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 0x7fb8b0be8b80>
Output: <function _pvtol_output at 0x7fb8b0be8670>

Forward: <function _pvtol_flat_forward at 0x7fb8b11dbe20>
Reverse: <function _pvtol_flat_reverse at 0x7fb8b11dab00>

<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 0x7fb8b11dad40>
Output: <function _noisy_output at 0x7fb8b1218160>
```

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)
```

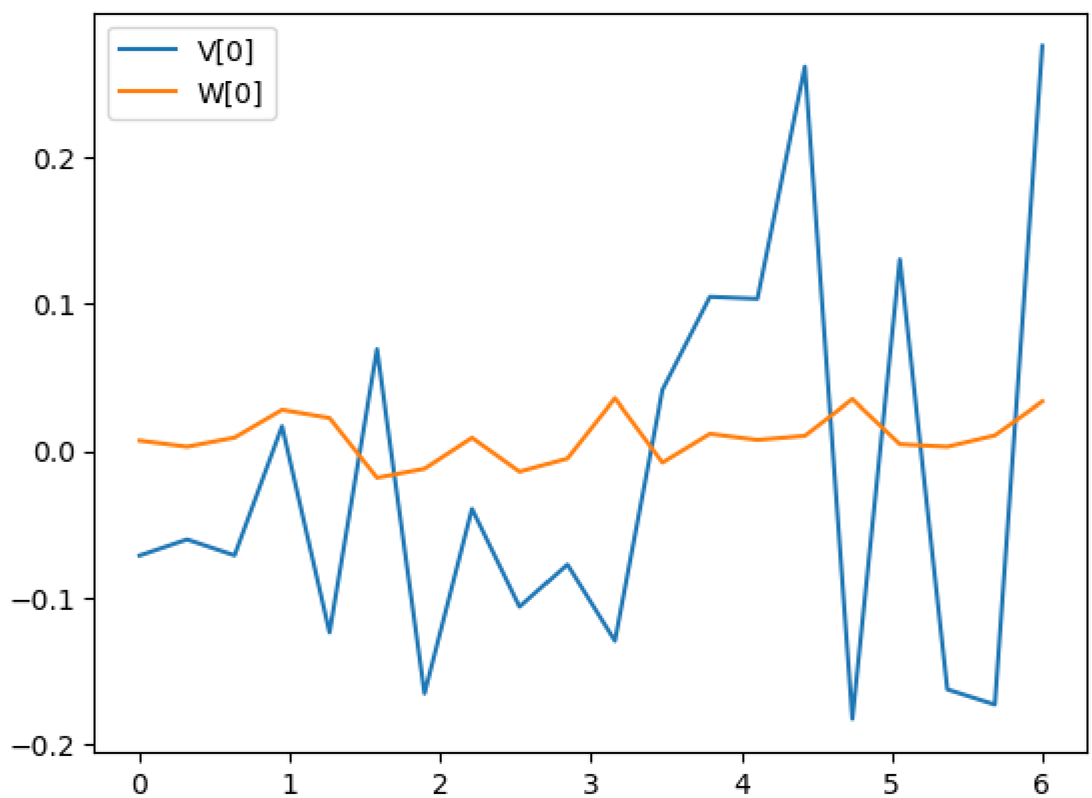
```
<InterconnectedSystem>: F2
Inputs (13): ['xd[0]', 'xd[1]', 'xd[2]', 'xd[3]', 'xd[4]', 'xd[5]', 'ud
[0]', 'ud[1]', 'Dx', 'Dy', 'Nx', 'Ny', 'Nth']
Outputs (8): ['x0', 'x1', 'x2', 'x3', 'x4', 'x5', 'F1', 'F2']
States (6): ['pvtol_noisy_x0', 'pvtol_noisy_x1', 'pvtol_noisy_x2', 'pvtol_n
oisy_x3', 'pvtol_noisy_x4',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

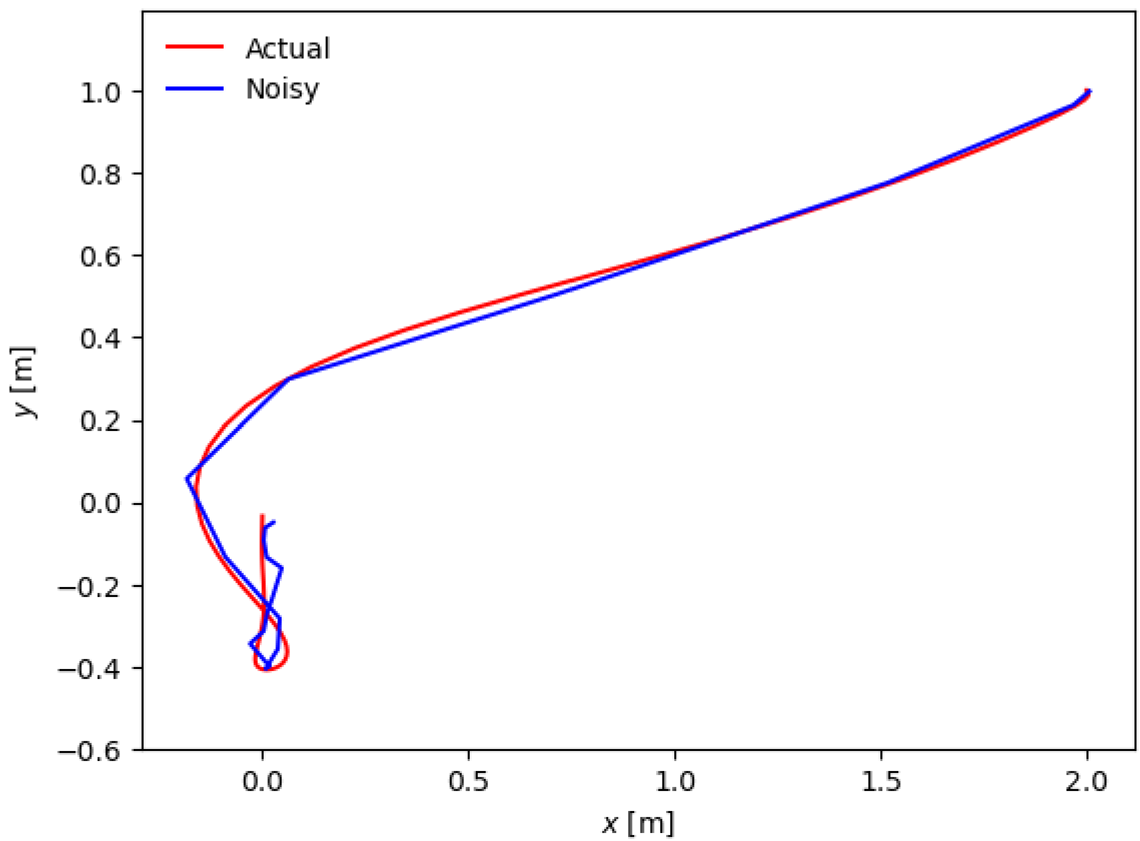
In [6]:

```
# 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();
```

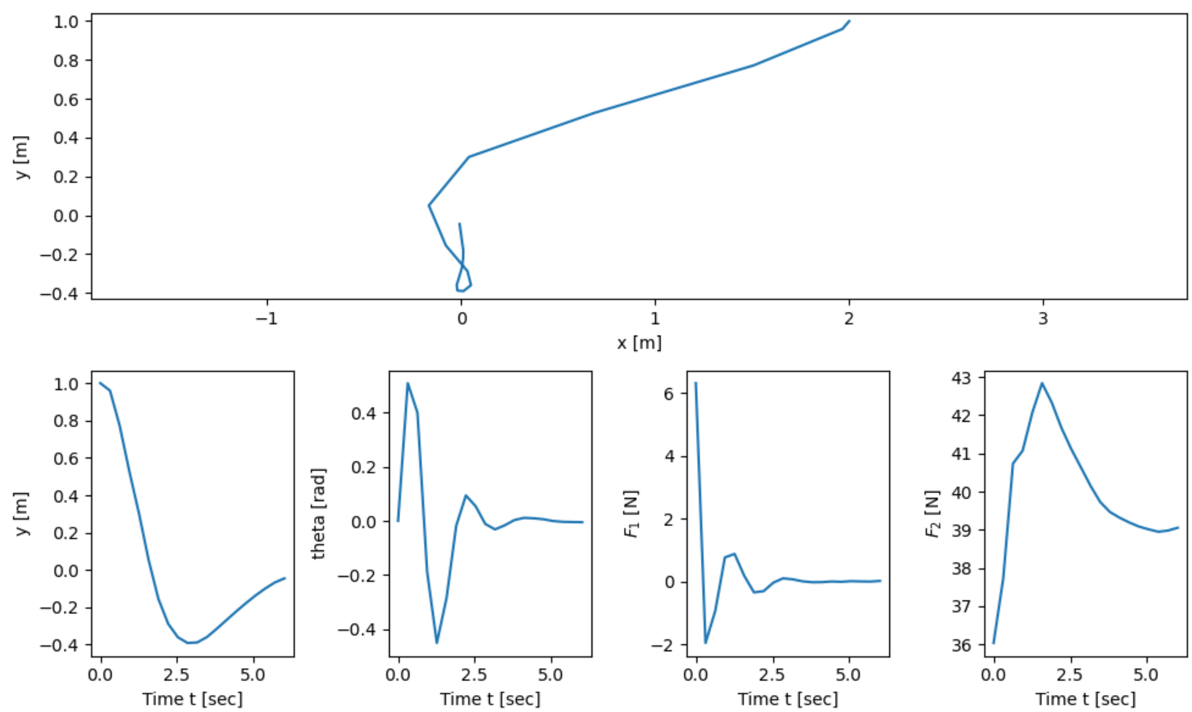


In [7]:

```
# 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]:

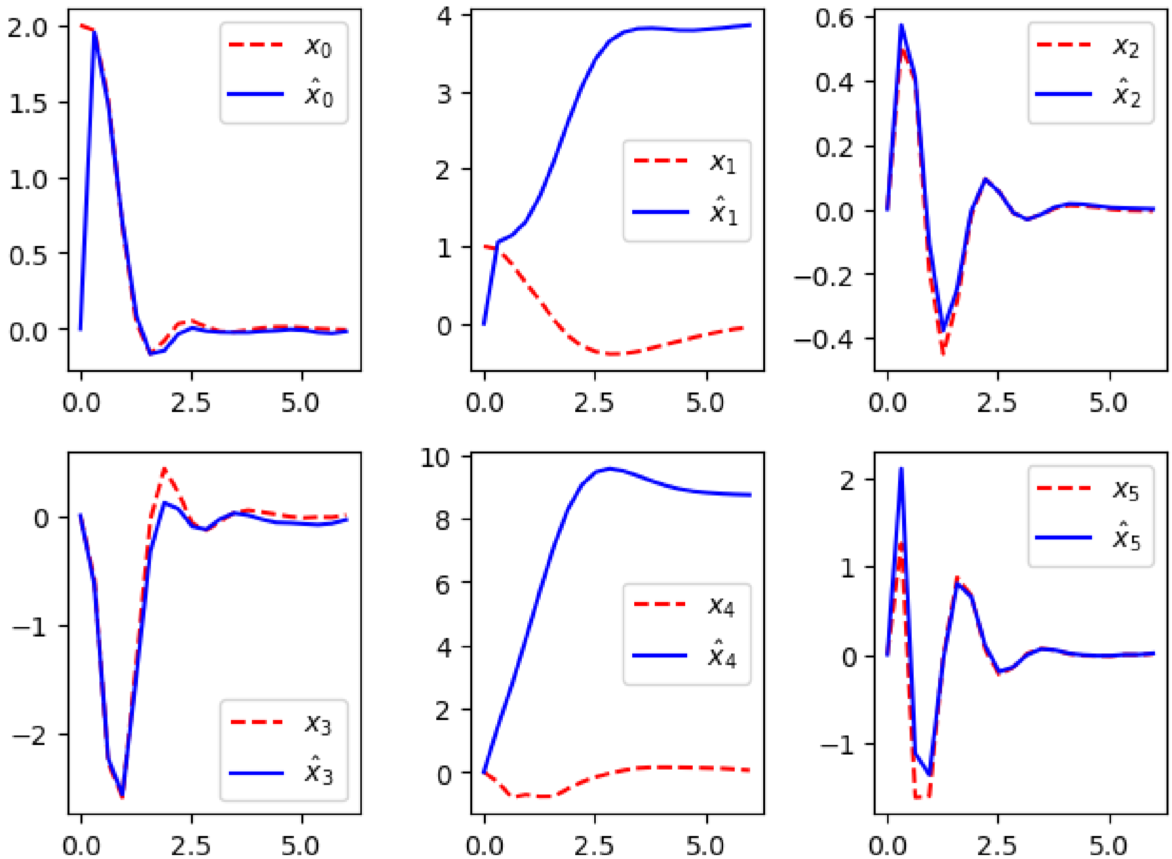
```
# Create a new system with only x, y, theta as outputs
# TODO: add this to pvtol.py?
sys = ct.NonlinearIOSystem(
    pvt._noisy_update, lambda t, x, u, params: x[0:3], name="pvtol_noisy",
    states = [f'x{i}' for i in range(6)],
```

```
    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 0x7fb
8a29cee60>
Output: <function create_estimator_iosystem.<locals>._estim_output at 0x7fb
8a29ceef0>
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

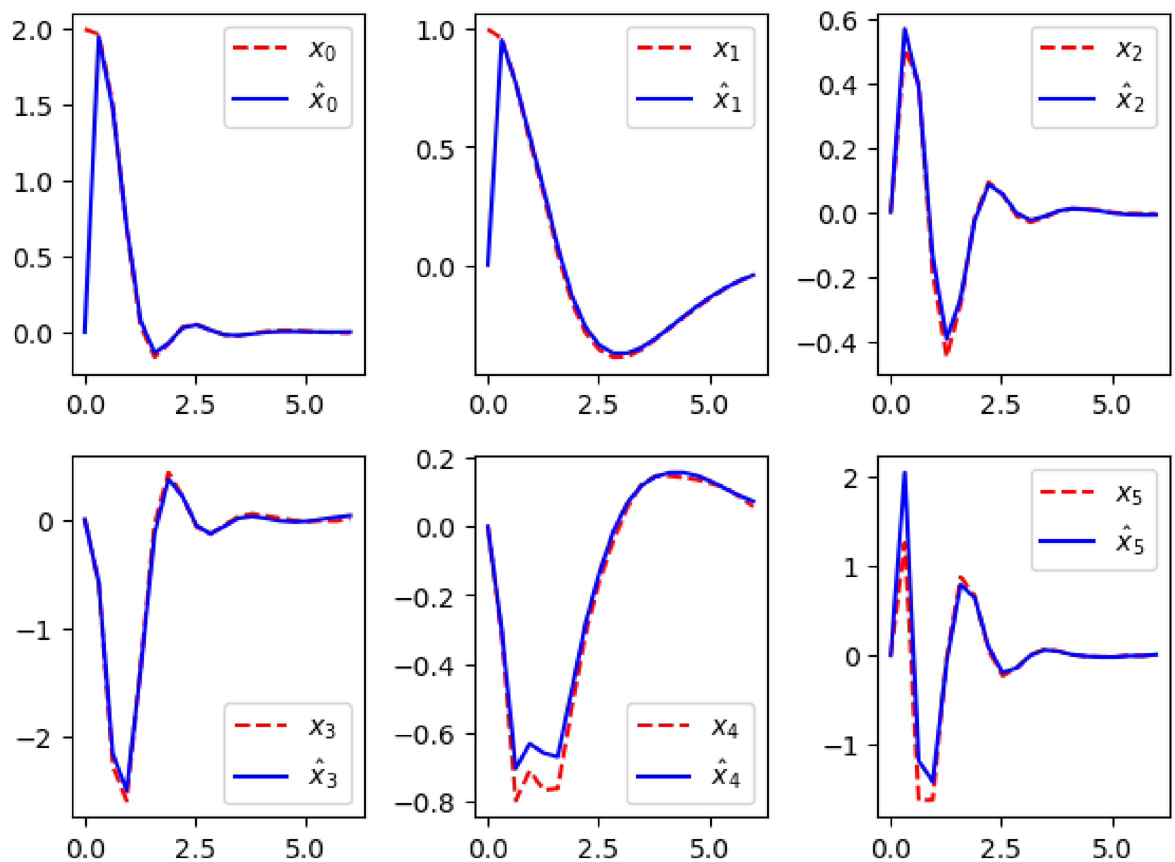
In [11]:
```
# Define the disturbance input and measured output matrices
F = np.array([[0, 0], [0, 0], [0, 0], [1/pvtol.params['m'], 0], [0, 1/pvtol.
C = np.eye(3, 6)

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 0x7fb8906d1e10>
Output: <function estimator_output at 0x7fb8906d1a20>
```

In [12]:

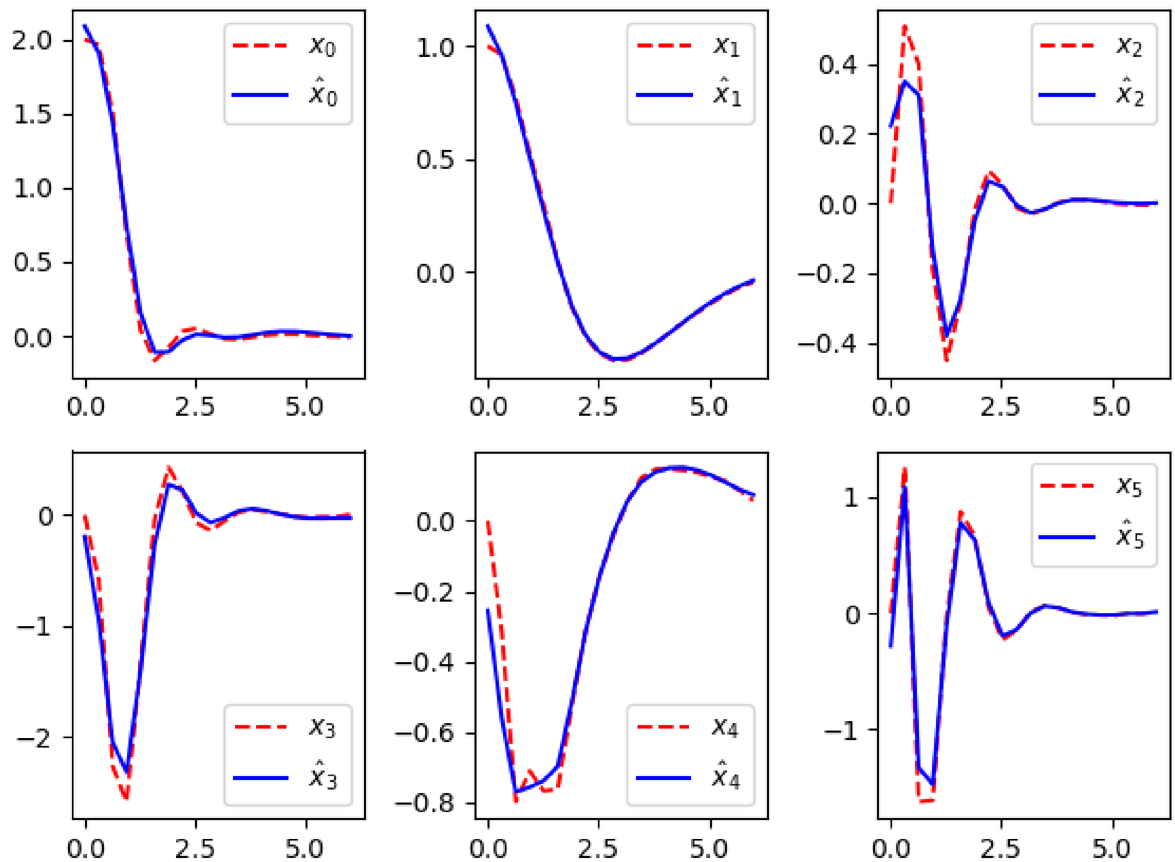
```
ekf_resp = ct.input_output_response(
    ekf, timepts, [lqr_resp.states, lqr_resp.outputs[6:8]],
    X0=[xe, P0.reshape(-1)])
plot_state_comparison(timepts, ekf_resp.outputs, lqr_resp.states)
```



In [13]:

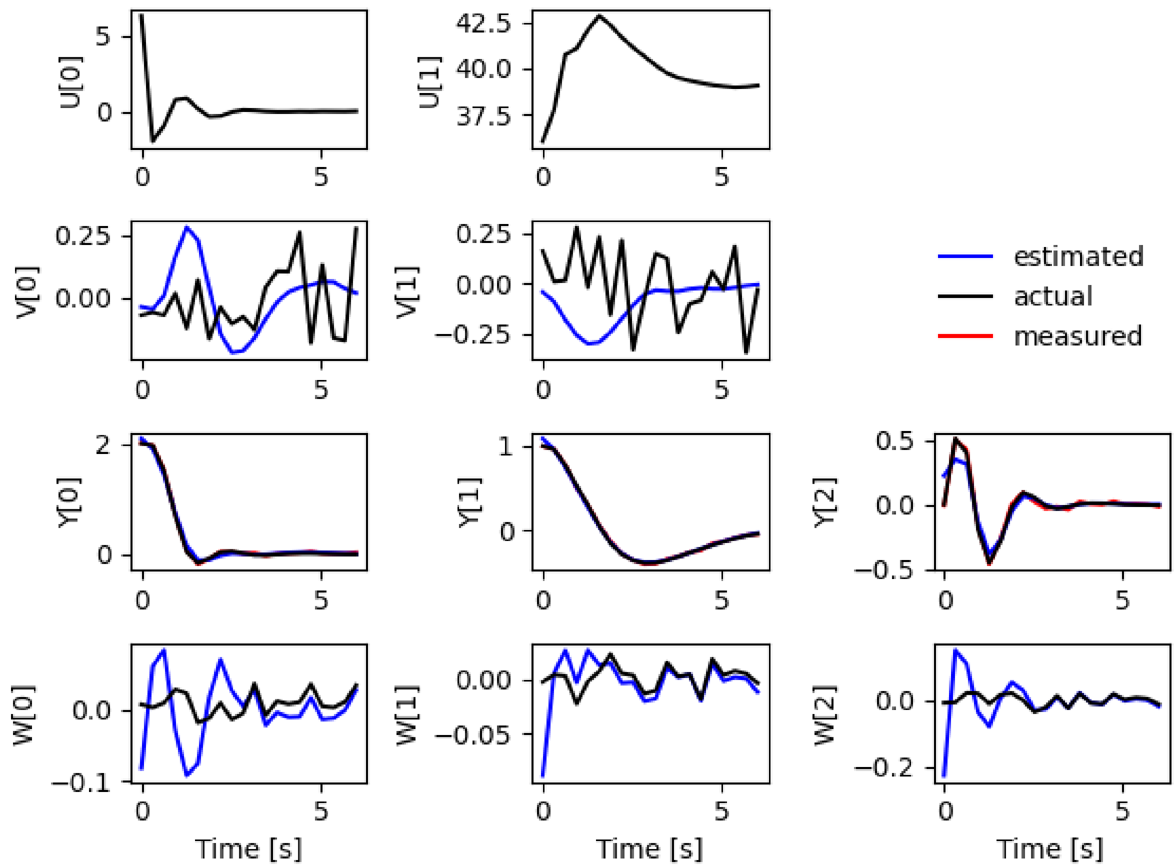
```
# Define the optimal estimation problem
traj_cost = opt_.gaussian_likelihood_cost(sys, Qv, Qw)
init_cost = lambda xhat, x: (xhat - x) @ P0 @ (xhat - x)
oep = opt_.OptimalEstimationProblem(
        sys, timepts, traj_cost, terminal_cost=init_cost)

# 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: 5373
* Final cost: 380.61139713791175
```



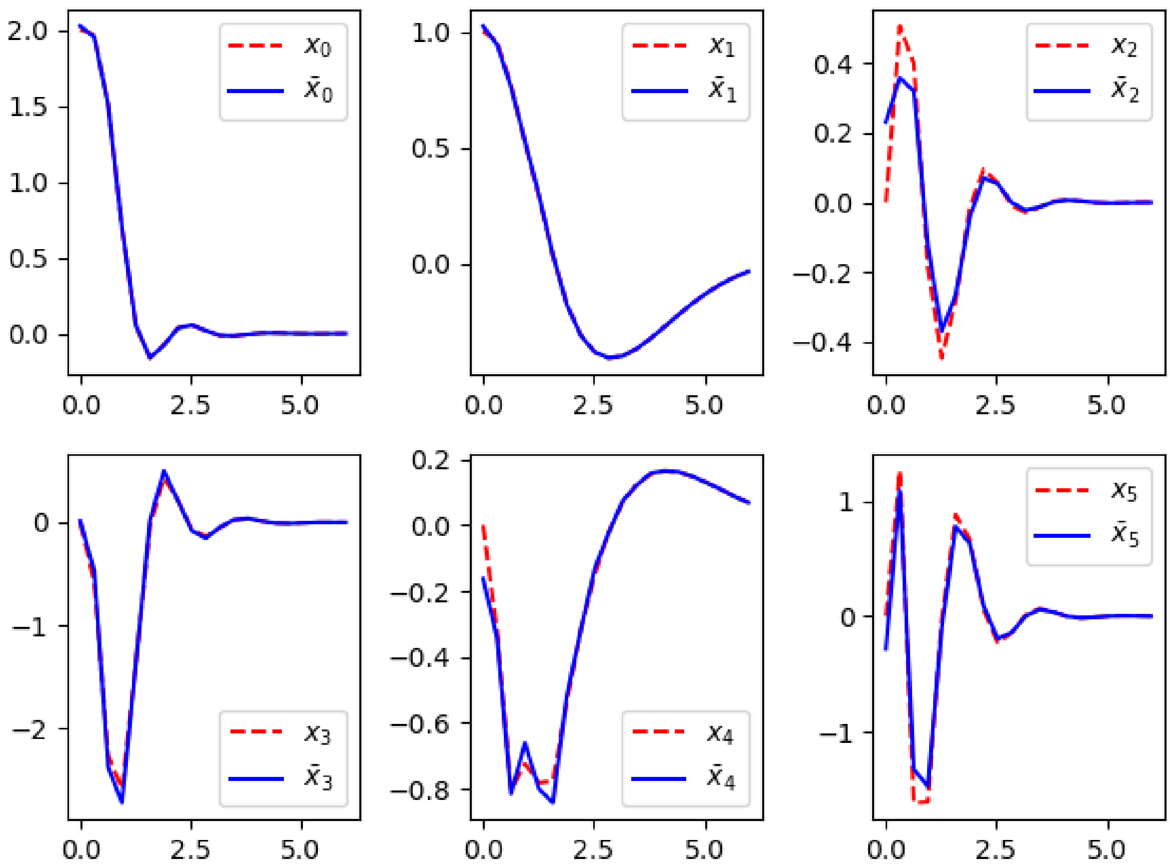
In [14]:

```
# Plot the response of the estimator
plot_estimator_response(timepts, est, U, V, Y, W)
```

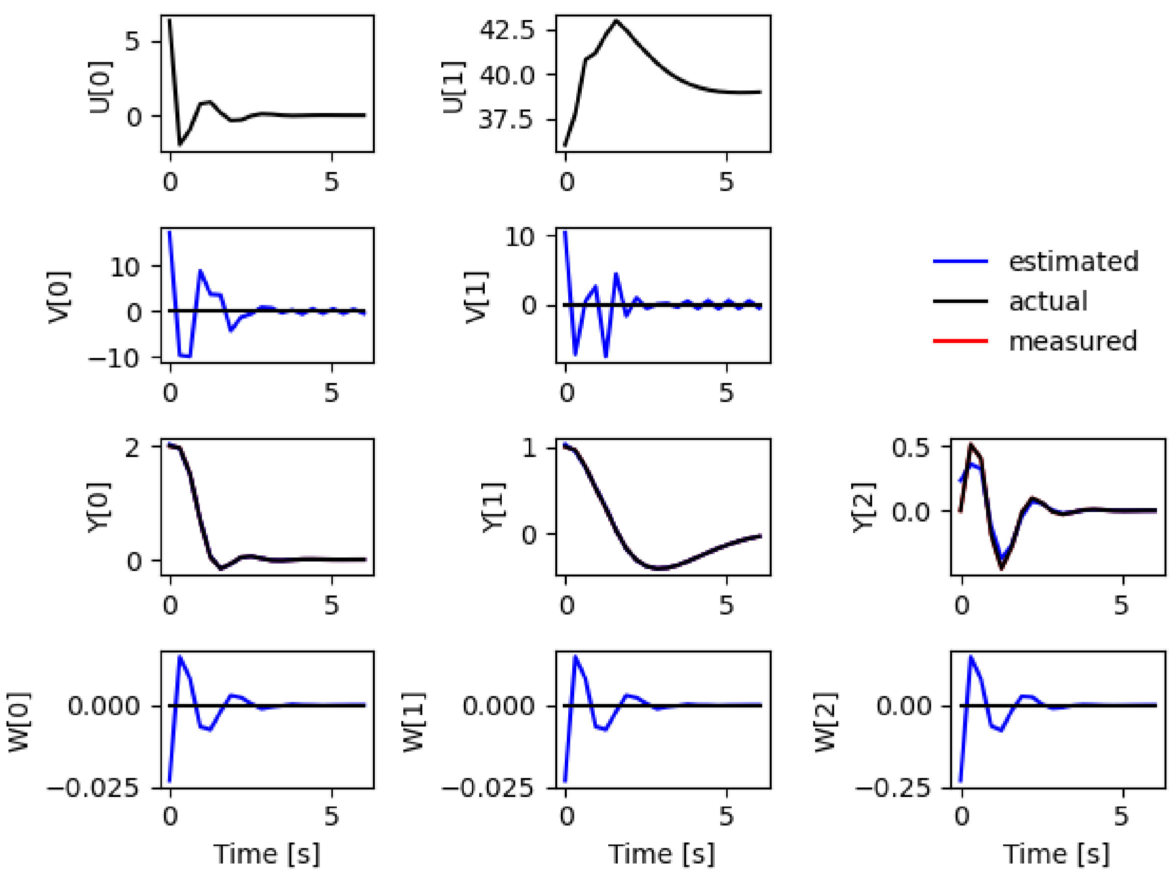



```
In [15]: # Noise free and disturbance free => estimation should be near perfect
noisefree_cost = opt_.gaussian_likelihood_cost(sys, Qv, Qw*1e-6)
oep0 = opt_.OptimalEstimationProblem(
        sys, timepts, noisefree_cost, terminal_cost=init_cost)
est0 = oep0.compute_estimate(Y0, U0, X0=lqr0_resp.states[:, 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
```



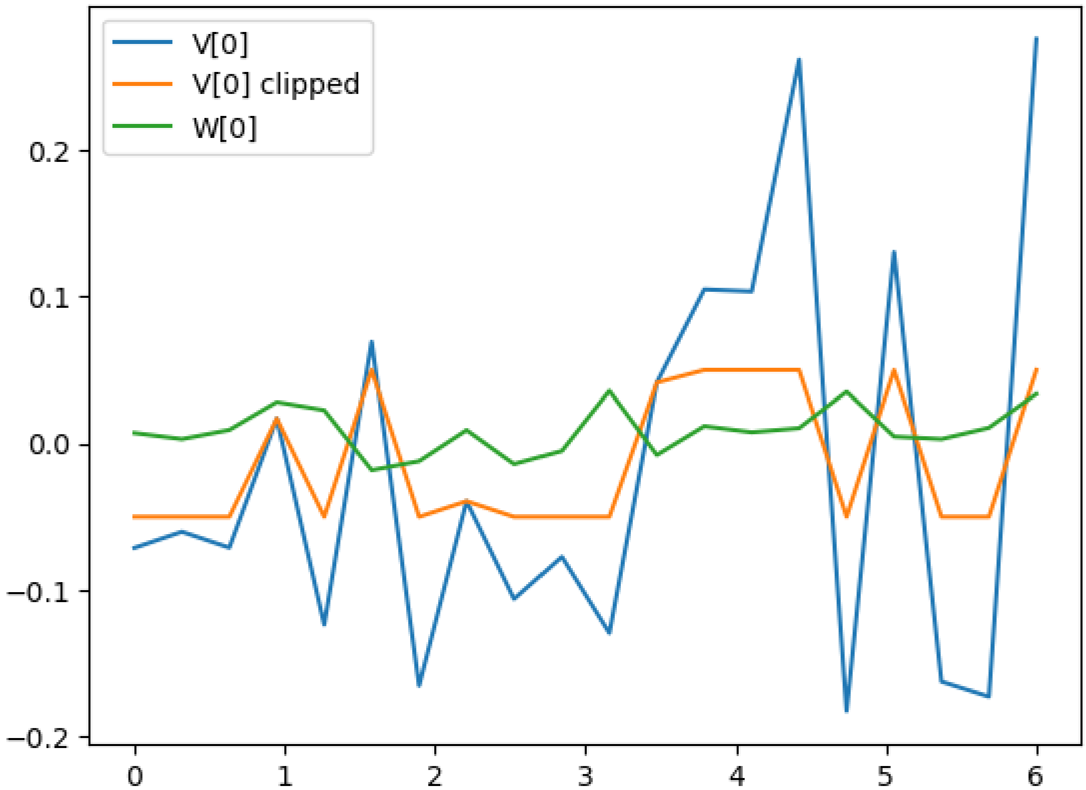
In [16]:
```
plot_estimator_response(timepts, est0, U0, V*0, Y0, W*0)
```



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.

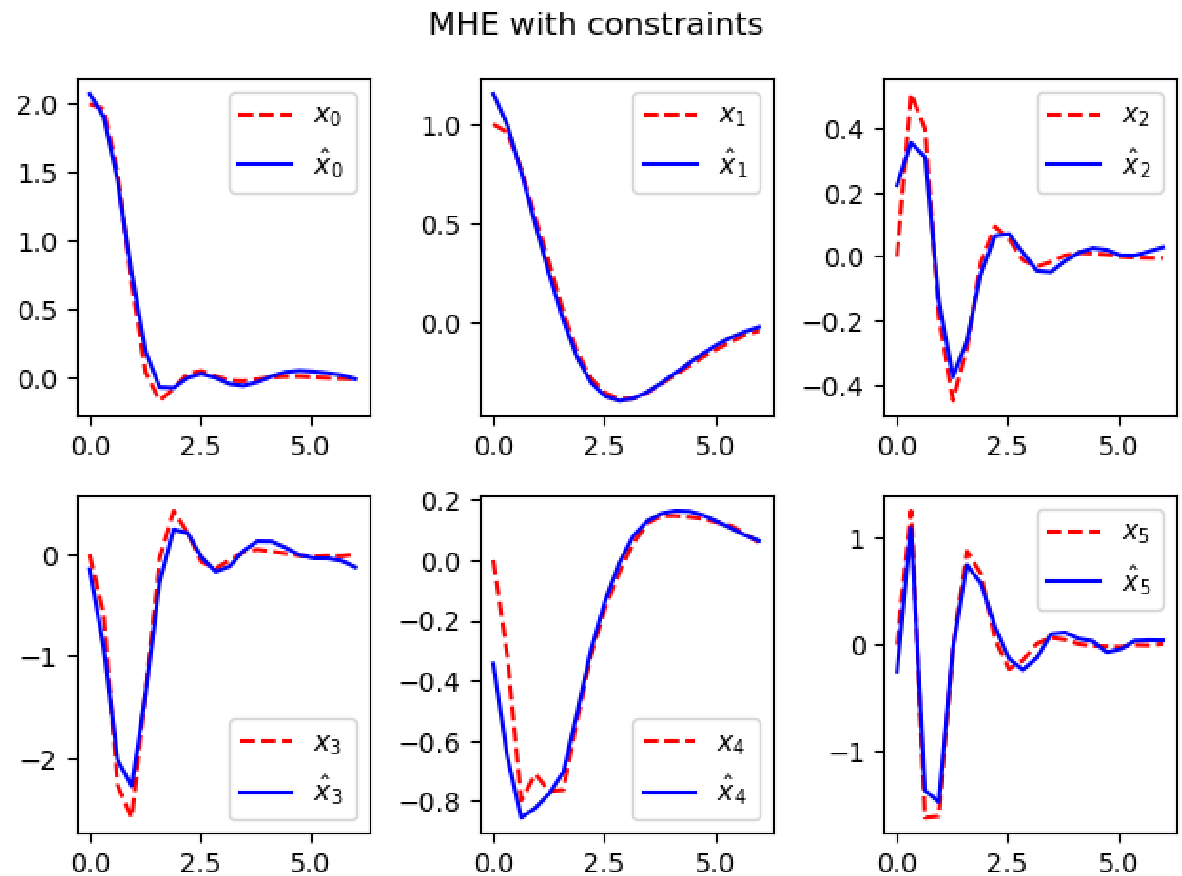
In [17]:
```
V_clipped = np.clip(V, -0.05, 0.05)

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();
```

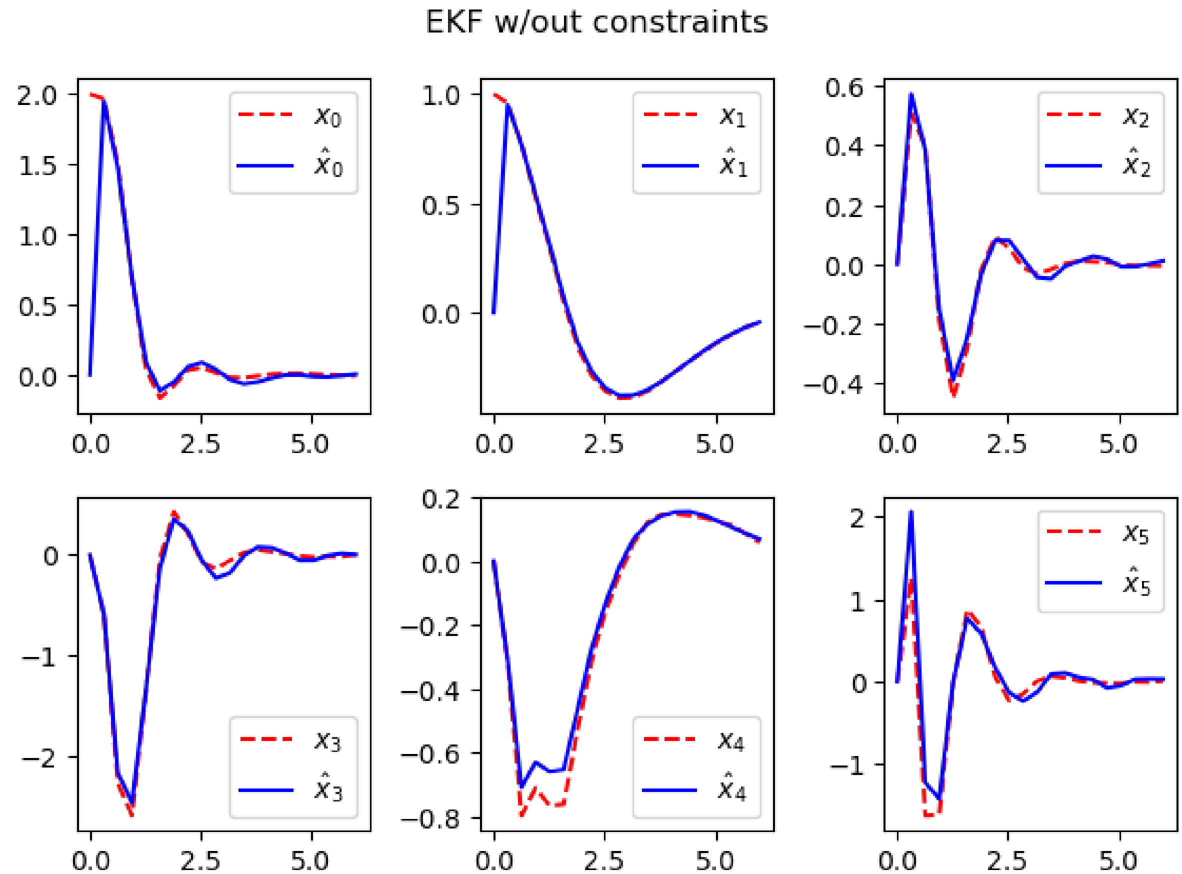


In [18]:
```
uvec = [xe, ue, V_clipped, W]
clipped_resp = ct.input_output_response(lqr_clsys, timepts, uvec, x0)
U_clipped = clipped_resp.outputs[6:8]        # controller input signals
Y_clipped = clipped_resp.outputs[0:3] + W    # noisy output signals

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()
```

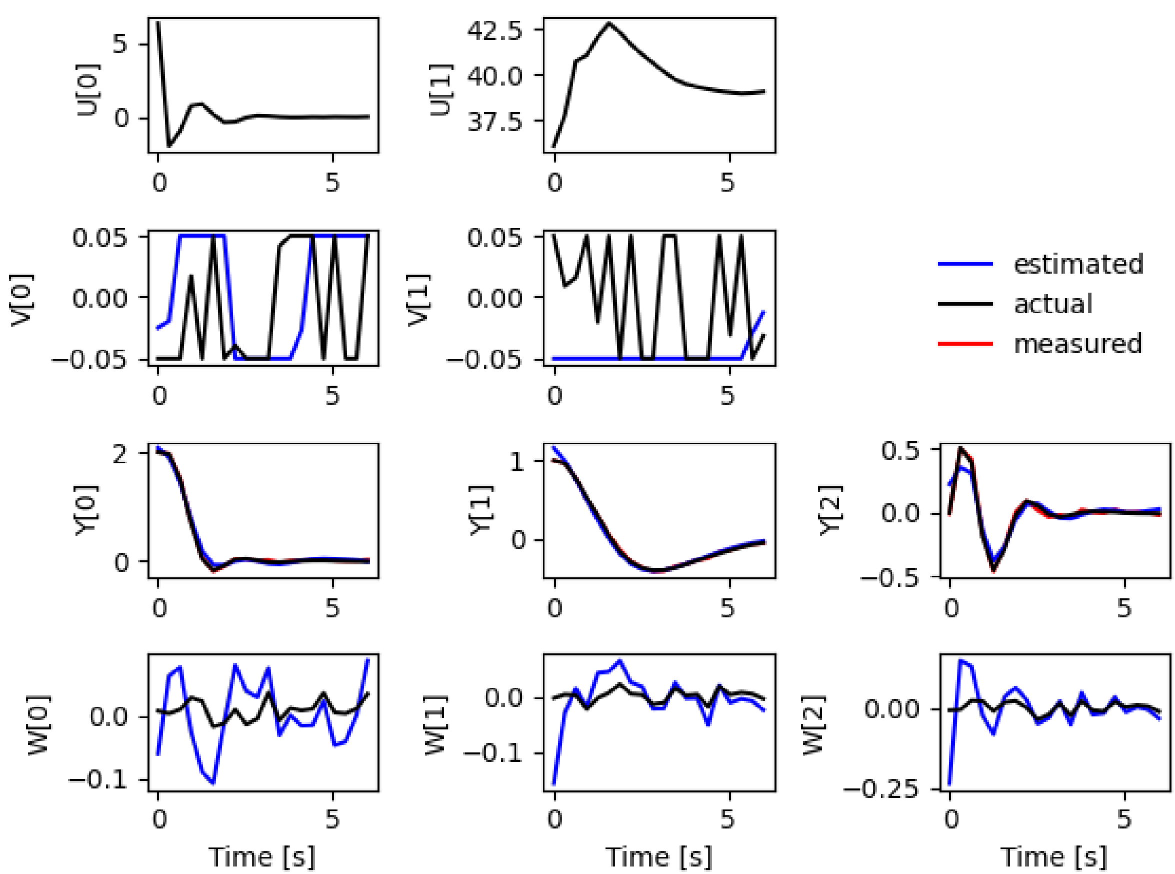
```
ekf_unclipped = ct.input_output_response(
    ekf, timepts, [clipped_resp.states, clipped_resp.outputs[6:8]],
    X0=[xe, P0.reshape(-1)])

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

```
Summary statistics:
* Cost function calls: 3411
* Constraint calls: 3594
* Final cost: 531.4809029482428
```





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.

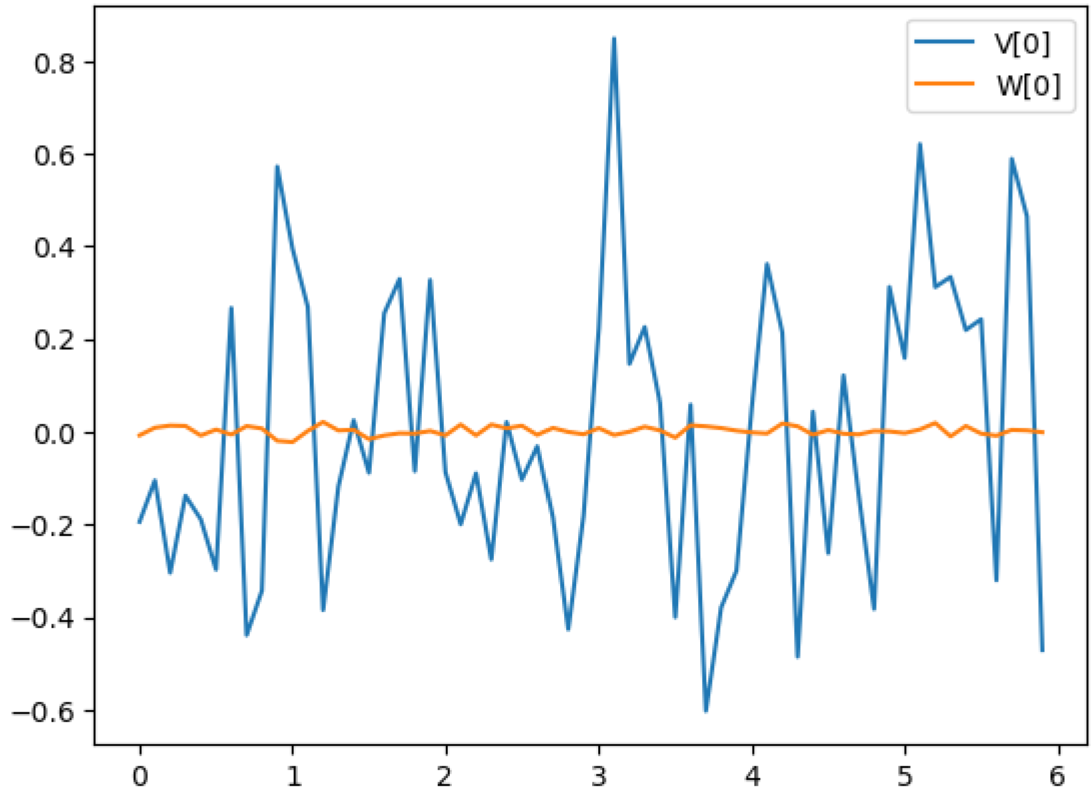
In [20]:
```
# Use a shorter horizon
mhe_timepts = timepts[0:5]
oep = opt_.OptimalEstimationProblem(
        sys, mhe_timepts, traj_cost, terminal_cost=init_cost)

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 0x7fb8a2b37be0>
Output: <function <lambda> at 0x7fb8a29cf400>
```

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();
```

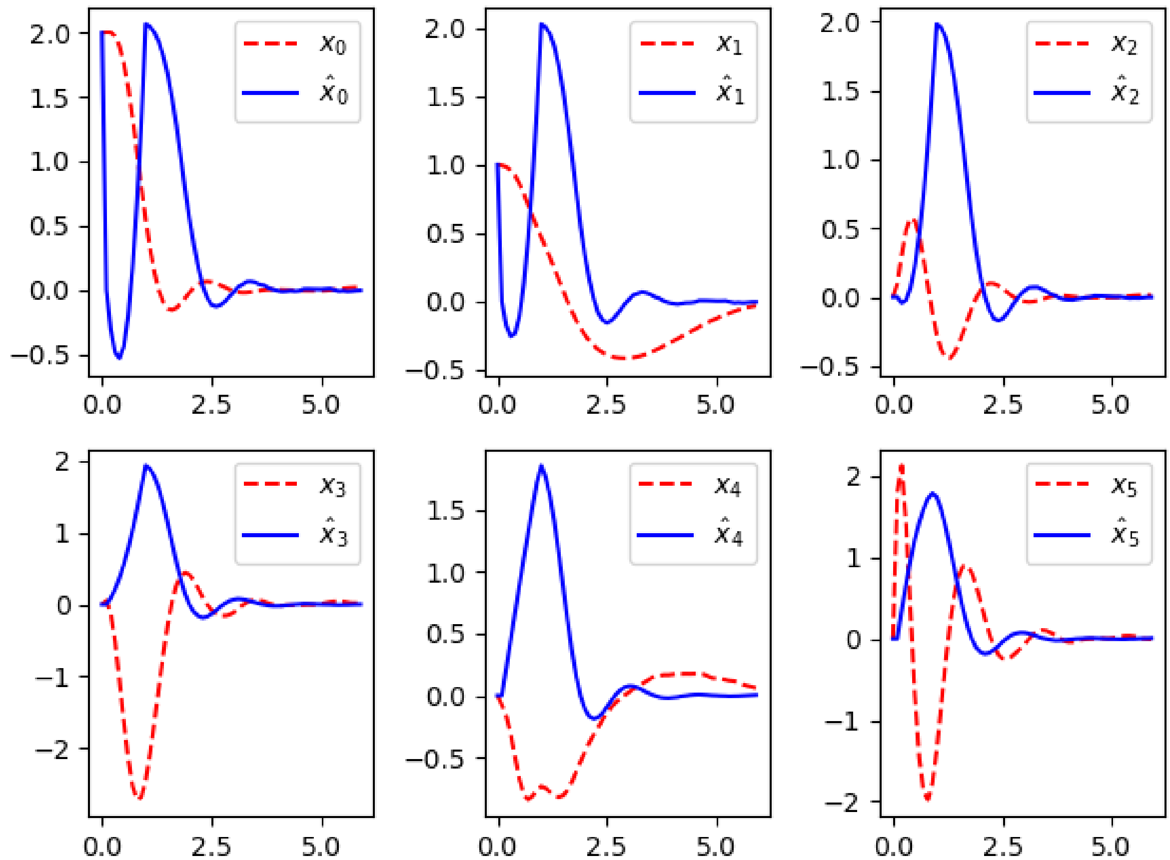


In [23]:
```
# Generate a new trajectory over the longer time vector
uvec = [xd, ud, V, W*0]
lqr_resp = ct.input_output_response(lqr_clsys, timepts, uvec, x0)
U = lqr_resp.outputs[6:8]                      # controller input signals
Y = lqr_resp.outputs[0:3] + W                  # noisy output signals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

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]
[ 8.91218488e-01 -4.46492004e-01  2.44864614e-03  1.44026750e+00
 -3.77747933e+00  1.90026859e+00]
[ 1.62618800e+00  1.44072913e-03  2.34179523e-01  2.44150815e+00
 -2.37111921e+00  3.11846269e+00]
[ 2.18699557  0.41377268  0.5736986   2.81869517 -1.22781759  2.82948617]
[ 2.54259035  0.74801009  0.84692413  2.58213179 -0.49292893  1.82725891]
[ 2.65887281  0.96606349  0.9461033   1.87403    -0.16783022  0.57643081]
[ 2.52746851  1.05474285  0.86005203  0.78097101 -0.12479611 -0.62681553]
[ 2.13425546  0.99333097  0.60740436 -0.59590214 -0.26221051 -1.61756872]
[ 1.54018515  0.8051531   0.22935257 -1.94091159 -0.48718507 -2.27562032]
[ 0.80771838  0.53248179 -0.11780412 -3.11133521 -0.86024232 -2.32821066]
[ 0.45239883  0.43919411 -0.41553108 -3.00854607 -0.85007201 -2.17193525]
[ 0.15432408  0.36086762 -0.59648259 -2.67758997 -0.88164936 -1.65012024]
[-0.0520054   0.29847059 -0.6613073  -2.14287244 -0.8946458  -0.92692034]
[-0.16922309  0.24508841 -0.61887957 -1.50621544 -0.86840703 -0.15944735]
[-0.23341362  0.17139566 -0.5118525  -0.89361253 -0.84984085  0.49491936]
[-0.2326812   0.09015474 -0.38554077 -0.28474362 -0.82745198  0.96097351]
[-0.20384993  0.0034451  -0.23805774  0.1792742  -0.78926584  1.25335665]
[-0.14233177 -0.07626162 -0.11929546  0.54912821 -0.74007929  1.30952487]
[-0.06665466 -0.16082929 -0.01601888  0.7796366  -0.68684394  1.21907355]
[-0.00236777 -0.21977868  0.06778942  0.83525927 -0.59399795  1.01513528]
[ 0.05449506 -0.27240244  0.12029687  0.79454542 -0.50587006  0.74314954]
[ 0.09229953 -0.31028036  0.1472144   0.66009926 -0.41055612  0.44458741]
[ 0.12721988 -0.3297885   0.15632381  0.4947291  -0.30551587  0.16916913]
[ 0.12939363 -0.35702331  0.14574812  0.28628118 -0.24007474 -0.05657709]
[ 0.12532434 -0.3793574   0.11389987  0.1079153  -0.1821153  -0.23758831]
[ 0.10435471 -0.38961706  0.06620042 -0.02844943 -0.11712421 -0.35260291]
[ 0.07907162 -0.40003404  0.01411352 -0.11519577 -0.05992168 -0.41006176]
[ 0.03920662 -0.41420523 -0.01849626 -0.20818104 -0.03491653 -0.39194604]
[ 0.01348724 -0.4125022  -0.04473493 -0.22430977  0.01337723 -0.3298449 ]
[-0.0132768  -0.41448912 -0.0551766  -0.23219593  0.04119142 -0.24472168]
[-0.03475741 -0.40139607 -0.06162631 -0.20224483  0.08552065 -0.15541926]
[-0.03716905 -0.388405   -0.05505894 -0.13897464  0.12543832 -0.05244972]
[-0.04439949 -0.39002869 -0.05057047 -0.0779106   0.12894578  0.0365504 ]
[-0.03877083 -0.38085834 -0.02757787 -0.02817371  0.14412222  0.12532881]
[-0.02458786 -0.36277568 -0.01895123  0.04092016  0.17058974  0.1475997 ]
[-0.01283634 -0.36300115 -0.00665831  0.07521987  0.15709132  0.14231452]
[-0.00898571 -0.35654858  0.00706922  0.0786478   0.14968832  0.12446028]
[ 0.00803002 -0.34652439  0.02003345  0.08283999  0.14793334  0.09388459]
[ 0.02182348 -0.3326117   0.02237674  0.08952947  0.1478255   0.05173357]
[ 0.02532379 -0.30918345  0.0207878   0.06792052  0.16864147  0.00495658]
[ 0.02448798 -0.28999894  0.01524361  0.04479671  0.17845578 -0.03194025]
[ 0.01144241 -0.27582726  0.012192   -0.01052176  0.1704494  -0.04789619]
[ 0.00210109 -0.26496825  0.00477219 -0.02996578  0.16192917 -0.04157093]
[ 0.00489503 -0.24621765 -0.0045386  -0.0178769   0.16896846 -0.0345983 ]
[ 0.00478005 -0.23092907 -0.00918808 -0.00788404  0.1598157  -0.02673815]
[-0.00796752 -0.20938681 -0.00950462 -0.02721378  0.16240967 -0.01931128]
[-0.00631268 -0.19090571 -0.00439085 -0.01844253  0.15818826 -0.00156478]
[-0.00924705 -0.17042132 -0.004662   -0.01678084  0.15895832 -0.00075372]
[-0.01125698 -0.14534765 -0.00452348 -0.01305335  0.16815086  0.00175471]
[-0.0119307  -0.13338412  0.00241101 -0.02393463  0.15609784  0.00774461]
[-0.0107125  -0.1208541   0.00662321 -0.02049746  0.14797311  0.02120043]
[-0.01340002 -0.11056464  0.00544058 -0.02098258  0.13061568  0.02586783]
[-0.00562278 -0.10496156  0.0023501   0.00473727  0.10765777  0.02976062]
```

```
[ 0.00857615 -0.095623   -0.00166629  0.03943354  0.09435684  0.0279741 ]
[ 0.00454976 -0.07819601 -0.00090422  0.03013675  0.09770189  0.02599464]
[ 0.01386053 -0.07311407  0.00520879  0.03294224  0.0818857   0.02603849]
[ 0.01516931 -0.06180277  0.00127049  0.03603364  0.07844304  0.00557587]
[ 0.01196372 -0.04400111  0.00404921  0.01430878  0.09096269 -0.01102636]
[ 0.01550558 -0.03669265  0.01760361 -0.01161542  0.08163193  0.00309693]
[ 0.0192345  -0.0333603   0.01038372 -0.00272278  0.06806309 -0.00699011]
```

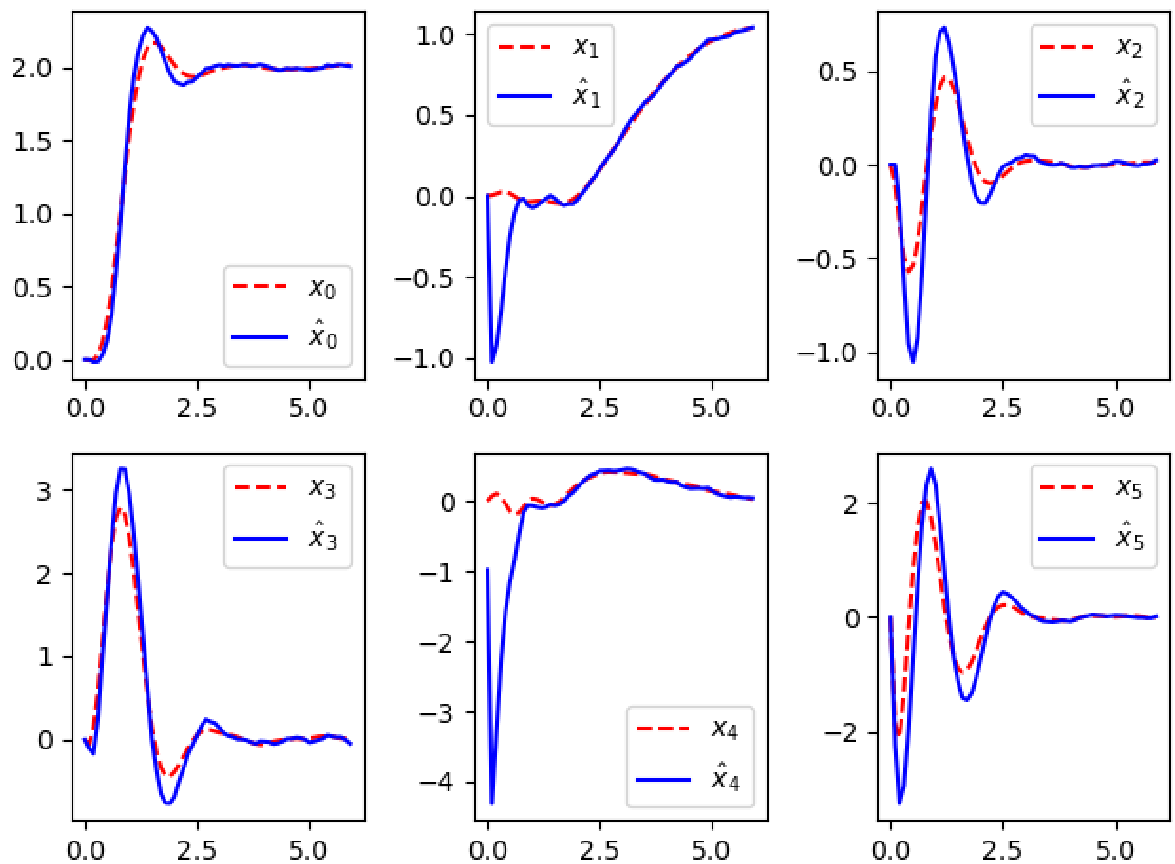


In [25]:
```
# Resimulate starting at the origin and moving to the "initial" condition
uvec = [x0, ue, V, W*0]
lqr_resp = ct.input_output_response(lqr_clsys, timepts, uvec, xe)
U = lqr_resp.outputs[6:8]                         # controller input signals
Y = lqr_resp.outputs[0:3] + W                     # 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]
[-0.00410692 -0.58834373  0.00342336 -0.11652757 -3.34070401 -2.2232086 ]
[-0.00705859 -0.48231418 -0.27953848 -0.06880408 -2.28786389 -3.36751924]
[ 1.00797549e-03 -3.22013438e-01 -6.52155773e-01  3.20167933e-01
 -1.39569935e+00 -3.03010781e+00]
[ 0.04386519 -0.1665474  -0.86758213  0.87057611 -0.81632955 -1.83671228]
[ 0.15030163 -0.05726421 -0.88305916  1.52488037 -0.51129949 -0.37887425]
[ 0.36377649  0.00396325 -0.69429052  2.24395025 -0.31136598  1.01510489]
[ 6.58561881e-01  5.32732639e-04 -3.63873262e-01  2.85628305e+00
 -1.85222422e-01  2.11685850e+00]
[ 1.01199111e+00 -4.34129129e-02  9.51712508e-04  3.25216567e+00
 -1.52736806e-01  2.73604783e+00]
[ 1.35425448 -0.06392709  0.306932    3.28229689 -0.09325346  2.71105447]
[ 1.63903007 -0.05841551  0.49325668  2.99599372 -0.02344227  2.14399624]
[ 1.86666426 -0.03756496  0.59462446  2.48025816  0.02234486  1.36680632]
[ 2.03654914 -0.02049607  0.59687861  1.85141404  0.02963976  0.49821639]
[ 2.15561164e+00 -1.25019576e-03  5.44345926e-01  1.19449556e+00
  3.18498663e-02 -2.65745247e-01]
[ 2.19956114 -0.01345627  0.4629515   0.50922346 -0.03361016 -0.82435913]
[ 2.20488594 -0.03619852  0.32297604 -0.02079109 -0.07117375 -1.25547859]
[ 2.16624734 -0.05197038  0.19196433 -0.44701575 -0.06531252 -1.43052987]
[ 2.10795504 -0.04977949  0.05774855 -0.70493848  0.00453164 -1.4303403 ]
[ 2.04387604 -0.05578026 -0.04437393 -0.8215791   0.05943296 -1.24365936]
[ 1.98869052 -0.03546636 -0.10840095 -0.80797339  0.14757395 -0.94634158]
[ 1.95748972 -0.00999854 -0.14438259 -0.65987231  0.23242536 -0.6085139 ]
[ 1.93113853  0.01844675 -0.13821951 -0.49769104  0.29146564 -0.24045057]
[ 1.93153191  0.05636002 -0.11432034 -0.29875067  0.34456617  0.05334777]
[ 1.92223885  0.08668959 -0.08081911 -0.15346519  0.36825518  0.27214633]
[ 1.93565358  0.12744116 -0.04889733 -0.00455878  0.40941608  0.38691138]
[ 1.95059209  0.1671323  -0.03591455  0.12678882  0.41720079  0.39097911]
[ 1.97118506  0.20402967 -0.0213131   0.22103498  0.42523844  0.35787601]
[1.97656958e+00 2.40018996e-01 1.83968966e-03 2.18626472e-01
 4.20248387e-01 3.02575497e-01]
[1.98930298 0.28490959 0.01382159 0.21281493 0.42899485 0.22236803]
[2.00084051 0.33096827 0.03498268 0.17718986 0.44383647 0.16285176]
[1.99948455 0.38533936 0.03771441 0.11554868 0.46189412 0.08253716]
[2.01126146 0.42971838 0.04121273 0.08693223 0.45677466 0.03723849]
[ 2.00827372  0.46254688  0.02199246  0.05204281  0.43660167 -0.03112207]
[ 2.01041949  0.49994573  0.01509037  0.01928664  0.41431621 -0.06099955]
[ 2.01672328  0.54046822  0.00751707  0.01255453  0.398378   -0.06916301]
[ 2.01991039e+00  5.64372124e-01  5.55501685e-03 -1.34532287e-04
  3.56240055e-01 -7.26076820e-02]
[ 2.01314143  0.59253884  0.00606838 -0.02490336  0.32047879 -0.06534159]
[ 2.01824422  0.62931567  0.0119287  -0.03681717  0.31781239 -0.04830818]
[ 2.02458886  0.66986659  0.00404975 -0.01550287  0.32215577 -0.05645452]
[ 2.0179569   0.70180436 -0.00569402 -0.02894419  0.29848417 -0.07536145]
[ 2.00947007  0.7336265  -0.00851031 -0.0379642   0.28780945 -0.06013545]
[ 1.99869382  0.77031725 -0.00538358 -0.05915753  0.29511753 -0.03605432]
[ 1.98889772  0.78441652 -0.01145597 -0.05471192  0.24550831 -0.01396139]
[ 1.98857441  0.80544862 -0.0141571  -0.03850346  0.21889196  0.01304708]
[ 1.99534403e+00  8.29405531e-01 -9.46860018e-03  1.12962849e-05
  1.99752958e-01  4.15408371e-02]
[ 1.99017032  0.86066839 -0.0078698   0.00351718  0.20224174  0.03644255]
[ 1.99372985  0.88964356 -0.00516315  0.01942194  0.20496185  0.03128193]
[ 1.99086276  0.915102   -0.00465084  0.01556757  0.19952224  0.01862385]
```

```
[1.98806296 0.9445064  0.00271356 0.00582972 0.2058909  0.03261041]
[ 1.9858724   0.9505617   0.01409901 -0.02164037  0.16069379  0.04061198]
[ 1.99174435  0.95649815  0.013308   -0.00713727  0.12281904  0.02950701]
[1.99456645 0.96763049 0.00796569 0.00236773 0.10142319 0.01359905]
[1.99829072e+00 9.75239105e-01 1.55580043e-03 1.22915865e-02
 8.14894351e-02 7.62631481e-03]
[ 2.01377235  0.98562909 -0.00326251  0.04885168  0.06972769  0.00400542]
[ 2.00777338  1.00381453 -0.00489441  0.03236474  0.07674607 -0.00239903]
[2.01514781e+00 1.01114216e+00 1.55843599e-03 2.90289836e-02
 7.16515908e-02 2.98308991e-03]
[2.01684869e+00 1.01891558e+00 1.95826122e-03 3.22497549e-02
 6.00804196e-02 1.52352306e-03]
[ 2.01305022  1.02989166  0.00790382  0.00670491  0.05801975 -0.00309479]
[ 2.01330667  1.03596977  0.02161675 -0.0267718   0.0527487   0.01605633]
[ 2.01606977  1.03354605  0.01057366 -0.01561493  0.02864041 -0.00368643]
```



In [ ]: