1 (a)

$$\dot{x} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u \quad \left(= A_c x + b_c u \right)$$

(b)

$T = 0,5$

$$A = e^{A_c T} = \mathbb{I} + A_c T + A_c \frac{T}{2} + \dots$$

$$= \mathbb{I} + A_c T$$

$$A = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \qquad b = \int_0^T e^{A_c \tau} d\tau \; b_c = \begin{bmatrix} T & \frac{1}{2}T^2 \\ 0 & T \end{bmatrix} \cdot \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix}$$

$$b = \begin{bmatrix} \frac{1}{2m} T^2 \\ \frac{T}{m} \end{bmatrix} = \begin{bmatrix} 0,125 \\ 0,5 \end{bmatrix}$$

Look mom, no MATLAB!

(c)

$$J(z) = \frac{1}{2} \sum_{t=0}^{N-1} \left(x_{t+1}^T Q x_{t+1} + u_t^T R u_t \right), \quad Q = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}, \; R = 2$$

The Riccati equation is:

$$P_t = Q + A^T P_{t+1} \left(\mathbb{I} + B R^{-1} B^T P_{t+1} \right)^{-1} A, \quad t = 0, \dots, N-1,$$

$$P_N = Q$$

By solving the system for all P_t we get a sequence of optimal controller gains:

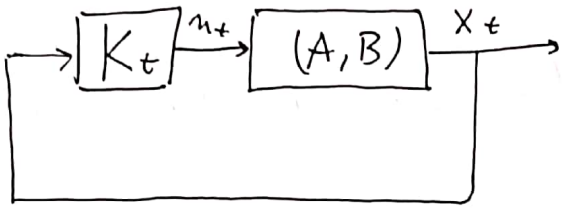
$$K_t = R^{-1} B^T P_{t+1} \left(\mathbb{I} + B R^{-1} B^T P_{t+1} \right)^{-1} A$$

$$u_t = -K_t x_t$$

```
A = [1 0.5; 0 1];
B = [0.125; 0.5];
Q = eye(2);
R = 1;
[K,S,e] = dlqr(A,B,Q,R)
```

```
K =

    0.6514    1.3142


S =

    4.0350    2.0616
    2.0616    4.1438


e =

   0.6307 + 0.1628i
   0.6307 - 0.1628i
```

Published with MATLAB® R2018b



(d) $K = [0,6514 \quad 1,3142]$, $P = \begin{bmatrix} 4,0350 & 2,0616 \\ 2,0616 & 4,1438 \end{bmatrix}$

$\lambda = 0,6307 \pm 0,1628i$

Since the eigenvalues are inside the unit circle, the system is stable.

(e) The LQR is stable as long as (A,B) is stabilizable and (A,D) is detectable, where $Q = D^T D$.

2 $x_{t+1} = 3x_t + 2u_t$

$f(z) = \frac{1}{2}\sum_{t=0}^{\infty}(qx_{t+1}^2 + u_t^2)$

(a) Scalar Riccati:

$p_t = q + a p_{t+1}(1 + b^2 p_{t+1}\frac{1}{r})^{-1} a$

Infinite horizon: $p_t = p_{t+1}$

$\Rightarrow \underline{p = q + \frac{a^2 p}{1 + b^2 p/r} = q + \frac{a^2 p r}{r + b^2 p}}$ □

$a = 3, \; b = 2, \; q = 2, \; r = 1$

$\Rightarrow p = 2 + \frac{9p}{1+4p}$

$\Leftrightarrow p^2 - 4p - \frac{1}{2} = 0 \Rightarrow \underline{p = 2 \pm \frac{3}{2}\sqrt{2}}$

$$p = 2 + \frac{3}{2}\sqrt{2} \approx 4,1213$$

(b) $k = \frac{1}{r} bp(1 + \frac{1}{r} b^2 p)^{-1} a = \frac{6p}{1+4p} = \frac{4+3\sqrt{2}}{3+2\sqrt{2}}$

$k \approx 1,4142$

(c) The LQR is stable as long as (A, B) is stabilizable and (A, D) is detectable, where $Q = D^T D$.
Why did you ask this twice?
If you meant a general finite horizon LQ controller earlier, we do not know anything about the stability in general, as far as I know, like the MPC.

3 $x_{t+1} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0{,}1 & -0{,}79 & 1{,}78 \end{bmatrix} x_t + \begin{bmatrix} 1 \\ 0 \\ 0{,}1 \end{bmatrix} u_t, \quad y_t = [0\ 0\ 1]\, x_t,$

$x_0 = [0\ 0\ 1]^T,$

$f(z) = \sum_{t=0}^{N-1} (y_{t+1}^2 + r u_t^2), \quad N = 30,$

$-1 \le u_t \le 1, \quad t = 0, \dots, N-1$

(b) Let u_t be constant for each step of 5 normal timesteps.

The system is now expressed as:

$$3N \left\{ \left[\begin{array}{ccc|ccc} \text{II} & 0 & \cdots & -b & 0 & \cdots \\ -A & \text{II} & & -b & 0 & \\ 0 & -A & \ddots & -b & 0 & \\ \vdots & & 0 & -b & 0 & \\ 0 & \cdots & -A\ \text{II} & 0 & -b & -b \end{array}\right] \right. \begin{matrix} \\ \\ \\ \\ \\ \end{matrix} \quad z = \begin{bmatrix} Ax_0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$3N$ columns on the left block; $\frac{N}{5}$ columns on the right block.

Since we now have $\frac{N}{5}$ control variables, the R-terms of G is only $\frac{N}{5}$ long, not N. We have to update the A matrix in $Az \le b$ similarily.

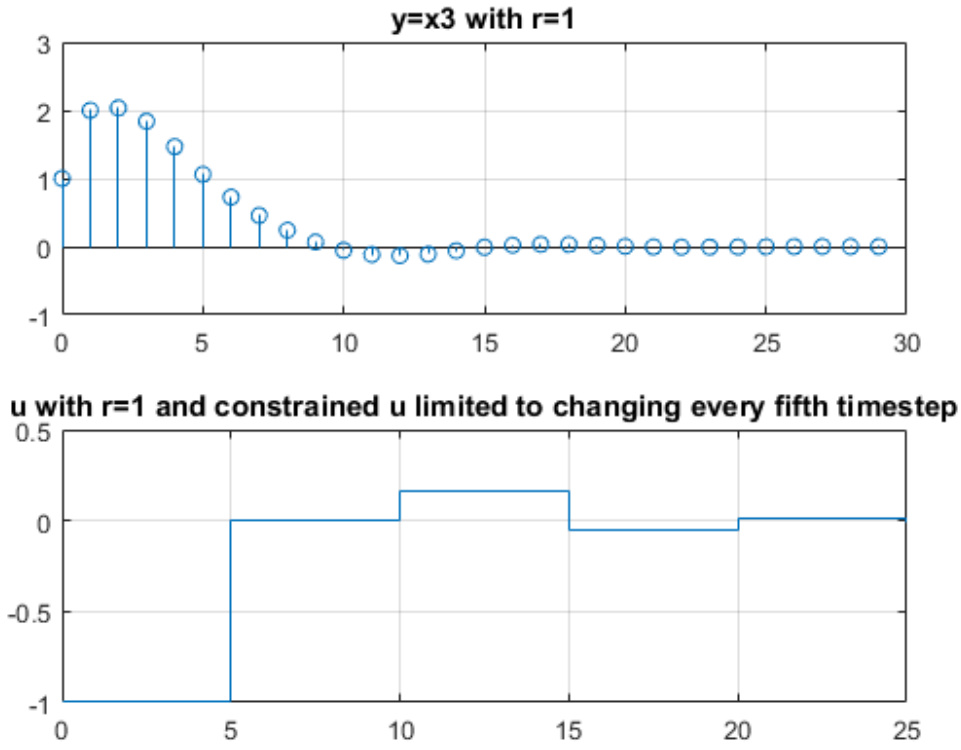
The resulting response is more oscillatory as the controller is slower (and r is low), but the system handles the reduction of variables quite nicely. Iterations = 4.

3b

```
N = 30;
t = (0:N-1);
c_step = 5;
t_u = (0:c_step:N-1);
Q = blkdiag(0,0,1);
R = 1;
ulb = -1;
uub = 1;
G = 0.5*[kron(eye(N), Q) zeros(N * 3, N/c_step); zeros(N/c_step, N *
 3)  R * eye(N/c_step)];
A_d = [0 0 0; 0 0 1; 0.1 -0.79 1.78];
b_d = [1 0 0.1]';
x_0 = [0 0 1]';
A_eq = [(eye(3 * N) + [zeros(3, 3 * N); kron(eye(N - 1), -A_d) zeros(3
 * (N - 1), 3)]) kron(eye(N/c_step), kron(ones(c_step,1), -b_d))];
b_eq = [A_d * x_0; zeros(3 * (N - 1), 1)];
A = [zeros(2 * N / c_step, 3 * N) kron(eye(N/c_step), [uub; ulb])];
b = ones(2 * N / c_step, 1);
[w, fval, flag, out] = quadprog(G, [], A, b, A_eq, b_eq);
disp(out.iterations)
x = w(1:3*N);
u = w(3*N+1:3*N + N/c_step);
x1 = [x_0(1)];
x2 = [x_0(2)];
x3 = [x_0(3)];

for i = 1:(N-1)
    x1 = [x1; x(3*i + 1)];
    x2 = [x2; x(3*i + 2)];
    x3 = [x3; x(3*i + 3)];
end

figure
subplot(211);
stem(t, x3);
grid on
title('y=x3 with r=1');
subplot(212);
stairs(t_u, u);
grid on
title('u with r=1 and constrained u limited to changing every fifth
 timestep');
```

```
Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-
decreasing in
feasible directions, to within the default value of the optimality
 tolerance,
and constraints are satisfied to within the default value of the
 constraint tolerance.
```

4



Published with MATLAB® R2017a

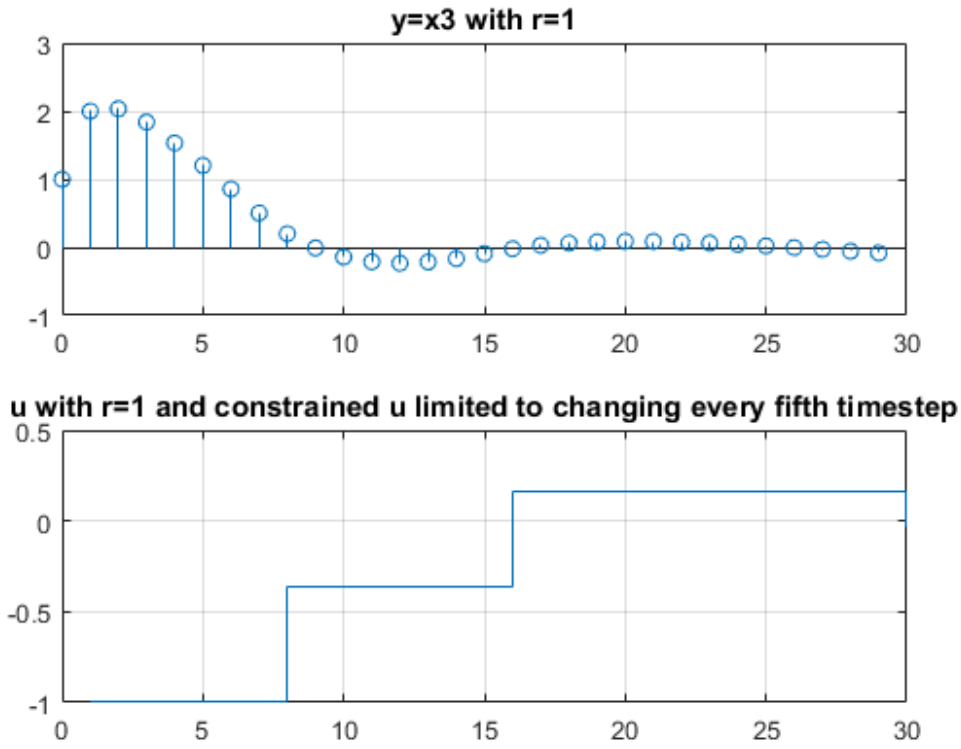
3c

```
N = 30;
t = (0:N-1);
c_steps = [1,1,2,4,8,14];
t_u = [1,2,4,8,16,30];
N_u = length(c_steps);
Q = blkdiag(0,0,1);
R = 1;
ulb = -1;
uub = 1;
G = 0.5*[kron(eye(N), Q) zeros(N * 3, N_u); zeros(N_u, N * 3)  R *
 eye(N_u)];
A_d = [0 0 0; 0 0 1; 0.1 -0.79 1.78];
b_d = [1 0 0.1]';
x_0 = [0 0 1]';
A_eq_l = [eye(3 * N) + [zeros(3, 3 * N); kron(eye(N - 1), -A_d)
 zeros(3 * (N - 1), 3)]];
A_eq_r = [];
for i = 1:N_u
    A_eq_r = [A_eq_r; zeros(c_steps(i) * 3, i-1),
 kron(ones(c_steps(i), 1), -b_d), zeros(c_steps(i) * 3, N_u - i)];
end
A_eq = [A_eq_l A_eq_r];
b_eq = [A_d * x_0; zeros(3 * (N - 1), 1)];
A = [zeros(2 * N_u, 3 * N) kron(eye(N_u), [uub; ulb])];
b = ones(2 * N_u, 1);
[w, fval, flag, out] = quadprog(G, [], A, b, A_eq, b_eq);
disp(out.iterations)
x = w(1:3*N);
u = w(3*N+1:3*N + N_u);
x1 = [x_0(1)];
x2 = [x_0(2)];
x3 = [x_0(3)];

for i = 1:(N-1)
    x1 = [x1; x(3*i + 1)];
    x2 = [x2; x(3*i + 2)];
    x3 = [x3; x(3*i + 3)];
end

figure
subplot(211);
stem(t, x3);
grid on
title('y=x3 with r=1');
subplot(212);
stairs(t_u, u);
grid on
title('u with r=1 and constrained u limited to changing every fifth
 timestep');
```

Minimum found that satisfies the constraints.

```
Optimization completed because the objective function is non-
decreasing in
feasible directions, to within the default value of the optimality
 tolerance,
and constraints are satisfied to within the default value of the
 constraint tolerance.
```

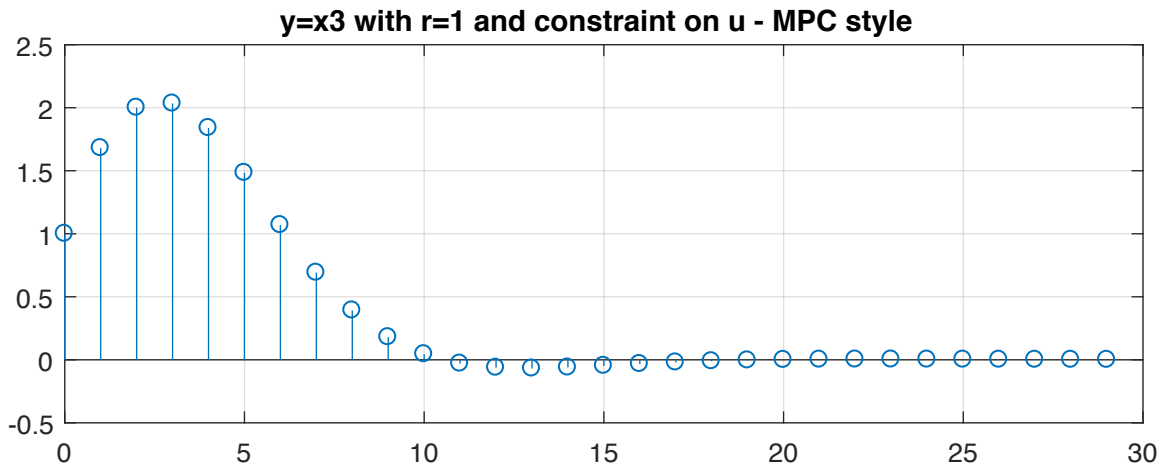
```
     4
```

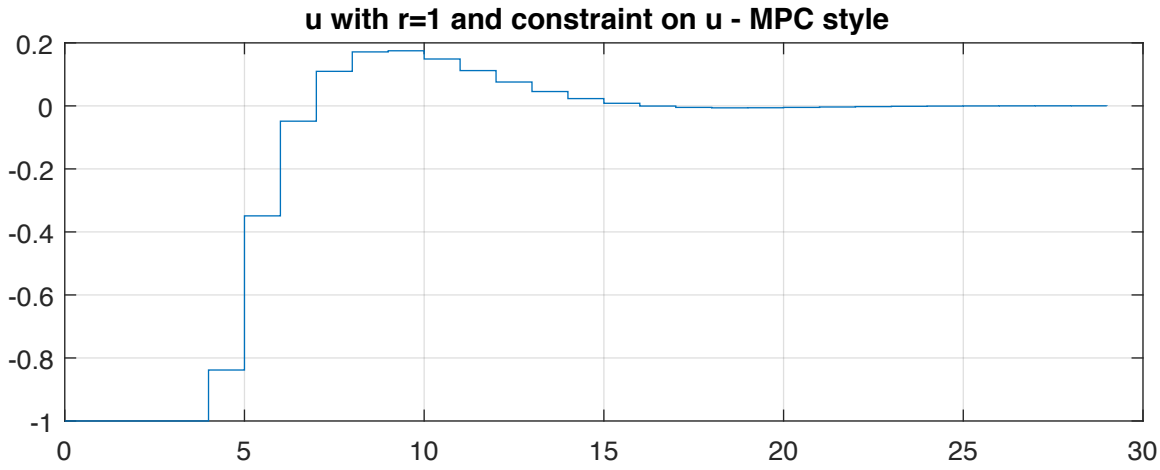


Published with MATLAB® R2017a

This configuration with r=1 was quite boring,
as we do not really see the increasing input blocking over time that much.
Therefore the resulting y-plot is not much different compared to fixed step input blocking in this configuration.
But considering most of the input movement usually happens in the beginning of the time horizon, an increasing input block makes a lot of sense, and will give much better performance in general.

3d


y=x3 with r=1 and constraint on u - MPC style
2.5
2
1.5
1
0.5
0
-0.5
0
5
10
15
20
25
30


u with r=1 and constraint on u - MPC style
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
5
10
15
20
25
30

3e

```
%%MPC with input blocking
N = 30;
M = 30;
c_steps = [1,1,2,4,8,14];
N_u = length(c_steps);
t = (0:N-1);
Q = blkdiag(0,0,1);
R = 1;
ulb = -1;
uub = 1;
G = 0.5*[kron(eye(N), Q) zeros(N * 3, N_u); zeros(N_u, N * 3)  R *
 eye(N_u)];
A_d = [0 0 0; 0 0 1; 0.1 -0.79 1.78];
b_d = [1 0 0.1]';
x_0 = [0 0 1]';

A_eq_l = [eye(3 * N) + [zeros(3, 3 * N); kron(eye(N - 1), -A_d)
 zeros(3 * (N - 1), 3)]];
A_eq_r = [];
for i = 1:N_u
    A_eq_r = [A_eq_r; zeros(c_steps(i) * 3, i-1),
 kron(ones(c_steps(i), 1), -b_d), zeros(c_steps(i) * 3, N_u - i)];
end
A_eq = [A_eq_l A_eq_r];
b_eq = [A_d * x_0; zeros(3 * (N - 1), 1)];

A = [zeros(2 * N_u, 3 * N) kron(eye(N_u), [uub; ulb])];
b = ones(2 * N_u, 1);

x1 = [x_0(1)];
x2 = [x_0(2)];
x3 = [x_0(3)];
u = [];
x_k = x_0;
u_k = [];

for i = 1:M
    b_eq = [A_d * x_k; zeros(3 * (N - 1), 1)];
    [z, fval, flag, out] = quadprog(G, [], A, b, A_eq, b_eq);
    x_k = [z(1); z(2); z(3)];
    x1 = [x1; x_k(1)];
    x2 = [x2; x_k(2)];
    x3 = [x3; x_k(3)];
    u = [u; z(3*N + 1)];
end

x3 = x3(1:M);

figure
subplot(211);
stem(t, x3);
grid on
```

```
title('y=x3 with r=1');
subplot(212);
stairs(t, u);
grid on
title('u with r=1 and constraint on u with increasing input blocking -
 MPC style');
```

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in feasible directions, to within the default value of the optimality tolerance, and constraints are satisfied to within the default value of the constraint tolerance.

Minimum found that satisfies the constraints.

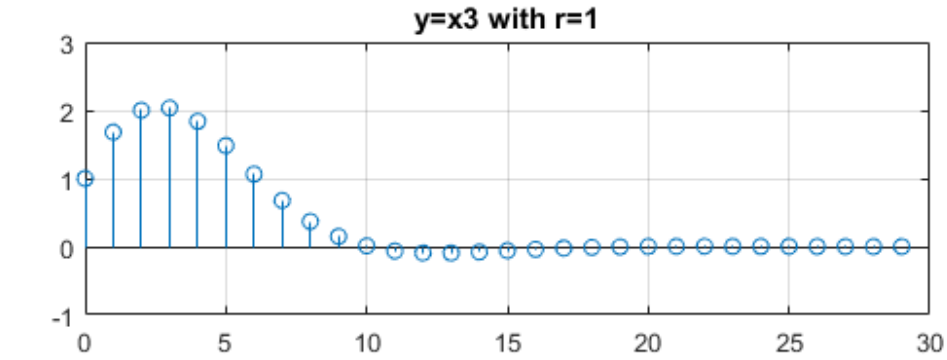
Optimization completed because the objective function is non-decreasing in feasible directions, to within the default value of the optimality tolerance, and constraints are satisfied to within the default value of the constraint tolerance.

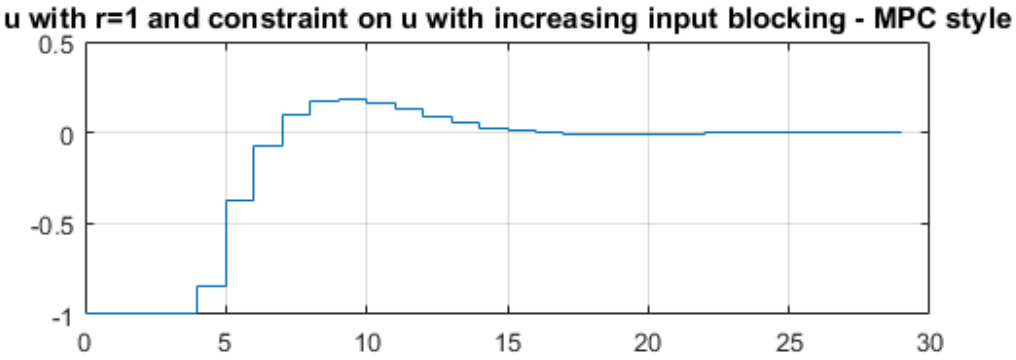
Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in feasible directions, to within the default value of the optimality tolerance, and constraints are satisfied to within the default value of the constraint tolerance.

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in feasible directions, to within the default value of the optimality tolerance, and constraints are satisfied to within the default value of the constraint tolerance.





Published with MATLAB® R2017a

There is not much difference between input blocking with increasing steps and the optimal control sequence where the step size is always 1. This is good news! It means we were able to reduce the number of control variables by a factor of 5 without much difference in the performance.

3f: The effect of input blocking in MPC obviously does not change the fact that we evaluate the optimization problem at each time step, so the input still changes at each iteration. But it reduces the problem size as we have fewer variables, and thus it makes the controller faster. This means that we can increase the sampling frequency, which will probably have a much greater impact on the performance of the controller compared to the slight step away from optimality. Another option is to increase the time horizon.
The main problem with MPC in my eyes is the enormous optimization problems that have to be solved in real time, so input blocking is a clever way to reduce the complexity of the problems.

We choose blocks of increasing length as most of the movement in u happens in the beginning of the time horizon, so we want good resolution in the transient and then we can sacrifice the resolution in the steady state phase where not a lot is happening.

Idea: would it be possible to use even more optimization to find the optimal number of control variables and the optimal steps between them given some threshold in reduced performance that we can allow?

Idea: would it be possible to only evaluate the optimization problems if the measured x has deviated some delta from the calculated trajectory? If the model is good, this could also save a lot of processing power, but it wouldnt be consistent so it probably would not be useful...