In [1]:

```
import pandas as pd
import numpy as np
from numpy.linalg import inv
import math as m
from math import sqrt
import sympy as sp
from sympy import collect, simplify, expand, fraction, latex
from IPython.display import display, Markdown, Math
import control as co
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import colors as mcolors
sp.init_printing(use_latex='mathjax')
plt.rcParams['figure.figsize'] = [20, 10]
```

In [2]:

```
class numden_coeff:
    def __init__(self, expr, symb):
        self.num, self.denum = fraction(expr)
        self.symb = symb
        self.common_factor = None
        self.lst_denum_coeff = self.build_lst(self.denum)
        self.lst_num_coeff = self.build_lst(self.num)

    def build_lst(self, poly):
        order = sp.Poly(poly, self.symb).degree()
        lst = [expand(poly).coeff(self.symb**i) for i in range((order), 0, -1)]
        lst.append(poly.subs(self.symb,0))
        if (self.common_factor == None):
            self.common_factor = lst[0]

        lst = [simplify(lst[i]/self.common_factor) for i in range(order + 1)]
        return lst

    def disp(self):
        display(Markdown(r"Numerator coefficients (\beta)"), self.lst_num_coeff)
        display(Markdown(r"Denominator coefficients (alpha)"), self.lst_denum_coeff)
```

In [3]:

```
def theta_hat_ploter(df, theta0, title, line_width=1.2):
    lst_color = ['b', 'y', 'g', 'r']
    lst_labels = df.columns
    graph = sns.lineplot(data=df, dashes=False)

    for i in range(len(theta0)):
        graph.axhline(y=theta0[i], color=lst_color[i], linestyle='--', linewidth=line_width, label=lst_labels

    plt.title(title, fontsize=20)
    plt.ylabel('Magnitude of "Theta_hat"', fontsize=18)
    plt.xlabel('Time Stamps "t"', fontsize=18)
    plt.legend(bbox_to_anchor=(1.05, 1),
               loc=2,
               borderaxespad=0,
               labels=lst_labels,
               fontsize='xx-large')
    plt.show()
```

Problem 1

Part 1

In [38]:

```
c, d= sp.symbols('c d')
s, zeta, omega = sp.symbols('s zeta omega')
r1, s0, s1, a0, t0 = sp.symbols('r_1 s_0 s_1 a_0 t_0')

a = 1
b = 1
G1 = b/(s + a)
G2 = c/(s+d)
G = collect(expand(G1*G2), s)
B, A = fraction(G)
B_minus = B
```

$$G_1(s)G_2(s) = G(s) = \frac{c}{d + s^2 + s(d+1)}$$

Therefore

$$B = c$$

$$A = d + s^2 + s(d+1)$$

Since $Deg(B)$ is clearly 0, $B^+ = 1$ and $B^- = c$

In [39]:

```
Am = s**2 + 2*s + 1
Bm = 1
Gm = Bm/Am
Bm_prime = Bm/B_minus
```

A_m is given to be $s^2 + 2s + 1$. Letting the desired model take the form of

$$G_m = \frac{\omega^2}{s^2+2\zeta\omega+\omega^2}$$

ω and ζ are equivalent to 1. Since $\omega = 1$, B_m must be equal to 1 which yeilds

$$G_m = \frac{1}{s^2 + 2s + 1}$$

Additionally, $B_m^{'} = \frac{B_m}{B^-} = \frac{1}{c}$

In order for minimum phase to be achived, the following conditions on the degrees of the polynomials making up the system must be met and will ultimatley guide the desing,

$$Deg(A_0) = Deg(A) - Deg(B^+) - 1 = 2 - 0 - 1 = 1$$

$$Deg(A_c) = 2(Deg(A)) - 1 = 2 * 2 - 1 = 3$$

$$Deg(R) = Deg(S) = Deg(A_c) - Deg(A) = 3 - 2 = 1$$

In [40]:

```
A0 = s + a0
R_prime = s + r1
R = R_prime
S = s0*s + s1
T_ = A0*Bm_prime
```

Since $Deg(B^+) = 0$ then $Deg(R^{'}) = 1$ and therfore

$$R = B^+R^{'} = R^{'} = r_1 + s$$

Additionally, considering the polynomial degrees derived above, we know that

$$A_0 = a_0 + s$$

$$S = ss_0 + s_1$$

Lastly,

$$T = A_0B_m^{'} = \frac{a_0 + s}{c}$$

In [41]:

```
# derivation of diophantine equation
LHS = collect(expand(A*R_prime + B_minus*S), s)
RHS = collect(expand(A0*Am), s)
equ = sp.Eq(LHS,RHS)

# Derivation of control paramteters
r_1 = sp.solve(sp.Eq(LHS.coeff(s**2),RHS.coeff(s**2)), r1)[0]
s_0 = sp.solve(sp.Eq(LHS.coeff(s**1),RHS.coeff(s**1)), s0)[0]
s_1 = sp.solve(sp.Eq(LHS.subs(s,0),RHS.subs(s,0)), s1)[0]
```

The Diophantine equation in terms of control parameters is given by

$$AR^{'} + B^-S = A_0A_m$$

$$\Rightarrow\ cs_1 + dr_1 + s^3 + s^2\left(d + r_1 + 1\right) + s\left(cs_0 + dr_1 + d + r_1\right) = a_0 + s^3 + s^2\left(a_0 + 2\right) + s\left(2a_0 + 1\right)$$

Grouping the coefficients of the same ordered s terms and solving for the control parameters yeilds

$$r_1 = a_0 - d + 1$$

$$s_0 = \frac{2a_0 - dr_1 - d - r_1 + 1}{c}$$

$$s_1 = \frac{a_0 - dr_1}{c}$$

Part 2

ODE of Plant

In [42]:

```
y_t, u_t, p = sp.symbols('y(t) u(t) p')

ode_RHS = ((-A.coeff(s**1)*p - A.subs(s,0))*y_t) + (B.coeff(s**2)*p**2 + B.coeff(s**1)*p**1 + B.subs(s,0))*u_t
ode_RHS
```

Out[42]: $cu(t) + y(t)(-d + p(-d-1))$

The ODE of 2^{nd} order describing the process is given by

$$p^2y(t) = -(d+1)py(t) - dy(t) + cu(t)$$

where p is the time shifting operator. The reliance of the RHS of the equation on derivatives can be changed to integrals by filtering the input $u(t)$ and output $y(t)$ of the plant by a filter whose denominator polynomial is of a greater order than the highest derivative term. The above equation becomes

$$p^2y_f(t) = -(d+1)py_f(t) - dy_f(t) + cu_f(t)$$

$$\Rightarrow p^2H_f(p)y(t) = -(d+1)pH_f(p)y(t) - dH_f(p)y(t) + cH_f(p)u(t)$$

$$\Rightarrow \frac{p^2}{A_m}y(t) = -(d+1)\frac{p}{A_m}y(t) - d\frac{1}{A_m}y(t) + c\frac{1}{A_m}u(t)$$

For simplicity, let $(d+1) = x$. The ODE then becomes

$$\Rightarrow \frac{p^2}{A_m}y(t) = -x\frac{p}{A_m}y(t) - d\frac{1}{A_m}y(t) + c\frac{1}{A_m}u(t)$$

This equation can be further simplified as

$$\Rightarrow y_{f2}(t) = -xy_{f1}(t) - dy_{f0}(t) + cu_{f0}(t)$$

This equation will be used for the measurment model i.e.

$$y_{f2}(t) = \phi(t)^T\theta = [-y_{f1}(t)\ \ -y_{f0}(t)\ \ u_{f0}(t)][x\ d\ c]^T$$

Bilinear Transformation of Filtered ODE

In [43]:

```
# Filter
H_f = (1/Am)
H_f_p = (H_f).subs(s,p)
```

The filter $H_f(p)$ is given to be

$$H_f(p) = \frac{1}{A_m} = \frac{1}{p^2 + 2p + 1}$$

This filter ,and the ODE above, are however, in terms of p and are therfore, in continuous time domain. To converte the filter to discrete time (q), a bilinear transformation will be performed. i.e.

$$p \rightarrow \frac{2(1-\frac{1}{q})}{T(1+\frac{1}{q})}$$

The ODE can now be represented in the discret time domain by

$$y_i(kT) = H_i(q^{-1})y(kT) = \frac{p^i}{A_m(p)}\Bigg|_{p \rightarrow \frac{2(1-\frac{1}{q})}{T(1+\frac{1}{q})}} y(kT), \quad u_i(kT) = \frac{p^i}{A_m(p)}\Bigg|_{p \rightarrow \frac{2(1-\frac{1}{q})}{T(1+\frac{1}{q})}} u(kT)$$

In [44]:

```python
T, q = sp.symbols('T q')

bilinear_T = (2/T)*((1 - q**(-1))/(1 + q**(-1))) # what will be subsituted for s (kept actual equations in te


H_fy2 = collect(simplify(expand(((s**2)*H_f).subs(s,bilinear_T))), q)
H_fy1 = collect(simplify(expand((s*H_f).subs(s,bilinear_T))), q)
H_fy0 = collect(simplify(expand((H_f).subs(s,bilinear_T))), q)
H_fu0 = collect(simplify(expand((H_f).subs(s,bilinear_T))), q)

# Creation of numerator and denominator coefficient extractor objects (numden_coeff() class defined at the to
obj_H_fy2 = numden_coeff(H_fy2, q)
obj_H_fy1 = numden_coeff(H_fy1, q)
obj_H_fy0 = numden_coeff(H_fy0, q)
obj_H_fu0 = numden_coeff(H_fu0, q)


aH_fy2 = obj_H_fy2.lst_denum_coeff
bH_fy2 = obj_H_fy2.lst_num_coeff

aH_fy1 = obj_H_fy1.lst_denum_coeff
bH_fy1 = obj_H_fy1.lst_num_coeff

aH_fy0 = obj_H_fy0.lst_denum_coeff
bH_fy0 = obj_H_fy0.lst_num_coeff

aH_fu0 = obj_H_fu0.lst_denum_coeff
bH_fu0 = obj_H_fu0.lst_num_coeff
```

For $y_{f1} = H_f(q^{-1})y(kT)$, the coefficients of the denominator αy_{f1} are

$$\alpha y_{f1} = \left[1, \ \frac{2(T-2)}{T+2}, \ \frac{T^2-4T+4}{T^2+4T+4}\right]$$

(ordered by powers of q going from q^0 to q^-2) and the coefficients of the numerator βy_{f1} are

$$\beta y_{f1} = \left[\frac{2T}{T^2+4T+4}, \ 0, \ -\frac{2T}{T^2+4T+4}\right]$$

which are also ordered by powers of q going from q^0 to q^-2. Similarly, the coefficients for the denominator (α) and numerator (β) of y_{f0} and u_{f0} are

$$\alpha y_{f0} = \left[1, \ \frac{2(T-2)}{T+2}, \ \frac{T^2-4T+4}{T^2+4T+4}\right]$$

$$\beta y_{f0} = \left[\frac{T^2}{T^2+4T+4}, \ \frac{2T^2}{T^2+4T+4}, \ \frac{T^2}{T^2+4T+4}\right]$$

$$\alpha u_{f0} = \left[1, \ \frac{2(T-2)}{T+2}, \ \frac{T^2-4T+4}{T^2+4T+4}\right]$$

$$\beta u_{f0} = \left[\frac{T^2}{T^2+4T+4}, \ \frac{2T^2}{T^2+4T+4}, \ \frac{T^2}{T^2+4T+4}\right]$$

Note that $\alpha y_{f0} = \alpha u_{f0}$ and $\beta y_{f0} = \beta u_{f0}$

$$\alpha y_{f2} = \left[1, \ \frac{2(T-2)}{T+2}, \ \frac{T^2-4T+4}{T^2+4T+4}\right]$$

$$\beta y_{f2} = \left[\frac{4}{T^2+4T+4}, \ -\frac{8}{T^2+4T+4}, \ \frac{4}{T^2+4T+4}\right]$$

In [45]:

```python
y_k, y_k_1, y_k_2  = sp.symbols('y(k) y(k-1) y(k-2)')
u_k, u_k_1, u_k_2  = sp.symbols('u(k) u(k-1) u(k-2)')
y1_k_1, y1_k_2 = sp.symbols('y_{1}(k-1) y_{1}(k-2)')
y0_k_1, y0_k_2 = sp.symbols('y_{0}(k-1) y_{0}(k-2)')
u0_k_1, u0_k_2 = sp.symbols('u_{0}(k-1) u_{0}(k-2)')

# Derivation of filtered signal equations
y1_k = -y1_k_1*aH_fy1[1] - y1_k_2*aH_fy1[2] + y_k*bH_fy1[0] + y_k_2*bH_fy1[2]
y0_k = -y0_k_1*aH_fy0[1] - y0_k_2*aH_fy0[2] + y_k*bH_fy0[0] + y_k_1*bH_fy0[1] + y_k_2*bH_fy0[2]
u0_k = -u0_k_1*aH_fu0[1] - u0_k_2*aH_fu0[2] + u_k*bH_fu0[0] + u_k_1*bH_fu0[1] + u_k_2*bH_fu0[2]
```

The equations for the filtered output and input y_{f1}, y_{f0} and u_{f0} equations are

$$y_{f1}(kT) = \frac{2Ty(k)}{T^2+4T+4} - \frac{2Ty(k-2)}{T^2+4T+4} - \frac{2y_1(k-1)(T-2)}{T+2} - \frac{y_1(k-2)\left(T^2-4T+4\right)}{T^2+4T+4}$$

$$y_{f0}(kT) = \frac{T^2y(k)}{T^2+4T+4} + \frac{2T^2y(k-1)}{T^2+4T+4} + \frac{T^2y(k-2)}{T^2+4T+4} - \frac{2y_0(k-1)(T-2)}{T+2} - \frac{y_0(k-2)\left(T^2-4T+4\right)}{T^2+4T+4}$$

$$u_{f0}(kT) = \frac{T^2u(k)}{T^2+4T+4} + \frac{2T^2u(k-1)}{T^2+4T+4} + \frac{T^2u(k-2)}{T^2+4T+4} - \frac{2u_0(k-1)(T-2)}{T+2} - \frac{u_0(k-2)\left(T^2-4T+4\right)}{T^2+4T+4}$$

In [46]:

```python
y2_k, y2_k_1, y2_k_2 = sp.symbols('y_{f2}(k) y_{f2}(k-1) y_{f2}(k-2)')

y2_k_eq = -y2_k_1*aH_fy2[1] - y2_k_2*aH_fy2[2] + y_k*bH_fy2[0] + y_k_1*bH_fy2[1] + y_k_2*bH_fy2[2]

equ = sp.Eq(y2_k, y2_k_eq)
yk = sp.solve(equ, y_k)[0]
```

An equation for $y(kT)$ can be obtained by isolating the $y(kT)$ term in the $y_{f2}(kT)$ equation. The $y_{f2}(kT)$ equation is

$$y_{f2}(kT) = \frac{4y(k)}{T^2+4T+4} - \frac{8y(k-1)}{T^2+4T+4} + \frac{4y(k-2)}{T^2+4T+4} - \frac{2y_{f2}(k-1)(T-2)}{T+2} - \frac{y_{f2}(k-2)\left(T^2-4T+4\right)}{T^2+4T+4}$$

Isoalting $y(kT)$ gives

$$y(kT) = \frac{T^2y_{f2}(k)}{4} + \frac{T^2y_{f2}(k-1)}{2} + \frac{T^2y_{f2}(k-2)}{4} + Ty_{f2}(k) - Ty_{f2}(k-2) + 2y(k-1) - y(k-2) + y_{f2}(k) - 2y_{f2}(k-1) + y_{f2}(k-2)$$

The above equation depends only on present and past values of $y_{f2}(kT)$, in which the present value can be obtained via the measurment model ($\phi^T(t)\theta$) and past values of $y(kT)$

Bilinear Transformation of Control Signal u(t)

In [47]:

```python
T_R = simplify(T_/R)
S_R = simplify(S/R)

T_subd = T_
R_subd = R.subs(r1, r_1)
S_subd = collect(expand(S.subs([(s0,s_0), (s1,s_1), (r1, r_1)])), s)

T_R_subd = T_subd/R_subd
S_R_subd = simplify(S_subd/R_subd)

# # bilinear transformation of T/R and S/R in terms of plant params
# TR = collect(simplify(expand(T_R_subd.subs(s, bilinear_T))), q)
# SR = collect(simplify(expand(S_R_subd.subs(s, bilinear_T))), q)

# bilinear transformation of T/R and S/R in terms of control params
TR = collect(simplify(expand(T_R.subs(s, bilinear_T))), q)
SR = collect(simplify(expand(S_R.subs(s, bilinear_T))), q)
```

The control signal of the system is given by

$$u(t) = \frac{T}{R}u_c(t) - \frac{S}{R}y(t)$$

$$u(t) = \frac{a_0+s}{c\left(r_1+s\right)}u_c(t) - \frac{ss_0+s_1}{r_1+s}y(t)$$

$$u(t) = \frac{a_0+s}{c\left(a_0-d+s+1\right)}u_c(t) - \frac{-a_0d+a_0+d^2-d-s\left(a_0d-a_0-d^2+d\right)}{c\left(a_0-d+s+1\right)}y(t)$$

This however, must also be converted to the discrete time doamin with a bilinear transformation as well. This will be done by directly performing the transformation on $\frac{T}{R}$ and $\frac{S}{R}$ (no filtering) and using the α and β coefficients to derive difference equations for $u_c(kT)$ and $y(kT)$ respectivley.

The bilinear transformations of $\frac{T}{R}$ and $\frac{S}{R}$ are

$$\frac{T}{R}\Bigg|_{s \rightarrow \frac{2(1-\frac{1}{q})}{T(1+\frac{1}{q})}} = \frac{Ta_0+q\left(Ta_0+2\right)-2}{c\left(Tr_1+q\left(Tr_1+2\right)-2\right)}$$

$$\frac{S}{R}\Bigg|_{s \rightarrow \frac{2(1-\frac{1}{q})}{T(1+\frac{1}{q})}} = \frac{Ts_1+q\left(Ts_1+2s_0\right)-2s_0}{Tr_1+q\left(Tr_1+2\right)-2}$$

In [48]:

```python
obj_TR = numden_coeff(TR, q)
obj_SR = numden_coeff(SR, q)

aTR = obj_TR.lst_denum_coeff
bTR = obj_TR.lst_num_coeff

aSR = obj_SR.lst_denum_coeff
bSR = obj_SR.lst_num_coeff
```

For $\frac{T}{R}$, the coefficients of the numerator and denominator are

$$\alpha\frac{T}{R} = \left[1, \ \frac{Tr_1-2}{Tr_1+2}\right]$$

and

$$\beta\frac{T}{R} = \left[\frac{Ta_0+2}{c\left(Tr_1+2\right)}, \ \frac{Ta_0-2}{c\left(Tr_1+2\right)}\right]$$

while the coefficients of the numerator and denominator for $\frac{S}{R}$ are

$$\alpha\frac{S}{R} = \left[1, \ \frac{Tr_1-2}{Tr_1+2}\right]$$

and

$$\beta\frac{S}{R} = \left[\frac{Ts_1+2s_0}{Tr_1+2}, \ \frac{Ts_1-2s_0}{Tr_1+2}\right]$$

In [49]:

```python
uc_k, uc_k_1 = sp.symbols('u_{c}(k) u_{c}(k-1)')

uk = -u_k_1*aTR[1] + uc_k*bTR[0] + uc_k_1*bTR[1] - y_k*bSR[0] - y_k_1*bSR[1]
```

The difference equation representing the control signal becomes

$$u(k) = -\frac{u(k-1)\left(Tr_1-2\right)}{Tr_1+2} - \frac{y(k)\left(Ts_1+2s_0\right)}{Tr_1+2} - \frac{y(k-1)\left(Ts_1-2s_0\right)}{Tr_1+2} + \frac{u_c(k)\left(Ta_0+2\right)}{c\left(Tr_1+2\right)} + \frac{u_c(k-1)\left(Ta_0-2\right)}{c\left(Tr_1+2\right)}$$

part 3

In [16]:

```
T_val = 0.1
a_0_val = 1

y1_k_ = y1_k.subs(T,T_val)
y0_k_ = y0_k.subs(T,T_val)
u0_k_ = u0_k.subs(T,T_val)
yk_ = yk.subs(T,T_val)
uk_ = uk.subs([(T,T_val),(s0, s_0), (s1, s_1), (r1 ,r_1), (a0, a_0_val)])

# Convert control equation derived above symbolically into numeric function that can be used in implementatio
y1_k_func = sp.lambdify([y_k, y_k_2, y1_k_1, y1_k_2], y1_k_)
y0_k_func = sp.lambdify([y_k, y_k_1, y_k_2, y0_k_1, y0_k_2], y0_k_)
u0_k_func = sp.lambdify([u_k, u_k_1, u_k_2, u0_k_1, u0_k_2], u0_k_)
yk_func = sp.lambdify([y2_k, y2_k_1, y2_k_2, y_k_1, y_k_2], yk_)
uk_func = sp.lambdify([u_k_1, uc_k, uc_k_1, y_k, y_k_1, c, d], uk_)
```

The for the implementation of the design, a sampling period of 0.1 ($T = 0.1$) and an observer polynomial parameter of 1 ($a_0 = 1$) will be used. The difference equations for $y_1(kT), y_0(kT), u_0(kT), u(kT)$ and $y(kT)$ become

$$y_1(kT) = 0.0453514739229025y(k) - 0.0453514739229025y(k-2) + 1.80952380952381y_1(k-1) - 0.81859410430839y_1(k-2)$$

$$y_0(kT) = 0.0022675736961451 2y(k) + 0.00453514739229025y(k-1) + 0.00226757369614512y(k-2) + 1.80952380952381y_0(k-1) - 0.81859410430839y_0(k-2)$$

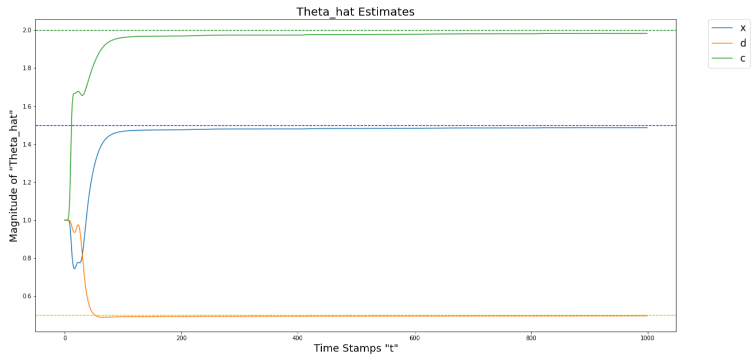
$$u_0(kT) = 0.00226757369614512u(k) + 0.00453514739229025u(k-1) + 0.00226757369614512u(k-2) + 1.80952380952381u_0(k-1) - 0.81859410430839u_0(k-2$$

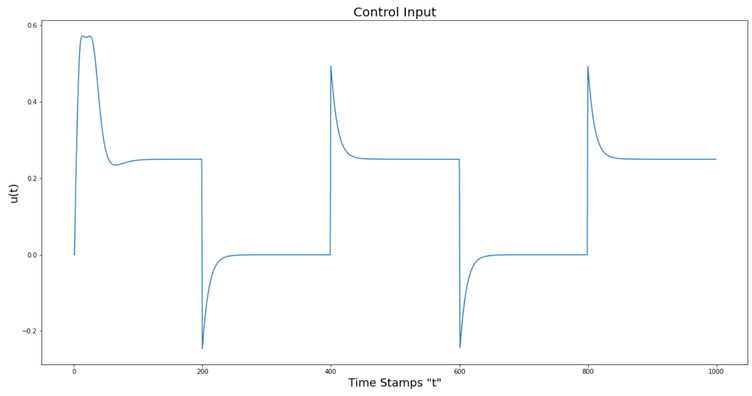
$$y(kT) = 2y(k-1) - y(k-2) + 1.1025y_{f2}(k) - 1.995y_{f2}(k-1) + 0.9025y_{f2}(k-2)$$

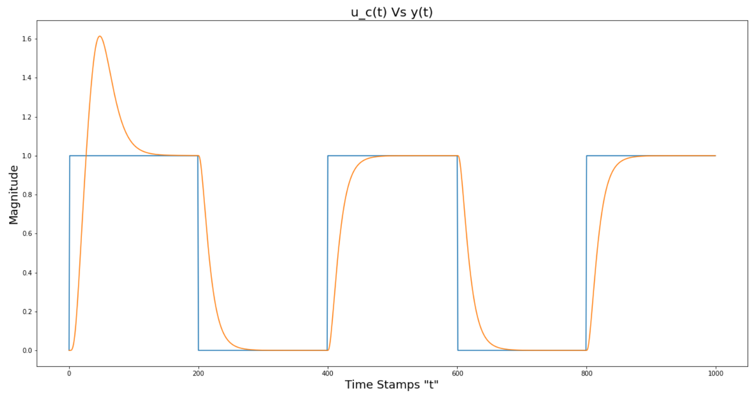
$$u(kT) = -\frac{u(k-1)(-0.1d-1.8)}{2.2-0.1d} + \frac{2.1u_c(k)}{c(2.2-0.1d)} - \frac{1.9u_c(k-1)}{c(2.2-0.1d)} - \frac{2.1y(k)(-d(2-d)+1)}{c(2.2-0.1d)} + \frac{1.9y(k-1)(-d(2-d)+1)}{c(2.2-0.1d)}$$

In [17]:

```
sample_depth = int(100/T_val) # 1000 samples totalling 100 seconds (since sample time T is 0.1 secons)
sample_range = range(sample_depth)
starting_samples = 2

# calculation of input signal
t = [i for i in sample_range]
u_c = np.ones(sample_depth)
u_c[np.where([m.sin(t[i]*m.pi*T_val/20)<=0 for i in sample_range])] = 0

# actual plant parameters
c = 2
d = 0.5
x = d + 1


theta0 = np.array([x, d, c]).reshape(-1,1)
theta_hat = [np.array([1]*3).reshape(-1,1) for _ in range(starting_samples)] # list for storing estimated para

y = [0]*starting_samples
u = [0]*starting_samples

# starting filtered signal values
y2 = [0]*starting_samples
y1 = [0]*starting_samples
y0 = [0]*starting_samples
u0 = [0]*starting_samples

lam = 0 # forgeting factor
I = np.identity(3)
p = 1000*I # P matrix

for k in range(2,sample_depth):
    phi = np.array([-y1[-1], -y0[-1], u0[-1]]).reshape(-1,1)

    theta_hat.append(theta_hat[-1] + T_val*(p@phi)*(phi.T@theta0 - phi.T@theta_hat[-1]))
    p = p + T_val*(lam*I - p@phi@phi.T)@p

    y2.append(np.reshape(phi.T@theta0, ()))

    y.append(np.reshape(yk_func(y2[k], y2[k-1], y2[k-2], y[k-1], y[k-2]), ()))
    u.append(np.reshape(uk_func(u[k-1], u_c[k], u_c[k-1], y[k], y[k-1], theta_hat[-1][2], theta_hat[-1][1]),

    y1.append(np.reshape(y1_k_func(y[k], y[k-2], y1[k-1], y1[k-2]), ()))
    y0.append(np.reshape(y0_k_func(y[k], y[k-1], y[k-2], y0[k-1], y0[k-2]), ()))
    u0.append(np.reshape(u0_k_func(u[k], u[k-1], u[k-2], u0[k-1], u0[k-2]), ()))

# ploting of estimated parameters
df_theta = pd.DataFrame(np.asarray(theta_hat).reshape(-1,3,), columns=['x', 'd', 'c'])
theta_hat_ploter(df_theta, theta0, 'Theta_hat Estimates')

# ploting of control sig
plt.plot(t,u)
plt.title('Control Input', fontsize=20)
plt.ylabel('u(t)', fontsize=18)
plt.xlabel('Time Stamps "t"', fontsize=18)
plt.show()

# ploting of input vs output
plt.plot(t,u_c)
plt.plot(t,y)
plt.title('u_c(t) Vs y(t)', fontsize=20)
plt.ylabel('Magnitude', fontsize=18)
plt.xlabel('Time Stamps "t"', fontsize=18)
plt.show()
```







Problem 2

part 1

In [18]:

```
G1__ = 1/(s + 1)
G2__ = 2/(s+0.5)
G__ = collect(expand(G1__*G2__), s)
G__
```

Out[18]:

$$\frac{2}{s^2 + 1.5s + 0.5}$$

In [19]:

```
H__q = co.sample_system(co.tf([2], [1, 1.5, 0.5]), Ts=2, method='zoh')
```

The pulse tranfer function can be obtained by taking performing a zero order hold on the plant. The zero order hold equation is

$$H(q) = (1 - q^{-1})Z[L^{-1}[\frac{G(s)}{s}](nT)](q)$$

If T = 2s, then the pulse function is

$$\frac{1.598z + 0.588}{z^2 - 0.5032z + 0.04979} \quad dt = 2$$

Which matches exactly the pulse transfer function given in the homework

part 2

The roots of the numerator of $H(q)$ $(q + \frac{0.587984}{1.59831})$ are

In [20]:

```
np.roots([1, 0.587984/1.59831])
```

Out[20]:

```
array([-0.36787857])
```

Since -0.36787857 is sufficiently far from the unit circle edge, it can be canceld without risk.

In [21]:

```
a1, a2, b0, b1 = sp.symbols('a_{1} a_{2} b_{0} b_{1}')
am1, am2, bm0, bm1 = sp.symbols('a_{m1} a_{m2} b_{m0} b_{m1}')
s, q = sp.symbols('s q')
r1, s0, s1, a0 = sp.symbols('r_1 s_0 s_1 a_0')
t0 = sp.symbols('t_0')

A = q**2 + a1*q + a2
B = b0*q + b1
```

```
A_pol = sp.Poly(A)
B_pol = sp.Poly(B)

B_minus = b0
B_plus = simplify(B/B_minus)
H = B/A
```

$$B = B^{-}B^{+} = (b_0)(q + \frac{b_1}{b_0})$$

Therfore,

$$Deg(B^{-}) = 0$$

and

$$Deg(B^{+}) = Deg(B) = Deg(B_m) = 1$$

Additionally

$$Deg(A) = Deg(A_m) = 2$$

$$Deg(S) = Deg(R) = Deg(A) - 1 = 2 - 1 = 1$$

$$Deg(R^{'}) = Deg(R) - Deg(B^{+}) = 1 - 1 = 0$$

$$Deg(A_0) = Deg(A) + Deg(R^{'}) - Deg(A_m) = 2 + 0 - 2 = 0$$

In [22]:

```
A0 = 1
S = s0*q + s1
R = q + r1
R_prime = 1
Am = q**2 + am1*q + am2
Bm = Am.subs(q,1)*q**B_pol.degree()
# T = simplify(A0*Bm/B_minus)
T = q*t0
```

The control polynomials become

$$A_0 = 1$$

$$S = qs_0 + s_1$$

$$R = q + r_1$$

$$R^{'} = 1 \quad \Rightarrow \quad R = B^{+}R^{'} = B^{+} = q + \frac{b_1}{b_0}$$

$$A_m = a_{m1}q + a_{m2} + q^2$$

Additionally, to achive unity gain, the final value theorem can implemented on A_m to obtain the value of B_m. This is achived by the equation $B_m = A_m(1)q^m$.This way, when k goes to infinity ($q \rightarrow 1$), $G_m = 1$ and $Deg(B_m) = Deg(B)$. Therfore,

$$B_m = q\left(a_{m1} + a_{m2} + 1\right)$$

From this result, T can be calculated

$$T = A_0\frac{B_m}{B^{-}} = \frac{q\left(a_{m1} + a_{m2} + 1\right)}{b_0} = qt_0$$

In [23]:

```
LHS_coeffs = sp.Poly(A*R_prime + B_minus*S, q).coeffs()[::-1]
RHS_coeffs = sp.Poly(A0*Am, q).coeffs()[::-1]

r_1 = b1/b0
s_0 = sp.solve(sp.Eq(LHS_coeffs[1], RHS_coeffs[1]), s0)[0]
s_1 = sp.solve(sp.Eq(LHS_coeffs[0], RHS_coeffs[0]), s1)[0]
t_0 = (am1 + am2 + 1)/b0
```

We can see that

$$r_1 = \frac{b_1}{b_0}$$

and

$$t_0 = \frac{a_{m1} + a_{m2} + 1}{b_0}$$

Solving the diophantine equation

$$AR^{'} + B^{-}S = A_0A_m \quad \Rightarrow \quad a_1q + a_2 + q^2 \ + \ b_0qs_0 + s_1 \ = \ a_{m1}q + a_{m2} + q^2$$

yeilds the following control paramters

$$s_0 = \frac{-a_1 + a_{m1}}{b_0}$$

$$s_1 = \frac{-a_2 + a_{m2}}{b_0}$$

In [24]:

```
S_ = S.subs([(s0, s_0), (s1, s_1)])
R_ = R.subs(r1, r_1)
T_ = T.subs(t0, t_0)
```

The control polynomials in terms of control parameters are

$$R = q + \frac{b_1}{b_0}$$

$$S = \frac{q\left(-a_1 + a_{m1}\right)}{b_0} + \frac{-a_2 + a_{m2}}{b_0}$$

$$T = \frac{q\left(a_{m1} + a_{m2} + 1\right)}{b_0}$$

Part 3

We know that the pulse transfer function is of the form

$$H(q) = \frac{b_0q + b_1}{a_1q + a_2 + q^2}$$

Therefore, the measurment model can be found to be

$$y(t+2) = -a_1y(t+1) - a_2y(t) + b_0u(t+1) + b_1u(t)$$

$$\Rightarrow y(t) = -a_1y(t-1) - a_2y(t-2) + b_0u(t-1) + b_1u(t-2)$$

Therfore

$$y(t) = \phi^T(t)\theta = [-y(t-1) \ \ -y(t-2) \ \ u(t-1) \ \ u(t-2)][a_1 \ a_2 \ b_0 \ b_1]^T$$

Control equation u(t)

In [25]:

```
TR = simplify(T_/R_)
SR = simplify(S_/R_)

obj_TR = numden_coeff(TR, q)
obj_SR = numden_coeff(SR, q)

aTR = obj_TR.lst_denum_coeff
bTR = obj_TR.lst_num_coeff

aSR = obj_SR.lst_denum_coeff
bSR = obj_SR.lst_num_coeff
```

For $\frac{T}{R}$, the coefficients of the numerator and denominator are

$$\alpha\frac{T}{R} = \left[1, \frac{b_1}{b_0}\right]$$

and

$$\beta\frac{T}{R} = \left[\frac{a_{m1} + a_{m2} + 1}{b_0}, 0\right]$$

while the coefficients of the numerator and denominator for $\frac{S}{R}$ are

$$\alpha\frac{S}{R} = \left[1, \frac{b_1}{b_0}\right]$$

and

$$\beta\frac{S}{R} = \left[\frac{-a_1 + a_{m1}}{b_0}, \frac{-a_2 + a_{m2}}{b_0}\right]$$

In [26]:

```
u_k, u_k_1, uc_k, y_k, y_k_1, y_k_2 = sp.symbols('u(k) u(k-1) u_{c}(k) y(k) y(k-1) y(k-2)')

uk = -u_k_1*aTR[1] + uc_k*bTR[0] - y_k*bSR[0] - y_k_1*bSR[1]
```

The control equation in terms of the plant and model parameters is

$$u(t) = -\frac{b_1 u(k-1)}{b_0} + \frac{u_c(k)\left(a_{m1} + a_{m2} + 1\right)}{b_0} - \frac{y(k)\left(-a_1 + a_{m1}\right)}{b_0} - \frac{y(k-1)\left(-a_2 + a_{m2}\right)}{b_0}$$

In [27]:

```
co.sample_system(co.tf([1], [1, 2, 1]), Ts=2, method='zoh') # pulse transfer function of model
```

Out[27]:

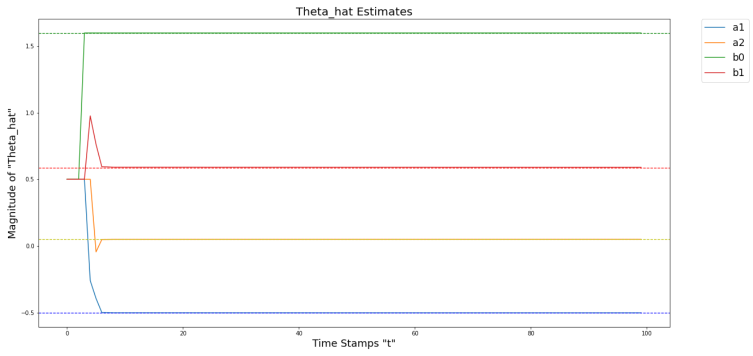
$$\frac{0.594z + 0.1537}{z^2 - 0.2707z + 0.01832} \quad dt = 2$$

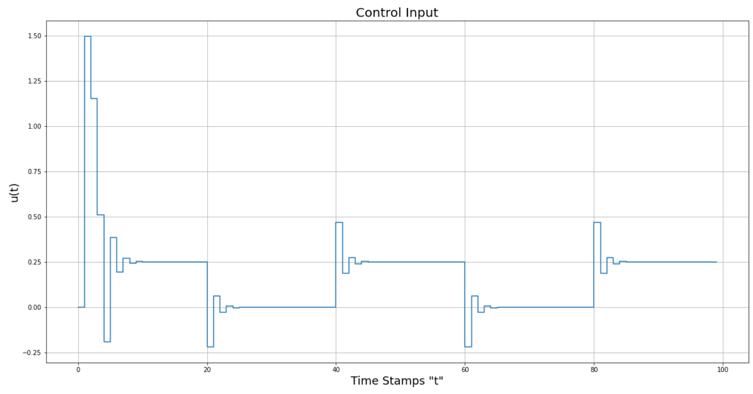
We can see from the pulse transfer function of model above that $a_{m1} = -0.2707$ and $a_{m2} = 0.01832$

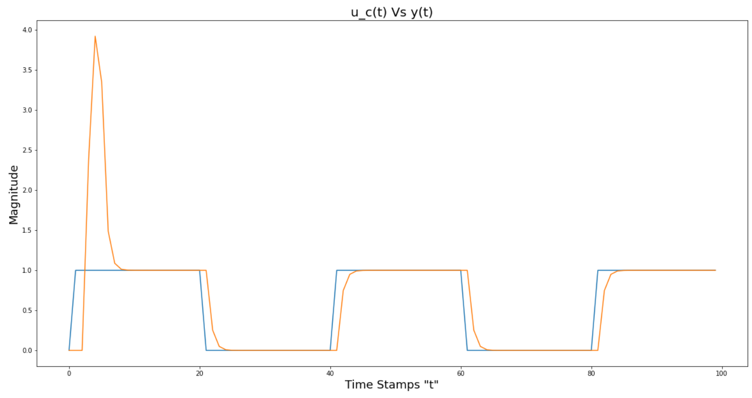
part 4

In [28]:

```
# Tunable components vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
sample_depth = 100
starting_samples = 2

# starting estimate for plant parameters
est = 0.5

lam = 1 # forgeting factor lambda

# true plant parameters
a_1 = -0.503215
a_2 = 0.0497871
b_0 = 1.59831
b_1 = 0.587984
theta0 = np.array([a_1, a_2, b_0, b_1]).reshape(-1,1)

# Tunable components ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

theta_hat = [np.array([est]*len(theta0)).reshape(-1,1) for _ in range(starting_samples)]

# model parameters
a_m1 = -0.2707
a_m2 = 0.01832

# Convert control equation derived above symbolically into numeric function
uk_subd = uk.subs([(am1, a_m1), (am2, a_m2)]) # subing in true values for am1 and am2 into control input funct
uk_func = sp.lambdify([u_k_1, uc_k, y_k, y_k_1, a1, a2, b0, b1], uk_subd)

# System input
sample_range = range(sample_depth)
t = [i for i in sample_range]
u_c = np.ones(sample_depth)
u_c[np.where([m.sin(t[i]*m.pi/20)<=0 for i in sample_range])] = 0

y = [0]*starting_samples
u = [0]*starting_samples

I = np.identity(len(theta0))
p = 1000*I

for k in range(starting_samples ,sample_depth):
    phi = np.array([-y[-1], -y[-2], u[-1], u[-2]]).reshape(-1,1)

    y.append(np.reshape(phi.T@theta0, ()))
    K = p@phi@inv(lam + phi.T@p@phi)
    p = (I - K@phi.T)@p/lam
    theta_hat.append(theta_hat[-1] + K*(y[k] - phi.T@theta_hat[-1]))

    u.append(np.reshape(uk_func(u[k-1], u_c[k], y[k], y[k-1],
                                theta_hat[-1][0], theta_hat[-1][1],
                                theta_hat[-1][2], theta_hat[-1][3]), ()))

# ploting of estimated parameters
df_theta = pd.DataFrame(np.asarray(theta_hat).reshape(-1,4,), columns=['a1', 'a2', 'b0', 'b1'])
theta_hat_ploter(df_theta, theta0, 'Theta_hat Estimates')

# ploting of control sig
plt.step(t,u)
plt.title('Control Input', fontsize=20)
plt.ylabel('u(t)', fontsize=18)
plt.xlabel('Time Stamps "t"', fontsize=18)
plt.grid()
plt.show()

# ploting of input vs output
plt.plot(t,u_c)
plt.plot(t,y)
plt.title('u_c(t) Vs y(t)', fontsize=20)
plt.ylabel('Magnitude', fontsize=18)
plt.xlabel('Time Stamps "t"', fontsize=18)
plt.show()
```







Step 5

In [29]:

```
x0, y0, r01, s00, s01 = sp.symbols('x_0 y_0 r_{01} s_{00} s_{01}')

Ad = q - 1 # Disturbance filter

# original Control Polynomials
R0 = q + r01
S0 = s00*q + s01
T = q*t0

X = q + x0
Y = y0

X_ = q
Y_ = (-X_*R0/B).subs(q,1)

R = collect(expand(X_*R0 + Y*B),q)
S = collect(expand(X_*S0 - Y*A), q)

r_01 = r_1
s_00 = s_0
s_01 = s_1

b_0 = sp.solve(sp.Eq(Y_, y0), b0)[0]
R_prime = simplify((R/Ad).subs(b0, b_0))
```

x_0 will be chosen to be equal to 0. This introduces a pole at the origin. Since the new R is given by

$A_d R^{'} = R_0 X + YB \quad \Rightarrow \quad (q-1)R^{'} = (q + r_{01})(q + x_0) + Y(b_0 q + b_1)$ Y must be a scalar value to maintain the order of R. This is done by finding the steady state of the above equation and solving for y_0. This gives

$$Y = y_0 = -\frac{r_{01}+1}{b_0+b_1}$$

Pluging this result back into the equation for R gives

$$R = \ (q)(q+r_{01}) \ + \ (-\frac{r_{01}+1}{b_0+b_1})(b_0q+b_1) \ = \ b_1y_0 + q^2 + q\left(b_0y_0 + r_{01}\right)$$

Similarly,

$$S = XS_0 - YA = \ -a_2y_0 + q^2\left(s_{00} - y_0\right) + q\left(- a_1y_0 + s_{01}\right)$$

Derivations of Equations of Filtered System

Filtered Output Equation y_f(t)

$$A_d(q)A(q)y(t) = A_d(q)B(q)(u(t) + v(t))$$

$$\Rightarrow q(1 - q^{-1})A(q)y(t) = q(1 - q^{-1})B(q)(u(t) + v(t))$$

$$\Rightarrow (1 - q^{-1})A(q)y(t) = (1 - q^{-1})B(q)(u(t) + v(t))$$

$$\Rightarrow A(q)y_f(t) = B(q)u_f(t) + e(t)$$

$$\Rightarrow \ (a_1q + a_2 + q^2)y_f(t) = \ (b_0q + b_1)u_f(t) + e(t)$$

Therefore

$$y_f(t+2) = \ -a_1y_f(t+1) - a_2y_f(t) + b_0u_f(t+1) + b_1u_f(t)$$

$$\Rightarrow \ y_f(t) = \ -a_1y_f(t-1) - a_2y_f(t-2) + b_0u_f(t-1) + b_1u_f(t-2)$$

This means that the measurment model is

$$y_f(t) = \phi^T\theta = [-y_f(t-1) \ -y_f(t-2) \ u_f(t-1) \ u_f(t-2) \ ][a_1 \ a_2 \ b_0 \ b_1]^T$$

Output Equation y(t)

We know that

$$y_f(t) = A_d(q^{-1})y(t)$$

$$\Rightarrow \ y_f(t) = \ (1 - q^{-1})y(t)$$

$$\Rightarrow \ y_f(t) = y(t) - y(t-1)$$

$$\Rightarrow \ y(t) = y_f(t) + y(t-1)$$

Filtered Control Input Equation u_f(t)

Similarly

$$u_f(t) = A_d(q^{-1})u(t)$$

$$\Rightarrow \ u_f(t) = u(t) - u(t-1)$$

Control Equation u(t)

In [30]:

```
TR = simplify(T/R)
SR = simplify(S/R)

obj_TR = numden_coeff(TR, q)
obj_SR = numden_coeff(SR, q)

aTR = obj_TR.lst_denum_coeff
bTR = obj_TR.lst_num_coeff

aSR = obj_SR.lst_denum_coeff
bSR = obj_SR.lst_num_coeff
```

For $\frac{T}{R}$, the coefficients of the numerator and denominator are

$$\alpha\frac{T}{R} = \ \left[1, \ b_0y_0 + r_{01}, \ b_1y_0\right]$$

and

$$\beta\frac{T}{R} = \ \left[t_0, \ 0\right]$$

while the coefficients of the numerator and denominator for $\frac{S}{R}$ are

$$\alpha\frac{S}{R} = \ \left[1, \ b_0y_0 + r_{01}, \ b_1y_0\right]$$

and

$$\beta\frac{S}{R} = \ \left[s_{00} - y_0, \ -a_1y_0 + s_{01}, \ -a_2y_0\right]$$

In [31]:

```
u_k_1, u_k_2, uc_k_1, y_k, y_k_1, y_k_2 = sp.symbols('u(k-1) u(k-2) u_{c}(k-1) y(k) y(k-1) y(k-2)')

uk_ = -u_k_1*aTR[1] -u_k_2*aTR[2] + uc_k_1*bTR[0] - y_k*bSR[0] - y_k_1*bSR[1] - y_k_2*bSR[2]
uk = simplify(uk_.subs([(t0, t_0), (y0, Y_), (r01, r_01), (s00, s_00), (s01, s_01)]))
```

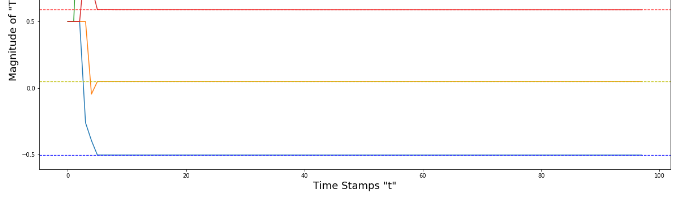
The control equation becomes

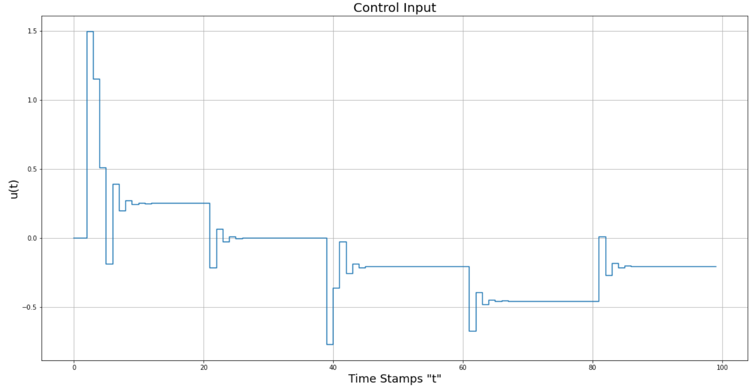
$$u(t) = \ a_2y(k-2)y_0 - b_1u(k-2)y_0 + t_0u_c(k-1) - u(k-1)\left(b_0y_0 + r_{01}\right) - y(k)\left(s_{00} - y_0\right) - y(k-1)\left(- a_1y_0 + s_{01}\right)$$

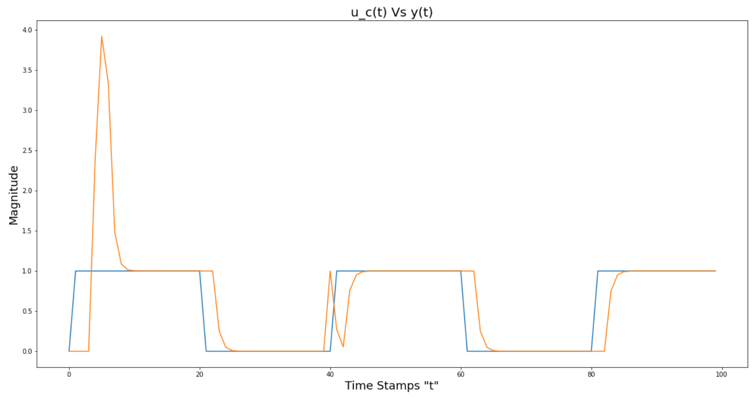
$$\Rightarrow \ u(t) = \frac{-a_2y(k-2) + b_1u(k-2) + u(k-1)\left(b_0 - b_1\right) + u_c(k-1)\left(a_{m1} + a_{m2} + 1\right) - y(k)\left(- a_1 + a_{m1} + 1\right) - y(k-1)\left(a_1 - a_2 + a_{m2}\right)}{b_0}$$

In [32]:

```
# Tunable components vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
sample_depth = 100
starting_samples = 3

# starting estimate for plant parameters
est = 0.5

lam = 1 # forgeting factor lambda

# true plant parameters
a_1 = -0.503215
a_2 = 0.0497871
b_0 = 1.59831
b_1 = 0.587984
theta0 = np.array([a_1, a_2, b_0, b_1]).reshape(-1,1)

# Tunable components ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

theta_hat = [np.array([est]*len(theta0)).reshape(-1,1)]

# model parameters
a_m1 = -0.2707
a_m2 = 0.01832

# Convert control equation derived above symbolically into numeric function
uk_subd = uk.subs([(am1, a_m1), (am2, a_m2)])
uk_func = sp.lambdify([u_k_1, u_k_2, uc_k_1, y_k, y_k_1, y_k_2,
                       a1, a2, b0, b1], uk_subd)

sample_range = range(sample_depth)

# system input
t = [i for i in sample_range]
u_c = np.ones(sample_depth)
u_c[np.where([m.sin(t[i]*m.pi/20)<=0 for i in sample_range])] = 0

# system disturbance starting at t=40
dist = np.zeros(sample_depth)
dist[np.where([t[i]>= 40 for i in sample_range])] = 1

y = [0]*starting_samples
u = [0]*starting_samples

yf = [0]*starting_samples
uf = [0]*starting_samples

I = np.identity(len(theta0))
p = 1000*I

for k in range(starting_samples ,sample_depth):
    phi = np.array([-yf[-1], -yf[-2], uf[-1], uf[-2]]).reshape(-1,1)
    yf.append(np.reshape(phi.T@theta0 + dist[k] - dist[k-1], ()))

    K = p@phi@inv(lam + phi.T@p@phi)
    p = (I - K@phi.T)@p/lam

    theta_hat.append(theta_hat[-1] + K*(yf[k] - phi.T@theta_hat[-1]))

    y.append(yf[k] + y[k-1])

    u.append(np.reshape(uk_func(u[k-1], u[k-2], u_c[k-1], y[k], y[k-1], y[k-2],
                                theta_hat[-1][0], theta_hat[-1][1],
                                theta_hat[-1][2], theta_hat[-1][3]), ()))

    uf.append(u[k] - u[k-1])


# ploting of estimated parameters
df_theta = pd.DataFrame(np.asarray(theta_hat).reshape(-1,4,), columns=['a1', 'a2', 'b0', 'b1'])
theta_hat_ploter(df_theta, theta0, 'Theta_hat Estimates')

# ploting of control sig
plt.step(t,u)
plt.title('Control Input', fontsize=20)
plt.ylabel('u(t)', fontsize=18)
plt.xlabel('Time Stamps "t"', fontsize=18)
plt.grid()
plt.show()

# ploting of input vs output
plt.plot(t,u_c)
plt.plot(t,y)
plt.title('u_c(t) Vs y(t)', fontsize=20)
plt.ylabel('Magnitude', fontsize=18)
plt.xlabel('Time Stamps "t"', fontsize=18)
plt.show()
```







Problem 3

In [33]:
```
r1, t0 = sp.symbols('r_1 t_0')

A = q**2 + a1*q + a2
B = b0*q + b1
B_minus = b0
B_plus = simplify(B/B_minus)
H = B/A
A0 = 1

S = s0*q + s1
R = q + r1 # define b1/b0 as r1
R_prime = 1
T = q*t0 # t0 = (am1 + am2 + 1)/b0
```

Part 1

$$H = \frac{b_0 q + b_1}{a_1 q + a_2 + q^2}$$

$$B = B^- B^+ = (b_0)(q + \frac{b_1}{b_0})$$

Therfore,

$$Deg(B^-) = 0$$

and

$$Deg(B^+) = Deg(B) = Deg(B_m) = 1$$

Additionally

$$Deg(A) = Deg(A_m) = 2$$

$$Deg(S) = Deg(R) = Deg(A) - 1 = 2 - 1 = 1$$

$$Deg(R^{'}) = Deg(R) - Deg(B^+) = 1 - 1 = 0$$

$$Deg(A_0) = Deg(A) + Deg(R^{'}) - Deg(A_m) = 2 + 0 - 2 = 0$$

The control polynomials become

$A_0 = 1$

$S = qs_0 + s_1$

$R = q + r_1$

$R^{'} = 1$

$A_m = a_{m1}q + a_{m2} + q^2$

Additionally, to achive unity gain, the final value theorem can implemented on A_m to obtain the value of B_m. This is achived by the equation $B_m = A_m(1)q^m$.This way, when k goes to infinity ($q \rightarrow 1$), $G_m = 1$ and $Deg(B_m) = Deg(B)$. Therfore,

$B_m = q\left(a_{m1} + a_{m2} + 1\right)$

From this result, T can be calculated

$T = qt_0$

Output Equation

In [34]:
```
coef_1 = R*B_minus/Am
coef_2 = S*B_minus/Am

coef_1_ = (R*B_minus).subs(q, q**(-1))
coef_2_ = (S*B_minus).subs(q, q**(-1))
```

The diophantine equation can be manipulated to give a formula for the output and subsequently, the measurment model.

$AR^{'} + B^-S = A_0A_m$

Multiplying both sides of the equation gives

$AR^{'}y(t) + B^-Sy(t) = A_0A_my(t)$

Since $Ay(t) = Bu(t)$ the above equation becomes

$R^{'}Bu(t) + B^-Sy(t) = A_0A_my(t)$

Isolating $y(t)$ on the right hand side of the equation gives

$y(t) = \frac{1}{A_oA_m}[R^{'}Bu(t) + B^-Sy(t)]$

Since $R^{'}B = R^{'}B^-B^+ = RB^-$ the above equation becomes

$y(t) = \frac{1}{A_oA_m}[RB^-u(t) + B^-Sy(t)]$

Substituting A_o, A_m, R, B^- and S with their polynomilas gives

$$y(t) = \frac{b_0\left(q + r_1\right)}{a_{m1}q + a_{m2} + q^2}u(t) + \frac{b_0\left(qs_0 + s_1\right)}{a_{m1}q + a_{m2} + q^2}y(t)$$

Noting that the numerator of both coefficients will have to be shifted by 2, considering the order of the denominator, we can rewrite the above equation as

$y(t) = b_0(q^{-1} + r_1q^{-2})u_f(t) + b_0(s_0q^{-1} + s_1q^{-2})y_f(t)$

Where $u_f(t)$ and $y_f(t)$ are ther filtered input and output respectivley. Expanding the above equation gives

$y(t) = b_0u_f(t-1) + b_0r_1u_f(t-2) + b_0s_0y_f(t-1) + b_0s_1y_f(t-2)$

For simplicity, let $b_0r_r = r_1^{'},\ b_0s_0 = s_0^{'}$ and $b_0s_1 = s_1^{'}$. The above equation becomes

$y(t) = b_0u_f(t-1) + r_1^{'}u_f(t-2) + s_0^{'}y_f(t-1) + s_1^{'}y_f(t-2)$

Therefore,

$y(t) = \phi^T\theta = [u_f(t-1),\ u_f(t-2),\ y_f(t-1),\ y_f(t-2)][b_0,\ r_1^{'},\ s_0^{'},\ s_1^{'}]^T$

The equation for $u_f(t)$ is given by

$u_f(t) = \frac{1}{A_m(q^{-1})A_0(q^{-1})}u(t) = \frac{1}{1+a_{m1}q^{-1}+a_{m2}q^{-2}}u(t)$

This can then be rearanged to give

$u_f(t) = -a_{m1}u_f(t-1) - a_{m2}u_f(t-2) + u(t)$

Similarly, the difference equation for the filtered output $y_f(t)$ is given by

$y_f(t) = \frac{1}{1+a_{m1}q^{-1}+a_{m2}q^{-2}}y(t) = -a_{m1}y_f(t-1) - a_{m2}y_f(t-2) + y(t)$

Control Law Equation

In [35]:
```
TR = simplify(T/R)
SR = simplify(S/R)

obj_TR = numden_coeff(TR, q)
obj_SR = numden_coeff(SR, q)

aTR = obj_TR.lst_denum_coeff
bTR = obj_TR.lst_num_coeff

aSR = obj_SR.lst_denum_coeff
bSR = obj_SR.lst_num_coeff
```

For $\frac{T}{R}$, the coefficients of the numerator and denominator are

$\alpha\frac{T}{R} = \left[1,\ r_1\right]$

and

$\beta\frac{T}{R} = \left[t_0,\ 0\right]$

while the coefficients of the numerator and denominator for $\frac{S}{R}$ are

$\alpha\frac{S}{R} = \left[1,\ r_1\right]$

and

$\beta\frac{S}{R} = \left[s_0,\ s_1\right]$

In [36]:

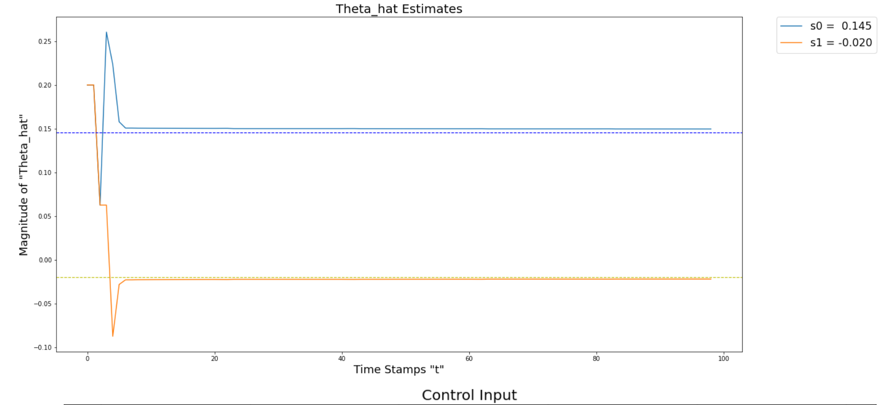
```
u_k, u_k_1, uc_k, y_k, y_k_1, y_k_2 = sp.symbols('u(k) u(k-1) u_{c}(k) y(k) y(k-1) y(k-2)')

uk = -u_k_1*aTR[1] + uc_k*bTR[0] - y_k*bSR[0] - y_k_1*bSR[1]
```

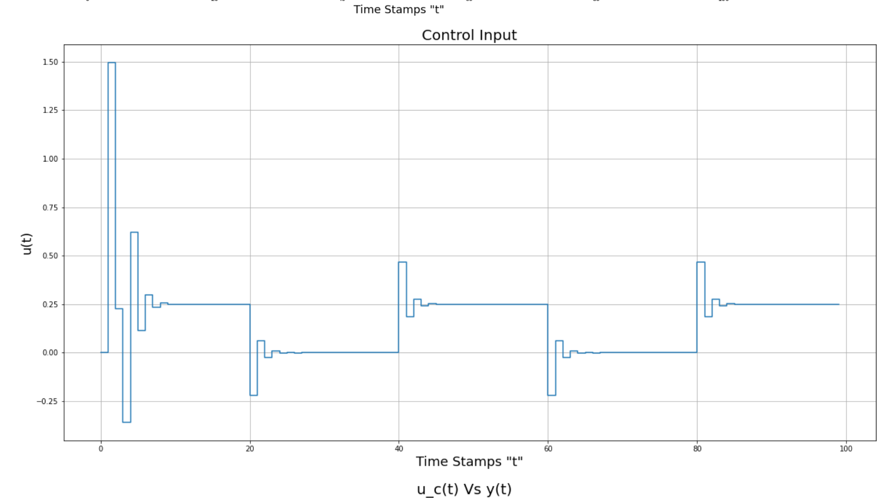
The control equation becomes

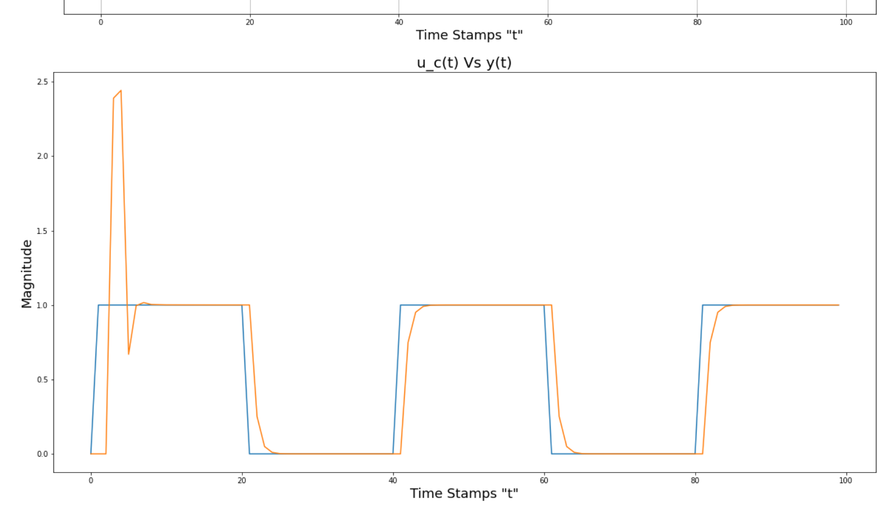
$$u(t) = -r_1 u(k-1) - s_0 y(k) - s_1 y(k-1) + t_0 u_c(k)$$

In [37]:

```
# Tunable components vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
sample_depth = 100
starting_samples = 2

# starting estimate for plant parameters
est = 0.1
est_offset = 5

lam = 1 # forgeting factor lambda

# model parameters
a_m1 = -0.2707
a_m2 = 0.01832

# true plant parameters
a_1 = -0.503215
a_2 = 0.0497871
b_0 = 1.59831
b_1 = 0.587984

# true control parameters (primed versions)
r_1_p = b_1
s_0_p = (-a_1 + a_m1)
s_1_p = (-a_2 + a_m2)

theta0 = np.array([b_0, r_1_p, s_0_p, s_1_p]).reshape(-1,1)

# Tunable components ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

theta_hat = [np.array([est]*len(theta0)).reshape(-1,1)]
theta_hat[0][0] = est*est_offset

# Convert control equation derived above symbolically into numeric function
uk_subd = uk.subs([(am1, a_m1), (am2, a_m2)])
uk_func = sp.lambdify([u_k_1, uc_k, y_k, y_k_1, r1, s0, s1, t0], uk_subd)

# system input
sample_range = range(sample_depth)
t = [i for i in sample_range]
u_c = np.ones(sample_depth)
u_c[np.where([m.sin(t[i]*m.pi/20)<=0 for i in sample_range])] = 0

y = [0]*starting_samples
u = [0]*starting_samples

yf = [0]*starting_samples
uf = [0]*starting_samples


I = np.eye(len(theta0))
p = 100*I

for k in range(starting_samples ,sample_depth):
    phi = np.array([uf[-1], uf[-2], yf[-1], yf[-2]]).reshape(-1,1)

    y.append(np.reshape(phi.T@theta0, ()))
    K = p@phi@inv(np.array(lam + phi.T@p@phi))
    theta_hat.append(theta_hat[-1] + K*(y[k] - phi.T@theta_hat[-1]))
    p = (I - K@phi.T)@p/lam

    # estimated control parameters
    r_1 = theta_hat[-1][1]/theta_hat[-1][0]
    s_0 = theta_hat[-1][2]/theta_hat[-1][0]
    s_1 = theta_hat[-1][3]/theta_hat[-1][0]
    t_0 = (a_m1 + a_m2 + 1)/theta_hat[-1][0]

    u.append(np.reshape(uk_func(u[k-1], u_c[k], y[k], y[k-1],
                                r_1, s_0, s_1, t_0), ()))

    uf.append(-a_m1*uf[k-1] - a_m2*uf[k-2] + u[k])
    yf.append(-a_m1*yf[k-1] - a_m2*yf[k-2] + y[k])

# ploting of estimated parameters
lst_cntrl_param = [theta_hat[i][-2::]/theta_hat[i][0] for i in range(len(theta_hat))]
df_theta = pd.DataFrame(np.asarray(lst_cntrl_param).reshape(-1,2,),
                        columns=[f's0 = {s_0_p/b_0: .3f}', f's1 = {s_1_p/b_0: .3f}'])

theta_hat_ploter(df_theta, theta0[-2::]/theta0[0], 'Theta_hat Estimates')

# ploting of control sig
plt.step(t,u)
plt.title('Control Input', fontsize=20)
plt.ylabel('u(t)', fontsize=18)
plt.xlabel('Time Stamps "t"', fontsize=18)
plt.grid()
plt.show()

# ploting of input vs output
plt.plot(t,u_c)
plt.plot(t,y)
plt.title('u_c(t) Vs y(t)', fontsize=20)
plt.ylabel('Magnitude', fontsize=18)
plt.xlabel('Time Stamps "t"', fontsize=18)
plt.show()
```


Theta_hat Estimates
s0 = 0.145
s1 = -0.020
Magnitude of "Theta_hat"
Time Stamps "t"


Control Input
u(t)
Time Stamps "t"


u_c(t) Vs y(t)
Magnitude
Time Stamps "t"