opt
import control.flatsys as fs
```

In [2]:
```
# Import the new MHE routines (to be moved to python-control)
import ctrlutil as opt_
```

System Description

The dynamics of the system with disturbances on the x and y variables is given by

$$
\begin{aligned}
m\ddot{x} &= F_1 \cos\theta - F_2 \sin\theta - c\dot{x} + d_x, \\
m\ddot{y} &= F_1 \sin\theta + F_2 \cos\theta - c\dot{y} - mg + d_y, \\
J\ddot{\theta} &= rF_1.
\end{aligned}
$$

The measured values of the system are the position and orientation, with added noise n_x, n_y, and n_θ:

$$
\vec{y} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}.
$$

In [3]:
```
# pvtol = nominal system (no disturbances or noise)
# noisy_pvtol = pvtol w/ process disturbances and sensor noise
from pvtol import pvtol, pvtol_noisy, plot_results
import pvtol as pvt

# Find the equiblirum point corresponding to the origin
xe, ue = ct.find_eqpt(
    pvtol, np.zeros(pvtol.nstates),
    np.zeros(pvtol.ninputs), [0, 0, 0, 0, 0, 0],
    iu=range(2, pvtol.ninputs), iy=[0, 1])

# Initial condition = 2 meters right, 1 meter up
x0, u0 = ct.find_eqpt(
    pvtol, np.zeros(pvtol.nstates),
    np.zeros(pvtol.ninputs), np.array([2, 1, 0, 0, 0, 0]),
```

```
    iu=range(2, pvtol.ninputs), iy=[0, 1])

# Extract the linearization for use in LQR design
pvtol_lin = pvtol.linearize(xe, ue)
A, B = pvtol_lin.A, pvtol_lin.B

print(pvtol, "\n")
print(pvtol_noisy)
```

```
<FlatSystem>: pvtol
Inputs (2): ['F1', 'F2']
Outputs (6): ['x0', 'x1', 'x2', 'x3', 'x4', 'x5']
States (6): ['x0', 'x1', 'x2', 'x3', 'x4', 'x5']

Update: <function _pvtol_update at 0x7f8710d60b80>
Output: <function _pvtol_output at 0x7f8710d60280>

Forward: <function _pvtol_flat_forward at 0x7f8710d3f010>
Reverse: <function _pvtol_flat_reverse at 0x7f87110bfb50>

<NonlinearIOSystem>: pvtol_noisy
Inputs (7): ['F1', 'F2', 'Dx', 'Dy', 'Nx', 'Ny', 'Nth']
Outputs (6): ['x0', 'x1', 'x2', 'x3', 'x4', 'x5']
States (6): ['x0', 'x1', 'x2', 'x3', 'x4', 'x5']

Update: <function _noisy_update at 0x7f87110bfbe0>
Output: <function _noisy_output at 0x7f87110bfc70>
```

Control Design

In [4]:

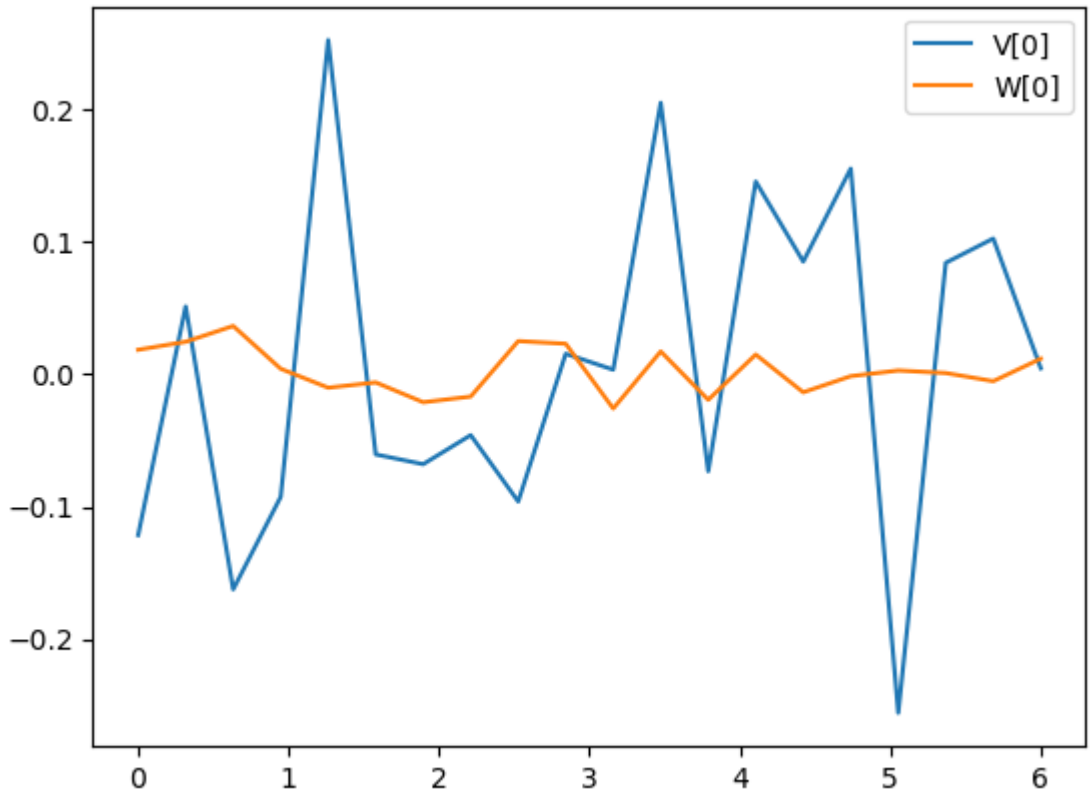
```
#
# LQR design w/ physically motivated weighting
#
# Shoot for 10 cm error in x, 10 cm error in y.  Try to keep the angle
# less than 5 degrees in making the adjustments.  Penalize side forces
# due to loss in efficiency.
#

Qx = np.diag([100, 10, (180/np.pi) / 5, 0, 0, 0])
Qu = np.diag([10, 1])
K, _, _ = ct.lqr(A, B, Qx, Qu)

# Compute the full state feedback solution
lqr_ctrl, _ = ct.create_statefbk_iosystem(pvtol, K)

# Define the closed loop system that will be used to generate trajectories
lqr_clsys = ct.interconnect(
    [pvtol_noisy, lqr_ctrl],
    inplist = lqr_ctrl.input_labels[0:pvtol.ninputs + pvtol.nstates] + \
        pvtol_noisy.input_labels[pvtol.ninputs:],
    inputs = lqr_ctrl.input_labels[0:pvtol.ninputs + pvtol.nstates] + \
        pvtol_noisy.input_labels[pvtol.ninputs:],
    outlist = pvtol.output_labels + lqr_ctrl.output_labels,
    outputs = pvtol.output_labels + lqr_ctrl.output_labels
```

```
)
print(lqr_clsys)
```

```
<InterconnectedSystem>: F2
Inputs (13): ['xd[0]', 'xd[1]', 'xd[2]', 'xd[3]', 'xd[4]', 'xd[5]', 'ud
[0]', 'ud[1]', 'Dx', 'Dy', 'Nx', 'Ny', 'Nth']
Outputs (8): ['x0', 'x1', 'x2', 'x3', 'x4', 'x5', 'F1', 'F2']
States (6): ['pvtol_noisy_x0', 'pvtol_noisy_x1', 'pvtol_noisy_x2', 'pvtol_n
oisy_x3', 'pvtol_noisy_x4', 'pvtol_noisy_x5']
```

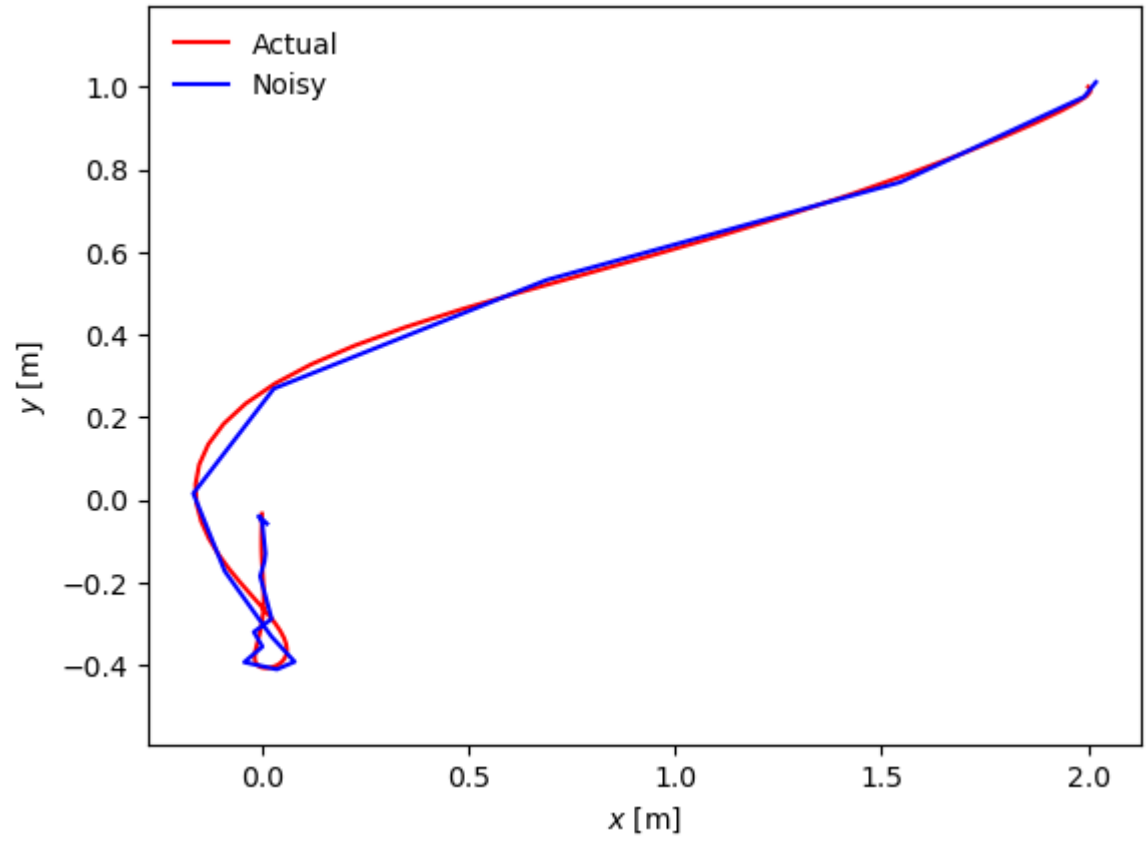
In [5]:
```
# Disturbance and noise intensities
Qv = np.diag([1e-2, 1e-2])
Qw = np.array([[1e-4, 0, 1e-5], [0, 1e-4, 1e-5], [1e-5, 1e-5, 1e-4]])

# Initial state covariance
P0 = np.eye(pvtol.nstates)
```

In [6]:
```
# Create the time vector for the simulation
Tf = 6
timepts = np.linspace(0, Tf, 20)

# Create representative process disturbance and sensor noise vectors
# np.random.seed(117)           # avoid figures changing from run to run
V = ct.white_noise(timepts, Qv)
# V = np.clip(V0, -0.1, 0.1)    # Hold for later
W = ct.white_noise(timepts, Qw)
# plt.plot(timepts, V0[0], 'b--', label="V[0]")
plt.plot(timepts, V[0], label="V[0]")
plt.plot(timepts, W[0], label="W[0]")
plt.legend();
```

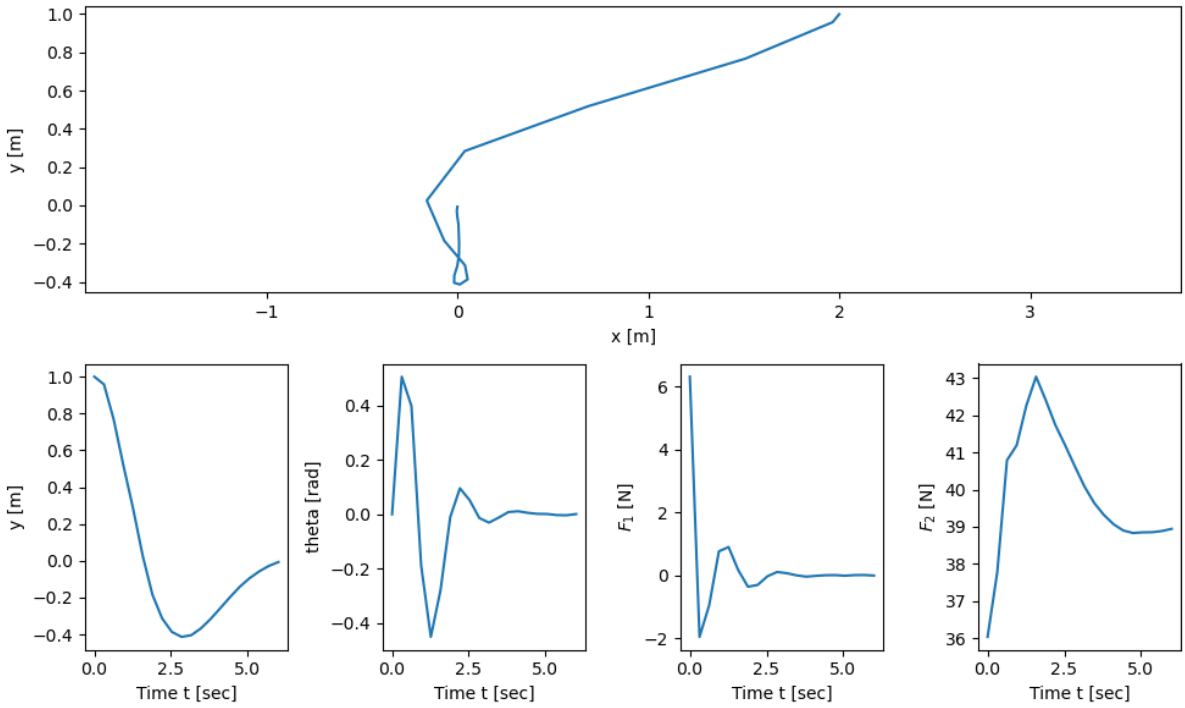


In [7]:

```
# Desired trajectory
xd, ud = xe, ue
# xd = np.vstack([
#     np.sin(2 * np.pi * timepts / timepts[-1]),
#     np.zeros((5, timepts.size))])
# ud = np.outer(ue, np.ones_like(timepts))

# Run a simulation with full state feedback (no noise) to generate a traject
uvec = [xd, ud, V, W*0]
lqr_resp = ct.input_output_response(lqr_clsys, timepts, uvec, x0)
U = lqr_resp.outputs[6:8]                    # controller input signals
Y = lqr_resp.outputs[0:3] + W                # noisy output signals (noise i

# Compare to the no noise case
uvec = [xd, ud, V*0, W*0]
lqr0_resp = ct.input_output_response(lqr_clsys, timepts, uvec, x0)
lqr0_fine = ct.input_output_response(lqr_clsys, timepts, uvec, x0,
                                     t_eval=np.linspace(timepts[0], timepts[
U0 = lqr0_resp.outputs[6:8]
Y0 = lqr0_resp.outputs[0:3]

# Compare the results
# plt.plot(Y0[0], Y0[1], 'k--', linewidth=2, label="No disturbances")
plt.plot(lqr0_fine.states[0], lqr0_fine.states[1], 'r-', label="Actual")
plt.plot(Y[0], Y[1], 'b-', label="Noisy")

plt.xlabel('$x$ [m]')
plt.ylabel('$y$ [m]')
plt.axis('equal')
plt.legend(frameon=False)

plt.figure()
plot_results(timepts, lqr_resp.states, lqr_resp.outputs[6:8]);
```




```
<Figure size 640x480 with 0 Axes>
```



In [8]:
```
# Utility functions for making plots
def plot_state_comparison(
    timepts, est_states, act_states=None, estimated_label='$\\hat x_{i}$', a
    start=0):
    for i in range(sys.nstates):
        plt.subplot(2, 3, i+1)
        if act_states is not None:
```

```
            plt.plot(timepts[start:], act_states[i, start:], 'r--',
                     label=actual_label.format(i=i))
        plt.plot(timepts[start:], est_states[i, start:], 'b',
                 label=estimated_label.format(i=i))
        plt.legend()
    plt.tight_layout()

# Define a function to plot out all of the relevant signals
def plot_estimator_response(timepts, estimated, U, V, Y, W, start=0):
    # Plot the input signal and disturbance
    for i in [0, 1]:
        # Input signal (the same across all)
        plt.subplot(4, 3, i+1)
        plt.plot(timepts[start:], U[i, start:], 'k')
        plt.ylabel(f'U[{i}]')

        # Plot the estimated disturbance signal
        plt.subplot(4, 3, 4+i)
        plt.plot(timepts[start:], estimated.inputs[i, start:], 'b-', label="
        plt.plot(timepts[start:], V[i, start:], 'k', label="actual")
        plt.ylabel(f'V[{i}]')

    plt.subplot(4, 3, 6)
    plt.plot(0, 0, 'b', label="estimated")
    plt.plot(0, 0, 'k', label="actual")
    plt.plot(0, 0, 'r', label="measured")
    plt.legend(frameon=False)
    plt.grid(False)
    plt.axis('off')

    # Plot the output (measured and estimated)
    for i in [0, 1, 2]:
        plt.subplot(4, 3, 7+i)
        plt.plot(timepts[start:], Y[i, start:], 'r', label="measured")
        plt.plot(timepts[start:], estimated.states[i, start:], 'b', label="m
        plt.plot(timepts[start:], Y[i, start:] - W[i, start:], 'k', label="a
        plt.ylabel(f'Y[{i}]')

    for i in [0, 1, 2]:
        plt.subplot(4, 3, 10+i)
        plt.plot(timepts[start:], estimated.outputs[i, start:], 'b', label="
        plt.plot(timepts[start:], W[i, start:], 'k', label="actual")
        plt.ylabel(f'W[{i}]')
        plt.xlabel('Time [s]')

    plt.tight_layout()
```

State Estimation

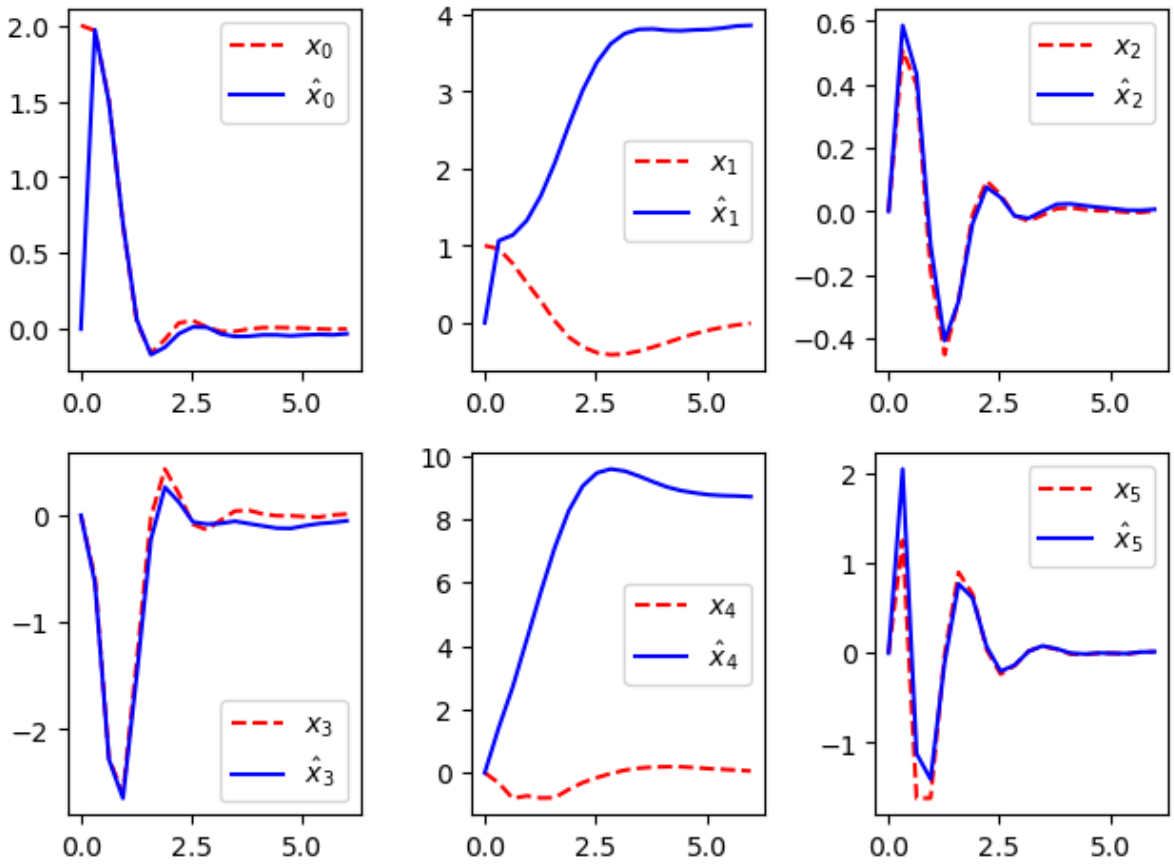
In [9]:
```
# Create a new system with only x, y, theta as outputs
# TODO: add this to pvtol.py?
sys = ct.NonlinearIOSystem(
    pvt._noisy_update, lambda t, x, u, params: x[0:3], name="pvtol_noisy",
    states = [f'x{i}' for i in range(6)],
```

```
    inputs = ['F1', 'F2'] + ['Dx', 'Dy'],
    outputs = ['x', 'y', 'theta']
)
```

In [10]:

```
# Standard Kalman filter
linsys = sys.linearize(xe, [ue, V[:, 0] * 0])
# print(linsys)
B = linsys.B[:, 0:2]
G = linsys.B[:, 2:4]
linsys = ct.ss(
    linsys.A, B, linsys.C, 0,
    states=sys.state_labels, inputs=sys.input_labels[0:2], outputs=sys.outpu
# print(linsys)

estim = ct.create_estimator_iosystem(linsys, Qv, Qw, G=G, P0=P0)
print(estim)
print(f'{xe=}, {P0=}')

kf_resp = ct.input_output_response(
    estim, timepts, [Y, U], X0 = [xe, P0.reshape(-1)])
plot_state_comparison(timepts, kf_resp.outputs, lqr_resp.states)
```

```
<NonlinearIOSystem>: sys[6]
Inputs (5): ['x', 'y', 'theta', 'F1', 'F2']
Outputs (6): ['xhat[0]', 'xhat[1]', 'xhat[2]', 'xhat[3]', 'xhat[4]', 'xhat
[5]']
States (42): ['xhat[0]', 'xhat[1]', 'xhat[2]', 'xhat[3]', 'xhat[4]', 'xhat
[5]', 'P[0,0]', 'P[0,1]', 'P[0,2]', 'P[0,3]', 'P[0,4]', 'P[0,5]', 'P[1,0]',
'P[1,1]', 'P[1,2]', 'P[1,3]', 'P[1,4]', 'P[1,5]', 'P[2,0]', 'P[2,1]', 'P[2,
2]', 'P[2,3]', 'P[2,4]', 'P[2,5]', 'P[3,0]', 'P[3,1]', 'P[3,2]', 'P[3,3]',
'P[3,4]', 'P[3,5]', 'P[4,0]', 'P[4,1]', 'P[4,2]', 'P[4,3]', 'P[4,4]', 'P[4,
5]', 'P[5,0]', 'P[5,1]', 'P[5,2]', 'P[5,3]', 'P[5,4]', 'P[5,5]']

Update: <function create_estimator_iosystem.<locals>._estim_update at 0x7f8
710d60a60>
Output: <function create_estimator_iosystem.<locals>._estim_output at 0x7f8
7110bfa30>
xe=array([ 0.000000e+00,  0.000000e+00,  0.000000e+00,  0.000000e+00,
       -1.766654e-27,  0.000000e+00]), P0=array([[1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 1.]])
```



Extended Kalman filter

In [11]:

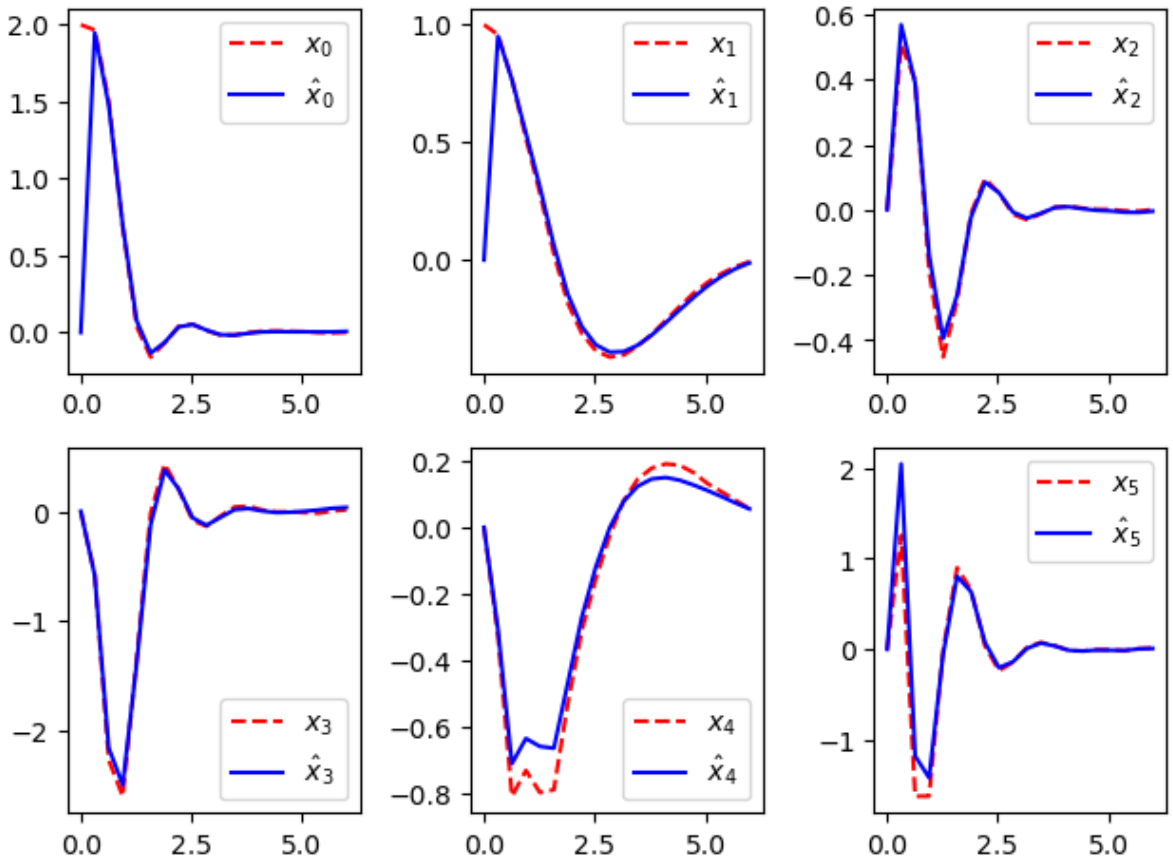
```
# Define the disturbance input and measured output matrices
F = np.array([[0, 0], [0, 0], [0, 0], [1/pvtol.params['m'], 0], [0, 1/pvtol.
C = np.eye(3, 6)

Qwinv = np.linalg.inv(Qw)

# Estimator update law
def estimator_update(t, x, u, params):
    # Extract the states of the estimator
    xhat = x[0:pvtol.nstates]
    P = x[pvtol.nstates:].reshape(pvtol.nstates, pvtol.nstates)

    # Extract the inputs to the estimator
    y = u[0:3]                  # just grab the first three outputs
    u = u[6:8]                  # get the inputs that were applied as well

    # Compute the linearization at the current state
    A = pvtol.A(xhat, u)        # A matrix depends on current state
    # A = pvtol.A(xe, ue)       # Fixed A matrix (for testing/comparison)

    # Compute the optimal "gain
    L = P @ C.T @ Qwinv

    # Update the state estimate
    xhatdot = pvtol.updfcn(t, xhat, u, params) - L @ (C @ xhat - y)
```

```
    # Update the covariance
    Pdot = A @ P + P @ A.T - P @ C.T @ Qwinv @ C @ P + F @ Qv @ F.T

    # Return the derivative
    return np.hstack([xhatdot, Pdot.reshape(-1)])

def estimator_output(t, x, u, params):
    # Return the estimator states
    return x[0:pvtol.nstates]

ekf = ct.NonlinearIOSystem(
    estimator_update, estimator_output,
    states=pvtol.nstates + pvtol.nstates**2,
    inputs= pvtol_noisy.output_labels \
        + pvtol_noisy.input_labels[0:pvtol.ninputs],
    outputs=[f'xh{i}' for i in range(pvtol.nstates)]
)
print(ekf)
```

```
<NonlinearIOSystem>: sys[7]
Inputs (8): ['x0', 'x1', 'x2', 'x3', 'x4', 'x5', 'F1', 'F2']
Outputs (6): ['xh0', 'xh1', 'xh2', 'xh3', 'xh4', 'xh5']
States (42): ['x[0]', 'x[1]', 'x[2]', 'x[3]', 'x[4]', 'x[5]', 'x[6]', 'x
[7]', 'x[8]', 'x[9]', 'x[10]', 'x[11]', 'x[12]', 'x[13]', 'x[14]', 'x[15]',
'x[16]', 'x[17]', 'x[18]', 'x[19]', 'x[20]', 'x[21]', 'x[22]', 'x[23]', 'x
[24]', 'x[25]', 'x[26]', 'x[27]', 'x[28]', 'x[29]', 'x[30]', 'x[31]', 'x[3
2]', 'x[33]', 'x[34]', 'x[35]', 'x[36]', 'x[37]', 'x[38]', 'x[39]', 'x[4
0]', 'x[41]']

Update: <function estimator_update at 0x7f87112d1630>
Output: <function estimator_output at 0x7f87112d0ee0>
```

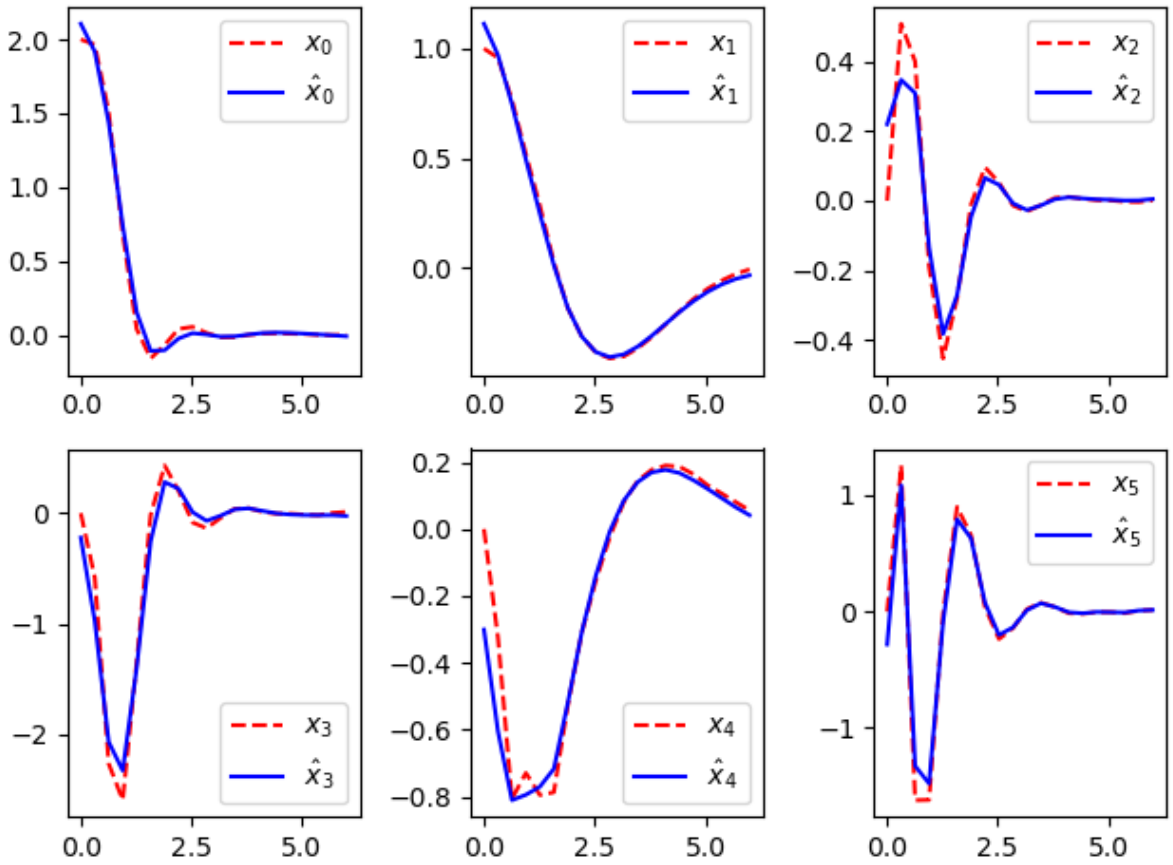
In [12]:

```
ekf_resp = ct.input_output_response(
    ekf, timepts, [lqr_resp.states, lqr_resp.outputs[6:8]],
    X0=[xe, P0.reshape(-1)])
plot_state_comparison(timepts, ekf_resp.outputs, lqr_resp.states)
```

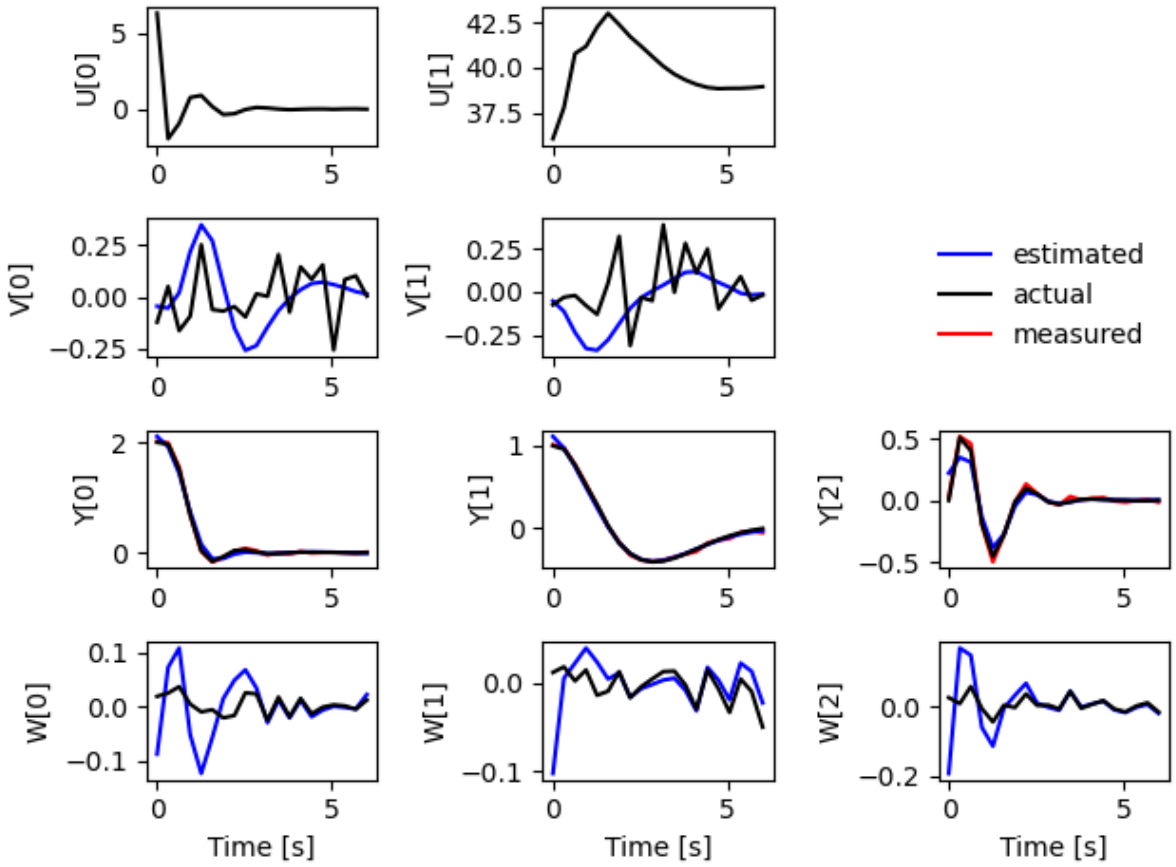


In [13]:
```
# Define the optimal estimation problem
traj_cost = opt_.gaussian_likelihood_cost(sys, Qv, Qw)
init_cost = lambda xhat, x: (xhat - x) @ P0 @ (xhat - x)
oep = opt_.OptimalEstimationProblem(
        sys, timepts, traj_cost, terminal_cost=init_cost)

# Compute the estimate from the noisy signals
est = oep.compute_estimate(Y, U, X0=lqr_resp.states[:, 0])
plot_state_comparison(timepts, est.states, lqr_resp.states)
```

```
Summary statistics:
* Cost function calls: 5374
* Final cost: 493.4132750101824
```

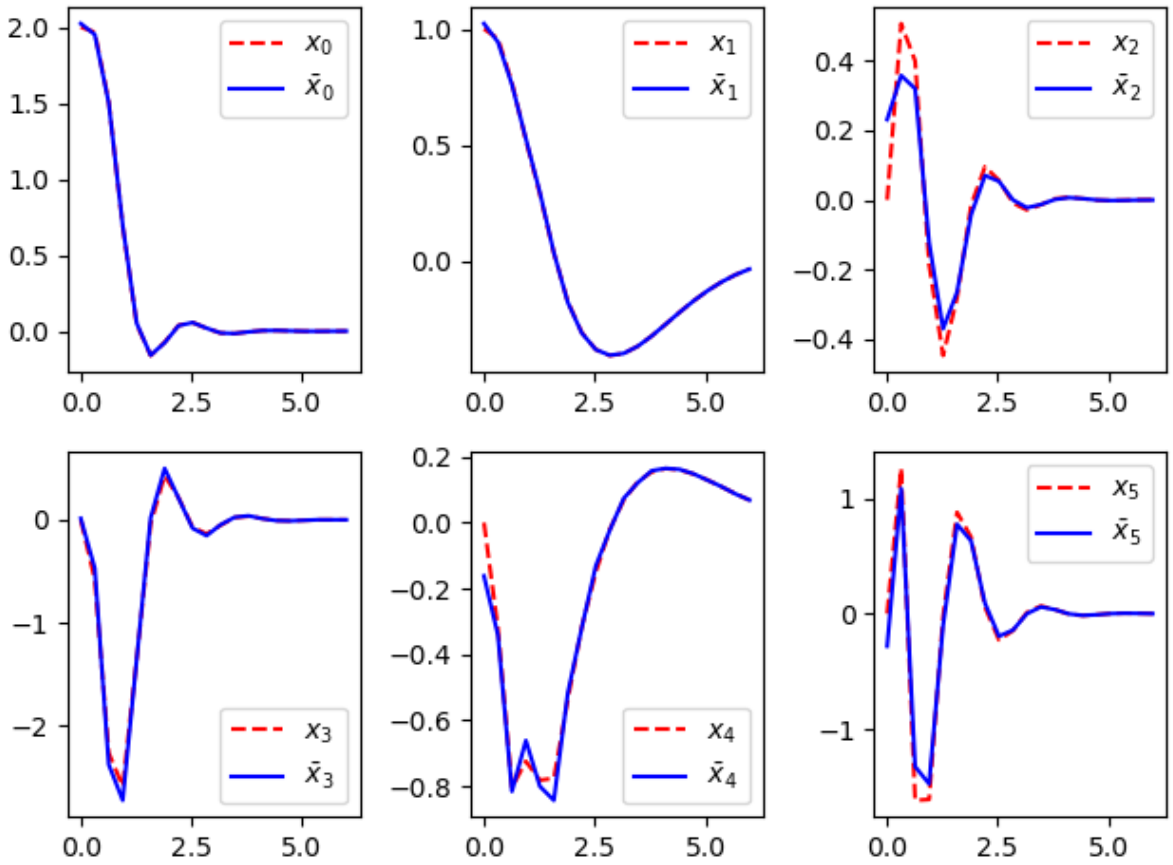


In [14]:
```
# Plot the response of the estimator
plot_estimator_response(timepts, est, U, V, Y, W)
```

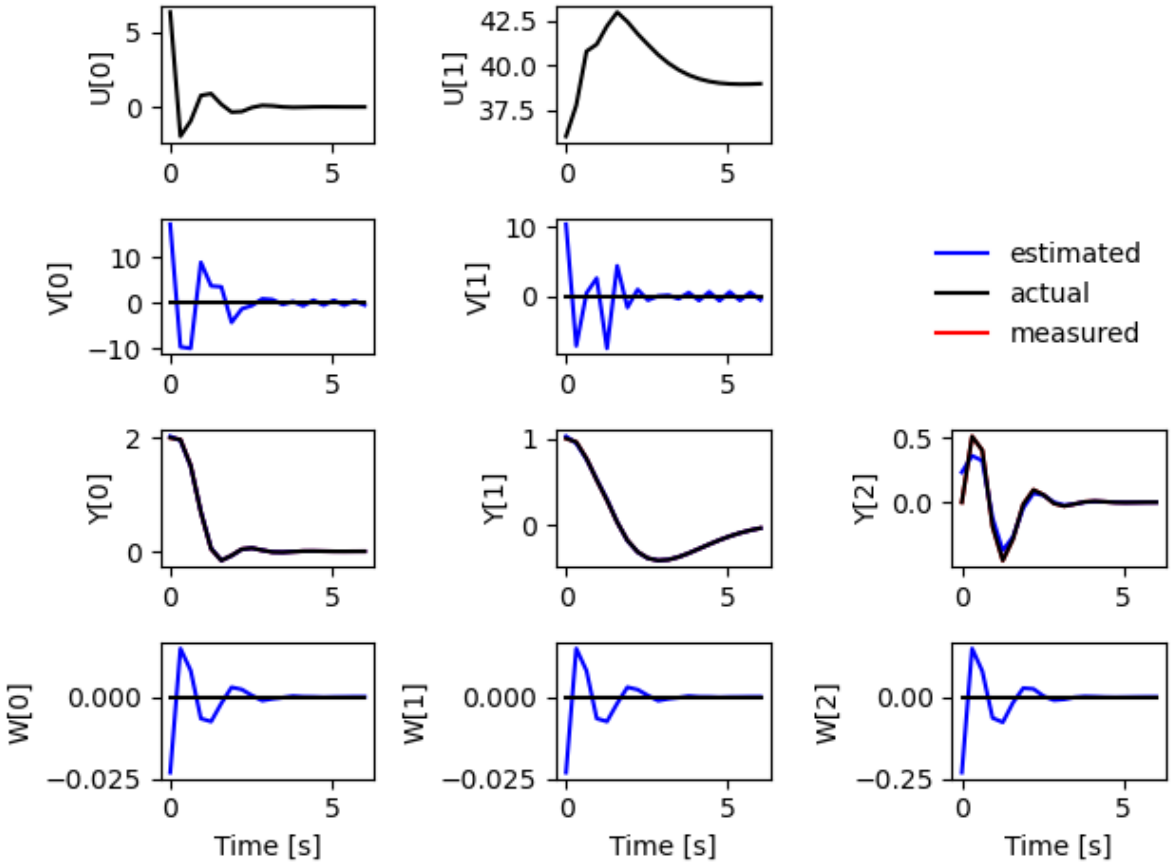


In [15]:
```
# Noise free and disturbance free => estimation should be near perfect
noisefree_cost = opt_.gaussian_likelihood_cost(sys, Qv, Qw*1e-6)
oep0 = opt_.OptimalEstimationProblem(
        sys, timepts, noisefree_cost, terminal_cost=init_cost)
est0 = oep0.compute_estimate(Y0, U0, X0=lqr0_resp.states[:, 0],
                             initial_guess=(lqr0_resp.states, V * 0))
plot_state_comparison(
    timepts, est0.states, lqr0_resp.states, estimated_label='$\\bar x_{i}$')
```

```
Summary statistics:
* Cost function calls: 9464
* Final cost: 212754409.97292745
```



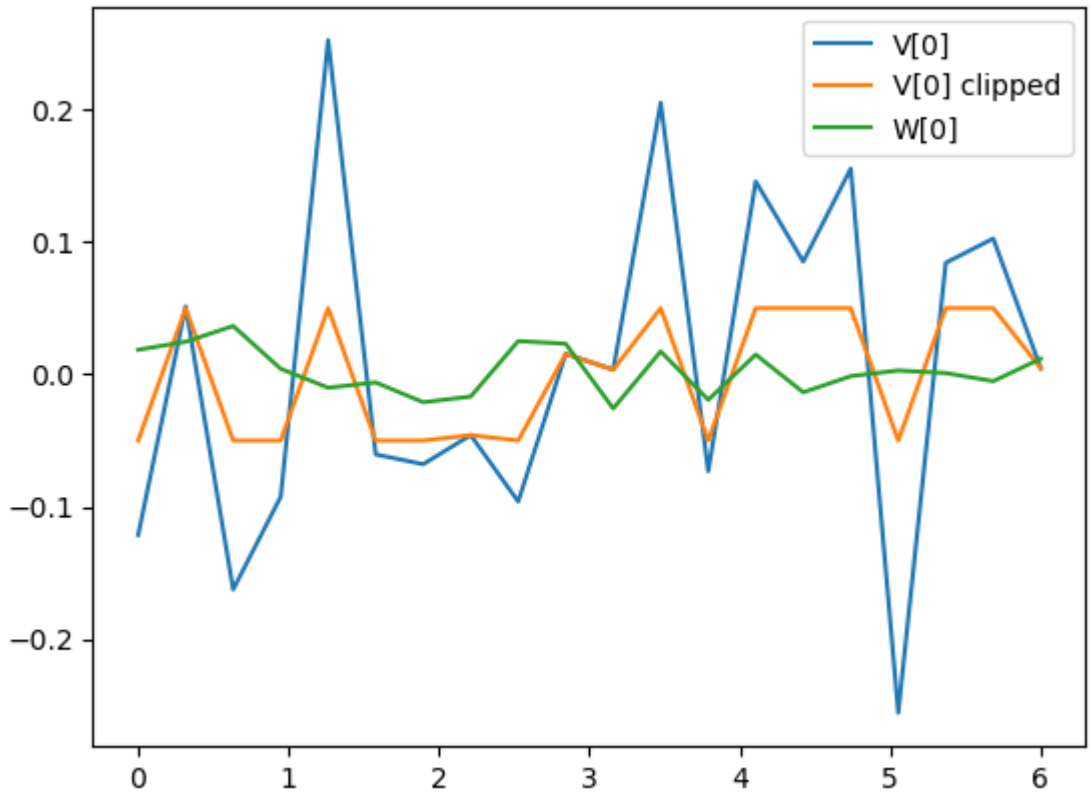
In [16]:
```
plot_estimator_response(timepts, est0, U0, V*0, Y0, W*0)
```



Bounded disturbances

Another thing that the MHE can handled is input distributions that are bounded. We implement that here by carrying out the optimal estimation problem with constraints.

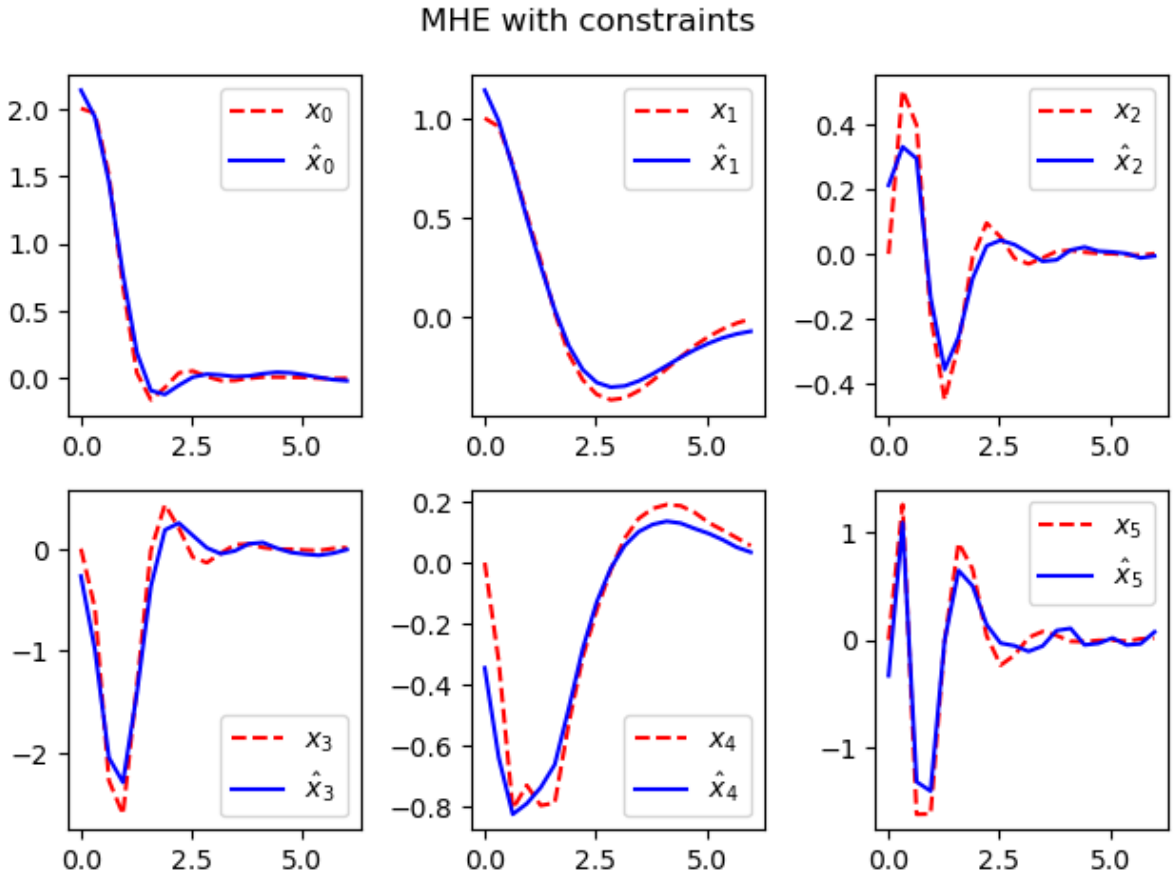
In [17]:
```
V_clipped = np.clip(V, -0.05, 0.05)

plt.plot(timepts, V[0], label="V[0]")
plt.plot(timepts, V_clipped[0], label="V[0] clipped")
plt.plot(timepts, W[0], label="W[0]")
plt.legend();
```

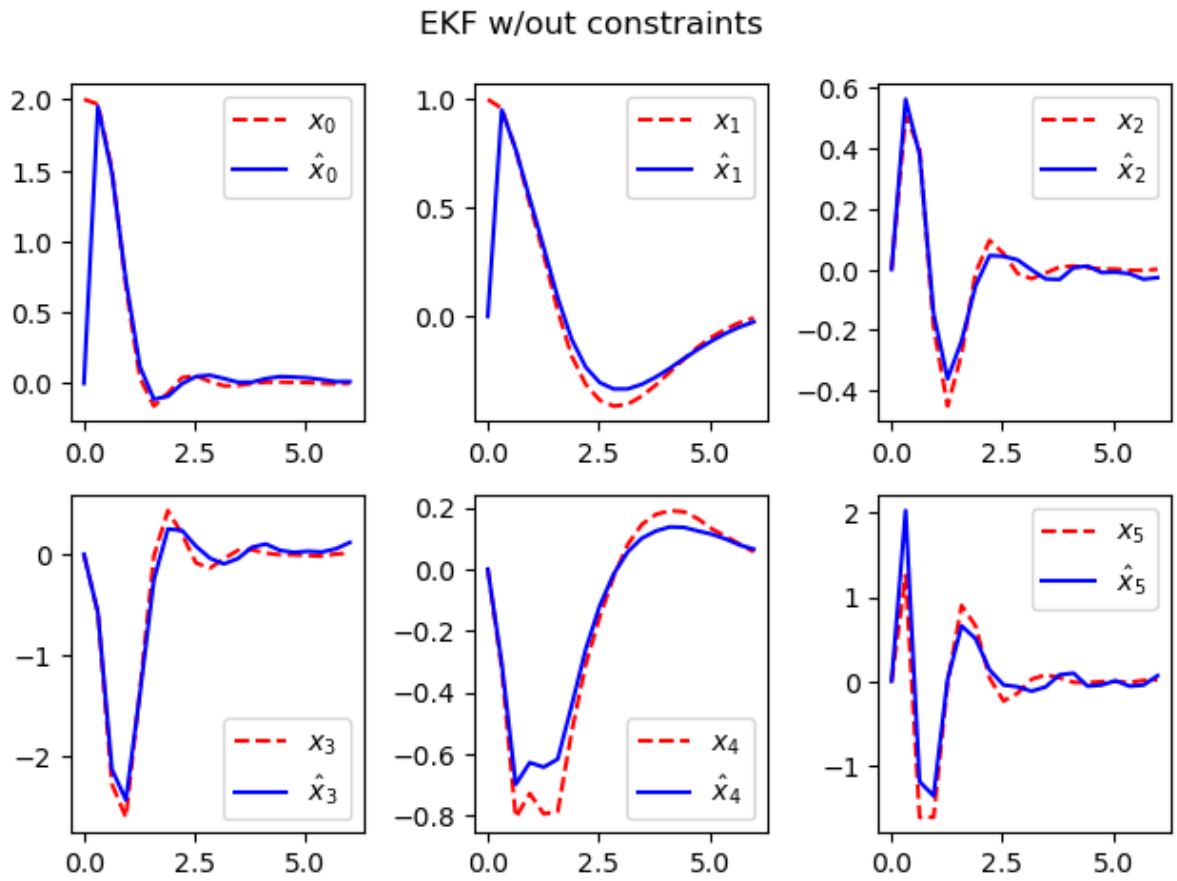


In [18]:
```
uvec = [xe, ue, V_clipped, W]
clipped_resp = ct.input_output_response(lqr_clsys, timepts, uvec, x0)
U_clipped = clipped_resp.outputs[6:8]        # controller input signals
Y_clipped = clipped_resp.outputs[0:3] + W    # noisy output signals

traj_constraint = opt_.add_disturbance_range_constraint(
    sys, [-0.05, -0.05], [0.05, 0.05])
oep_clipped = opt_.OptimalEstimationProblem(
        sys, timepts, traj_cost, terminal_cost=init_cost,
        trajectory_constraints=traj_constraint)

est_clipped = oep_clipped.compute_estimate(
    Y_clipped, U_clipped, X0=lqr0_resp.states[:, 0])
plot_state_comparison(timepts, est_clipped.states, lqr_resp.states)
plt.suptitle("MHE with constraints")
plt.tight_layout()

plt.figure()
```

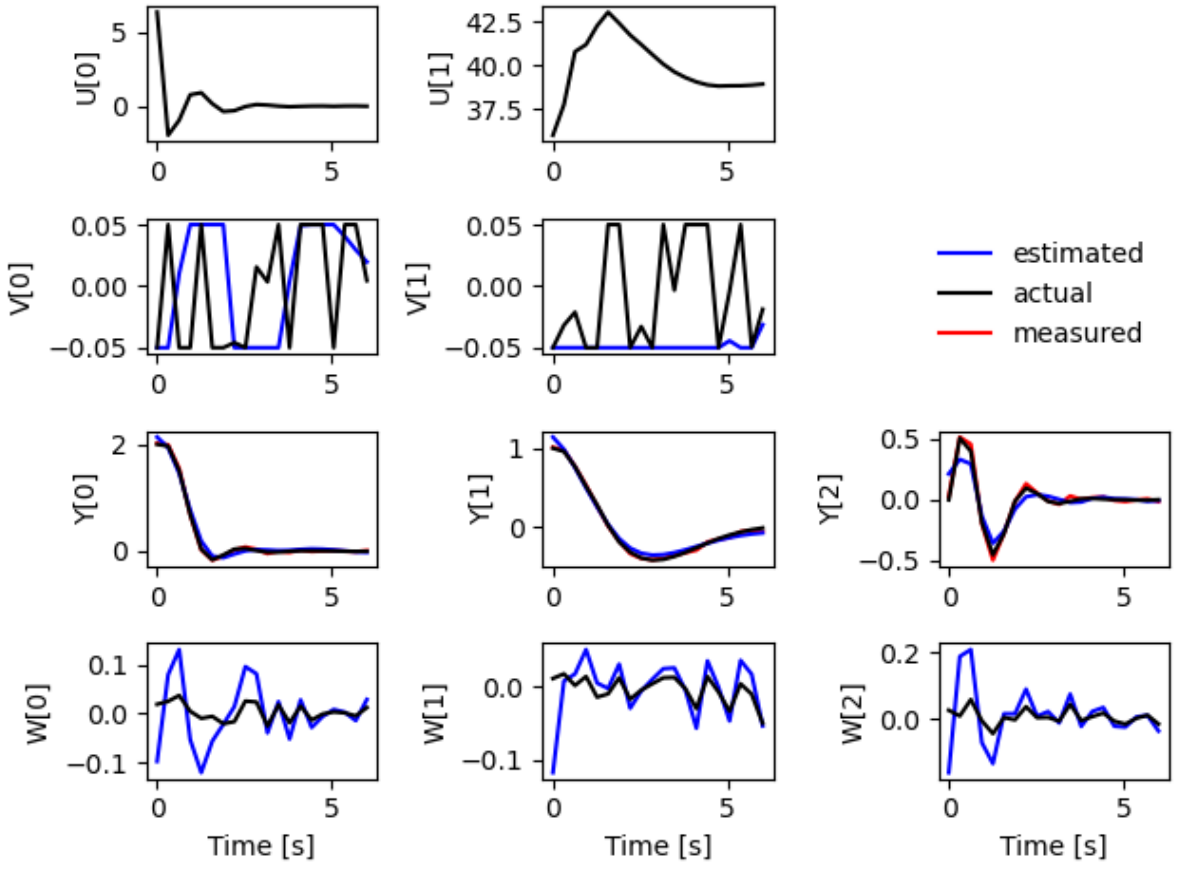
```
ekf_unclipped = ct.input_output_response(
    ekf, timepts, [clipped_resp.states, clipped_resp.outputs[6:8]],
    X0=[xe, P0.reshape(-1)])

plot_state_comparison(timepts, ekf_unclipped.outputs, lqr_resp.states)
plt.suptitle("EKF w/out constraints")
plt.tight_layout()
```

```
Summary statistics:
* Cost function calls: 4381
* Constraint calls: 4570
* Final cost: 679.0897644206816
```





In [19]:
```
plot_estimator_response(timepts, est_clipped, U, V_clipped, Y, W)
```



Moving Horizon Estimation (MHE)

We can now move to implementation of a moving horizon estimator, using our fixed horizon optimal estimator.

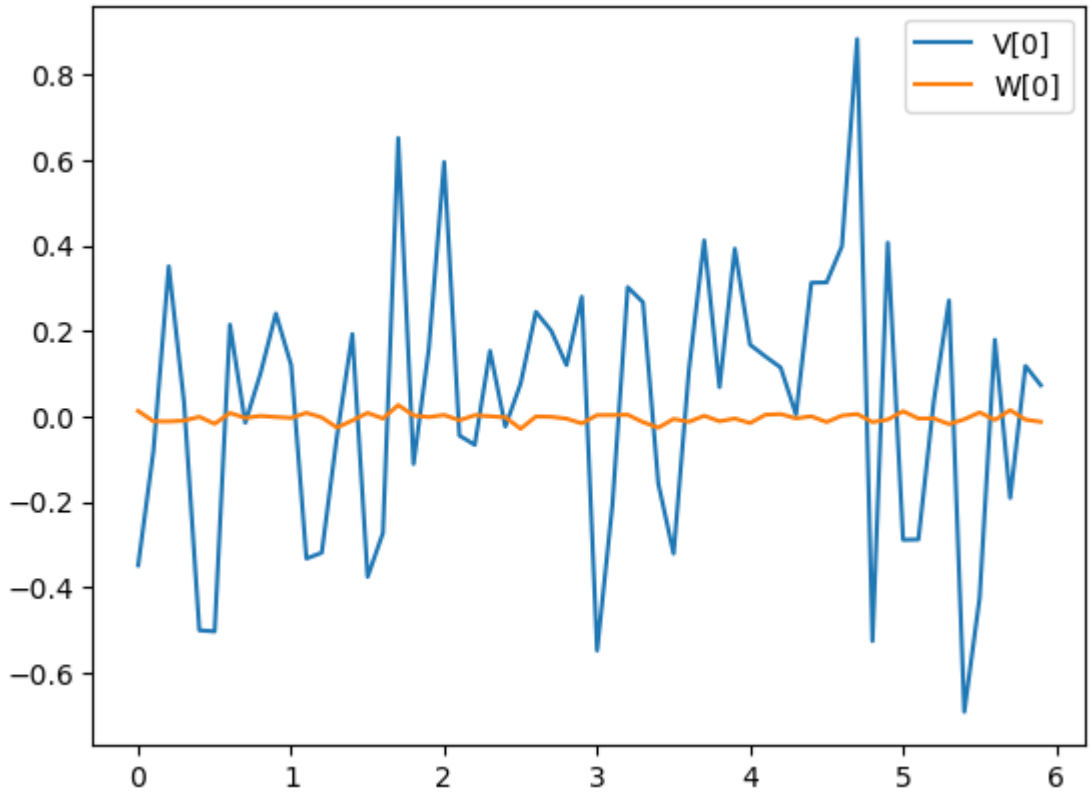
In [20]:
```
# Use a shorter horizon
mhe_timepts = timepts[0:5]
oep = opt_.OptimalEstimationProblem(
        sys, mhe_timepts, traj_cost, terminal_cost=init_cost)

try:
    mhe = oep.create_mhe_iosystem(2)

    est_mhe = ct.input_output_response(
        mhe, timepts, [Y, U], X0=resp.states[:, 0],
        params={'verbose': True}
    )
    plot_state_comparison(timepts, est_mhe.states, lqr_resp.states)
except:
    print("MHE for continuous time systems not implemented")
```

```
MHE for continuous time systems not implemented
```

In [21]:
```
# Create discrete time version of PVTOL
Ts = 0.1
print(f"Sample time: {Ts=}")
dsys = ct.NonlinearIOSystem(
    lambda t, x, u, params: x + Ts * sys.updfcn(t, x, u, params),
    sys.outfcn, dt=Ts, states=sys.state_labels,
    inputs=sys.input_labels, outputs=sys.output_labels,
)
print(dsys)
```

```
Sample time: Ts=0.1
<NonlinearIOSystem>: sys[8]
Inputs (4): ['F1', 'F2', 'Dx', 'Dy']
Outputs (3): ['x', 'y', 'theta']
States (6): ['x0', 'x1', 'x2', 'x3', 'x4', 'x5']

Update: <function <lambda> at 0x7f8711577eb0>
Output: <function <lambda> at 0x7f8711453640>
```

In [22]:
```
# Create a new list of time points
timepts = np.arange(0, Tf, Ts)

# Create representative process disturbance and sensor noise vectors
# np.random.seed(117)           # avoid figures changing from run to run
V = ct.white_noise(timepts, Qv)
# V = np.clip(V0, -0.1, 0.1)    # Hold for later
W = ct.white_noise(timepts, Qw, dt=Ts)
# plt.plot(timepts, V0[0], 'b--', label="V[0]")
plt.plot(timepts, V[0], label="V[0]")
plt.plot(timepts, W[0], label="W[0]")
plt.legend();
```

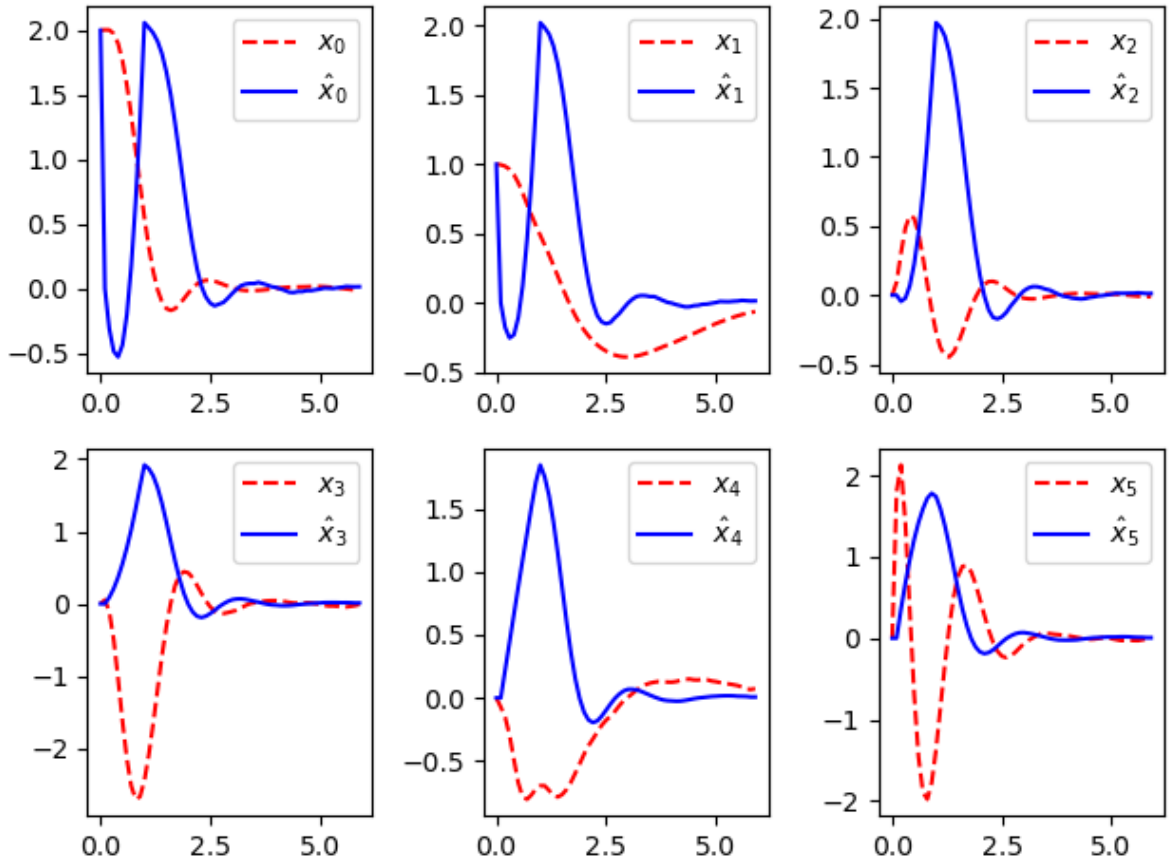


In [23]:
```
# Generate a new trajectory over the longer time vector
uvec = [xd, ud, V, W*0]
lqr_resp = ct.input_output_response(lqr_clsys, timepts, uvec, x0)
U = lqr_resp.outputs[6:8]                    # controller input signals
Y = lqr_resp.outputs[0:3] + W                # noisy output signals
```

In [24]:
```
mhe_timepts = timepts[0:10]
oep = opt_.OptimalEstimationProblem(
        dsys, mhe_timepts, traj_cost, terminal_cost=init_cost)
mhe = oep.create_mhe_iosystem(2)

mhe_resp = ct.input_output_response(
    mhe, timepts, [Y, U], X0=x0,
    params={'verbose': True}
)
plot_state_comparison(timepts, mhe_resp.states, lqr_resp.states)
```

```
[ 3.02893748e-05 -8.92735228e-01 -1.09812197e-04  1.55766294e-04
 -5.35577985e+00 -2.22566606e-04]
[ 9.00190517e-01 -4.38823921e-01  2.50955461e-03  1.45382867e+00
 -3.76584289e+00  1.90035997e+00]
[ 1.62441472  0.00785815  0.23120457  2.44168632 -2.36279755  3.11266398]
[ 2.17347372  0.41278324  0.575706    2.79457814 -1.23319028  2.83927853]
[ 2.52299923  0.74024754  0.84319935  2.56220044 -0.50664436  1.83743951]
[ 2.64800264  0.9654128   0.94537311  1.86810849 -0.17012558  0.58891   ]
[ 2.51313527  1.06120988  0.85628162  0.77692815 -0.11114535 -0.63137939]
[ 2.13413238  1.00233128  0.60232514 -0.57108649 -0.24383907 -1.63136874]
[ 1.53551452  0.81335452  0.23605535 -1.94349535 -0.49098356 -2.27073635]
[ 0.80614189  0.54163902 -0.11084586 -3.11176977 -0.86725216 -2.31323852]
[ 0.46960836  0.45574204 -0.407677   -2.98004333 -0.84389199 -2.15874013]
[ 0.17950828  0.38316786 -0.59231185 -2.64129762 -0.86940362 -1.65511214]
[-0.03051399  0.32015159 -0.65004522 -2.13780591 -0.88667637 -0.93472188]
[-0.16823455  0.25338239 -0.61676849 -1.53801435 -0.8955483  -0.19407014]
[-0.2458217   0.19223827 -0.52230399 -0.92814913 -0.86684625  0.44663941]
[-0.26056382  0.12752371 -0.39521046 -0.35330551 -0.81747741  0.92719147]
[-0.21841332  0.03955738 -0.25038314  0.14685681 -0.78898106  1.22675305]
[-0.1608708  -0.03888045 -0.12082192  0.50322063 -0.7408729   1.31443938]
[-0.07750611 -0.11013107 -0.00615192  0.73371446 -0.66989186  1.25790916]
[-0.01040619 -0.1640883   0.08199594  0.7994161  -0.57552011  1.0728022 ]
[ 0.04598209 -0.20964442  0.13860685  0.75892614 -0.48025644  0.80235771]
[ 0.09187582 -0.24531054  0.16697913  0.64403268 -0.39028435  0.50689445]
[ 0.11100001 -0.27686976  0.16485389  0.46746205 -0.31607077  0.21027143]
[ 0.11619549 -0.30142338  0.13667614  0.29234904 -0.24190794 -0.05328713]
[ 0.10647626 -0.32572151  0.10462736  0.11037089 -0.18324714 -0.23740799]
[ 0.0899348  -0.344917    0.06221106 -0.02552837 -0.13825225 -0.35822856]
[ 0.04843896 -0.36442414  0.02561636 -0.17404995 -0.09925925 -0.40691566]
[ 0.0290322  -0.37794485 -0.00200196 -0.2358156  -0.06239232 -0.38519386]
[ 0.00643738 -0.38856435 -0.019418   -0.27286428 -0.02644689 -0.3136669 ]
[-0.01326671 -0.39561968 -0.03385826 -0.26575157  0.00282791 -0.22875117]
[-0.02979991 -0.38255397 -0.0405699  -0.23244964  0.05450615 -0.13738841]
[-0.03509788 -0.3753297  -0.02898778 -0.19943202  0.0890875  -0.04051309]
[-0.0310634  -0.37000717 -0.03349905 -0.11771032  0.11134629 -0.00221818]
[-0.0223932  -0.36822506 -0.03573061 -0.03414659  0.11667042  0.0305285 ]
[-0.02458646 -0.35994393 -0.02983481 -0.00277795  0.13182708  0.05623676]
[-0.03374939 -0.34752409 -0.02317832  0.00907301  0.14321375  0.07118627]
[-0.02376386 -0.34152219 -0.01659138  0.05004111  0.14295438  0.06794458]
[-0.01841978 -0.32593075 -0.00447859  0.05780092  0.15127711  0.0675289 ]
[-0.00777559 -0.31614288  0.0007041   0.07226709  0.14763132  0.05997825]
[-0.00897701 -0.29582337  0.00878331  0.04876779  0.16966059  0.05620806]
[ 0.00057175 -0.27289005  0.01588684  0.05421619  0.18894047  0.05953133]
[ 0.00227706 -0.26557468  0.01247487  0.05004236  0.17991929  0.03588911]
[ 0.01196987 -0.25365417  0.01053246  0.05038835  0.17432508  0.01120883]
[ 0.01993653 -0.241557    0.00795688  0.05235987  0.16607317 -0.00495462]
[ 0.01413767 -0.22932728  0.01549281  0.00525857  0.16104604 -0.00489514]
[ 0.0137971  -0.22606894  0.01774885 -0.01910298  0.13524766 -0.00927477]
[ 0.00677938 -0.21486686  0.01471964 -0.03884504  0.13098343 -0.01078787]
[ 0.01012876 -0.20218515  0.01083723 -0.03384748  0.12600436 -0.01080272]
[ 0.01235595 -0.18399195  0.01345429 -0.0377114   0.12932631  0.01143631]
[ 0.00774951 -0.16203548  0.004115   -0.03109128  0.14337228 -0.0047607 ]
[ 0.0040441  -0.15885835 -0.00369742 -0.02676868  0.12067969 -0.02398774]
[ 0.0135094  -0.14186364 -0.00207362 -0.01150946  0.12431228 -0.02572851]
[ 0.01140233 -0.12062602 -0.00171699 -0.02235497  0.1355986  -0.04077058]
[ 0.01027813 -0.10922087 -0.0033524  -0.01985724  0.12959842 -0.04214327]
```

```
[ 0.00283168 -0.09705975 -0.00655375 -0.02323846  0.12195697 -0.03563937]
[-0.00209423 -0.0874986  -0.01312655 -0.01673723  0.11128636 -0.03894094]
[ 0.00252041 -0.08078006 -0.01415605  0.00082072  0.09891418 -0.03844952]
[-0.00112246 -0.07767659 -0.01509356  0.00650644  0.08569608 -0.02753261]
[ 0.00129751 -0.06377385 -0.01976016  0.02391168  0.09233928 -0.01982272]
[-0.00812025 -0.06424363 -0.01605592  0.00840027  0.06880473 -0.00090044]
```

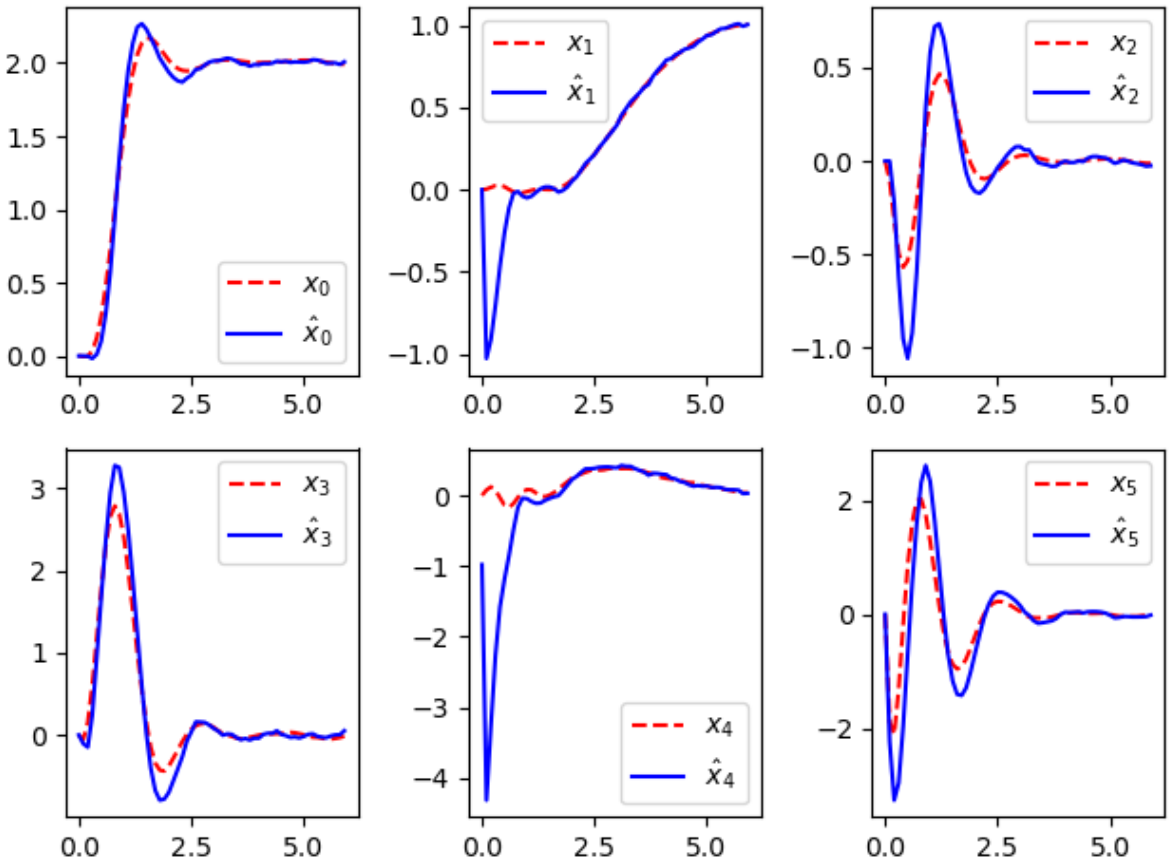


In [25]:
```
# Resimulate starting at the origin and moving to the "initial" condition
uvec = [x0, ue, V, W*0]
lqr_resp = ct.input_output_response(lqr_clsys, timepts, uvec, xe)
U = lqr_resp.outputs[6:8]                    # controller input signals
Y = lqr_resp.outputs[0:3] + W                # noisy output signals
```

In [26]:
```
mhe_timepts = timepts[0:8]
oep = opt_.OptimalEstimationProblem(
        dsys, mhe_timepts, traj_cost, terminal_cost=init_cost)
mhe = oep.create_mhe_iosystem(2)

mhe_resp = ct.input_output_response(
    mhe, timepts, [Y, U],
    params={'verbose': True}
)
plot_state_comparison(timepts, mhe_resp.outputs, lqr_resp.states)
```

```
[ 9.78010520e-06 -5.87007021e-01 -7.31266349e-05  5.54891739e-05
 -4.39707479e+00 -1.83648326e-04]
[ 0.00736218 -0.57854252  0.00350306 -0.0950746  -3.3223201  -2.22306071]
[-0.00999594 -0.47466094 -0.28341498 -0.07084596 -2.27537953 -3.37536205]
[-0.01693926 -0.32447979 -0.64939326  0.28011895 -1.40477619 -3.01704247]
[ 0.0187369  -0.17797896 -0.8727199   0.83290784 -0.84827855 -1.83090611]
[ 0.13766689 -0.05939554 -0.88355267  1.51181001 -0.52353733 -0.36686326]
[ 0.34700343  0.00969438 -0.6985863   2.23264314 -0.31361882  1.00739177]
[ 0.66075823  0.00792238 -0.3682074   2.8864743  -0.18805243  2.10205808]
[ 1.01723972 -0.02651168  0.009088    3.27417461 -0.12287228  2.74057788]
[ 1.35794249 -0.04283945  0.31093447  3.29700668 -0.0649495   2.71394113]
[ 1.65227797 -0.03773428  0.50268693  3.01785152 -0.01466122  2.16947976]
[ 1.88401158 -0.01704211  0.59757533  2.50661915  0.01509984  1.36903001]
[2.05734802 0.00378758 0.60643925 1.87877845 0.01996919 0.5035799 ]
[ 2.15069527  0.01571815  0.54211568  1.17164792  0.01172395 -0.2921152 ]
[ 2.181833    0.01869066  0.4406343   0.49511529 -0.01653334 -0.88780216]
[ 2.17746924  0.01112436  0.31755898 -0.06825326 -0.04257273 -1.2583884 ]
[ 2.15462843 -0.00836021  0.19653547 -0.46982342 -0.05717584 -1.40136768]
[ 2.10465149 -0.00406305  0.0755814  -0.71932209  0.00345127 -1.38092725]
[ 2.06493212  0.00893536 -0.01409303 -0.80372865  0.08642347 -1.17894554]
[ 2.01496392  0.03453017 -0.0758257  -0.7878993   0.17752554 -0.88152223]
[ 1.9699209   0.05504932 -0.11538598 -0.68699942  0.23248205 -0.56598764]
[ 1.93306848  0.08760163 -0.12676862 -0.55236984  0.29722551 -0.24589699]
[ 1.90831674  0.12442592 -0.11685731 -0.38983356  0.35187987  0.02910383]
[ 1.91346218  0.15065336 -0.09767081 -0.17858707  0.35545003  0.22233443]
[ 1.91975314  0.17880654 -0.06790081 -0.03344081  0.36442399  0.33751025]
[ 1.94372142  0.21743002 -0.04120257  0.1180216   0.38115571  0.37390505]
[ 1.9467001   0.24967559 -0.00723327  0.1507875   0.38795001  0.37116436]
[1.9683087  0.28553216 0.02294066 0.17691495 0.39345956 0.33577127]
[1.98885465 0.31833091 0.04810004 0.17123428 0.39054264 0.28206722]
[2.0002923  0.35502911 0.05723773 0.13805021 0.39101764 0.19939221]
[2.00522802 0.41194875 0.05018733 0.09910695 0.42554819 0.09049585]
[2.01871042 0.45165934 0.05656693 0.05486064 0.41645599 0.02837576]
[ 2.02826309  0.49201029  0.03115623  0.04488653  0.41806669 -0.08538059]
[ 2.02533757  0.52198701  0.00617983  0.01378326  0.38867645 -0.14971357]
[ 2.01544444e+00  5.53264100e-01 -2.02403201e-04 -3.06131807e-02
  3.64458258e-01 -1.47418892e-01]
[ 1.99975642  0.58870668 -0.00753574 -0.06471843  0.34401339 -0.13299805]
[ 1.99740097  0.61024751 -0.01630748 -0.05877533  0.29881069 -0.12263134]
[ 1.98594305  0.65836635 -0.01733958 -0.07918322  0.3260861  -0.09384464]
[ 1.99349184  0.6934056  -0.00819729 -0.0471812   0.32243196 -0.02283591]
[ 1.99108296e+00  7.25415141e-01 -1.98575378e-03 -4.17878406e-02
  3.11681686e-01  1.70093718e-02]
[ 1.99641589  0.75965072 -0.00919436 -0.00274225  0.30450306  0.01568833]
[ 1.99343376  0.77207528 -0.01018389  0.01062588  0.25690875  0.02832477]
[ 1.99730696  0.79695442 -0.00384263  0.01527424  0.24545811  0.04327894]
[ 2.00747516  0.81475366 -0.00605201  0.04580114  0.21164722  0.02866443]
[2.00599016 0.83478253 0.0102748  0.01754121 0.19774148 0.04798473]
[2.01100539 0.8458534  0.01906908 0.01130712 0.16452791 0.04920128]
[ 2.00458446  0.8595337   0.02006194 -0.01030924  0.14327959  0.04554036]
[ 2.00844857  0.8834877   0.01652297 -0.00799879  0.15409694  0.0302742 ]
[ 2.00886587  0.90297769  0.01439051 -0.02169155  0.14306139  0.02435163]
[ 2.00661162  0.92447232  0.0039419  -0.01352361  0.14500762 -0.00615776]
[ 2.00926434e+00  9.31311021e-01 -7.53277822e-03  1.58276408e-03
  1.24297299e-01 -4.98481943e-02]
[ 2.02133455e+00  9.49571787e-01 -6.39507404e-04  1.37893410e-02
```

```
  1.25569472e-01 -3.89618508e-02]
[ 2.02024576  0.96701818  0.00887049 -0.00629399  0.12161712 -0.02802327]
[ 2.0128944   0.97236606  0.00261971 -0.02497529  0.09668373 -0.04278901]
[ 2.00140869  0.98347347 -0.00517894 -0.03769797  0.08803087 -0.04641389]
[ 2.00103802  0.995008   -0.00983274 -0.02063716  0.08751109 -0.04165347]
[ 2.00175365  1.00215196 -0.01483419 -0.01370842  0.08164492 -0.05035157]
[ 1.99185065  0.99178577 -0.02219859 -0.01869053  0.03128436 -0.05316498]
[ 2.0046602   1.00156897 -0.02194677  0.0289988   0.03496342 -0.02969137]
[ 1.99724195  1.00544291 -0.01013993  0.01398819  0.03396231  0.01717259]
```



In [ ]: