## 2-lv SVPWM (including overmodualtion)

- Choosing and importing: mdl_pmsm_2kW.py

```
from model.interfaces import PWM, SVPWM_2LV, SVPWM_3LV,Delay
                ……
delay = Delay(1)
pwm = PWM(enabled=False)
svpwm_2lv = SVPWM_2LV(enabled=True)
svpwm_3lv = SVPWM_3LV(enabled=False)
                ……
mdl = Drive(motor, mech, converter, delay, pwm, svpwm_2lv, svpwm_3lv,datalog)
```

- Obtaining $u_{ref}$

vector.py ⇒

```
class VectorCtrl:
……
# Outputs
tau_M_ref, tau_L = self.speed_ctrl.output(w_m_ref/self.p, w_M)
i_s_ref, tau_M  = self.current_ref.output(tau_M_ref, w_m, u_dc)
u_s_ref, e = self.current_ctrl.output(i_s_ref, i_s)
d_abc_ref, u_s_ref_lim = self.pwm.output(u_s_ref, u_dc, theta_m, w_m)
u_ss = u_s_ref * np.exp(1j * theta_m)
……
return d_abc_ref, self.pwm.T_s, u_ss
```

# 2-lv SVPWM (including overmodualtion)

- Pass $u_{ref}$ to the solver : interfaces.py

```python
def solve(mdl, d_abc, u_ref, u_dc, t_span, max_step=np.inf):

                                ……

if mdl.svpwm_2lv.enabled:
    # Sampling period
    T_s = t_span[-1] - t_span[0]
    # Compute the normalized switching spans and the corresponding states
    tn_sw, q_sw= mdl.svpwm_2lv(u_ref, u_dc)
    # Convert the normalized switching spans to seconds
    t_sw = t_span[0] + T_s * tn_sw
    # Loop over the switching time spans
    for i, t_sw_span in enumerate(t_sw):
        # Update the switching state vector (constant over the time span)
        mdl.q = abc2complex(q_sw[i])
        # Run the solver
        run_solver(t_sw_span)
```

- Class 2-lv_SVPWM: interfaces.py

```python
# %%
class SVPWM_2LV:
    '''

    This module is to provide Space Vector PWM modulation method
    for two-level inverter.

    The method is based on a demo model "Lookup Table-Based PMSM" from Plexim.
    Url: https://www.plexim.com/sites/default/files/demo_models_categorized/plecs/look_up_table_based_pmsm.pdf

    The overmodulation technique used here is based on:
    Guohui Yin, Jianwu Luo, Jie Wang, Hongtao Wang. “Grapic over-modulation
    technique for space-vector PWM”. In: Small and Special Electrical Machines (2014).
    URL: https://kns.cnki.net/kcms/detail/detail.aspx?FileName=WTDJ201402018&DbName=CJFQ2014
    '''
                            ……
```

# 3-lv SVPWM (including overmodulation)

- Choosing and importing: mdl_pmsm_2kW.py

```
from model.interfaces import PWM, SVPWM_2LV, SVPWM_3LV,Delay
                 ……
# %% Computational delay and PWM
delay = Delay(1)
pwm = PWM(enabled=False)
svpwm_2lv = SVPWM_2LV(enabled=False)
svpwm_3lv = SVPWM_3LV(enabled=True)
mdl = Drive(motor, mech, converter, delay, pwm, svpwm_2lv, svpwm_3lv,datalog)
```

- Obtaining $u_{ref}$

vector.py ⇒

```
class VectorCtrl:
……
# Outputs
tau_M_ref, tau_L = self.speed_ctrl.output(w_m_ref/self.p, w_M)
i_s_ref, tau_M  = self.current_ref.output(tau_M_ref, w_m, u_dc)
u_s_ref, e = self.current_ctrl.output(i_s_ref, i_s)
d_abc_ref, u_s_ref_lim = self.pwm.output(u_s_ref, u_dc, theta_m, w_m)
u_ss = u_s_ref * np.exp(1j * theta_m)
……
return d_abc_ref, self.pwm.T_s, u_ss
```

# 3-lv SVPWM (including overmodulation)

- Pass $u_{ref}$ to the solver : interfaces.py

```
def solve(mdl, d_abc, u_ref, u_dc, t_span, max_step=np.inf):
```

……

```
elif mdl.svpwm_3lv.enabled:
    # Sampling period
    T_s = t_span[-1] - t_span[0]
    # Compute the normalized switching spans and the corresponding states
    tn_sw, q_sw= mdl.svpwm_3lv(u_ref, u_dc, mdl.converter.delta_uc)
    # Convert the normalized switching spans to seconds
    t_sw = t_span[0] + T_s * tn_sw
    # Loop over the switching time spans
    for i, t_sw_span in enumerate(t_sw):
        # Update the switching state vector (constant over the time span)
        phase_neutral = np.argwhere(q_sw[i] == 0)
        mdl.q = abc2complex(q_sw[i])
        # Run the solver
        run_solver(t_sw_span)
    mdl.converter.delta_ucs.append(mdl.converter.delta_uc)
```

- Class 3_SVPWM: interfaces.py

```
class SVPWM_3LV:
    '''

    This module is to provide Space Vector PWM modulation method
    for three-level inverter.
```

……

# 3-lv SVPWM for the unbalancing neutral point

- Measuring the deviation voltage: interfaces.py

```
# Common code
def run_solver(t_span):
            ……

# Measuring the delta_uc
if mdl.svpwm_3lv.enabled:
    mdl.converter.meas_delta_uc(phase_neutral, mdl, sol)
            ……
```

- Class 3_SVPWM: interfaces.py

```
elif mdl.svpwm_3lv.enabled:
        ……
    for i, t_sw_span in enumerate(t_sw):
        # Update the switching state vector (constant over the time span)
        # Find which phase is on the state "0"
        phase_neutral = np.argwhere(q_sw[i] == 0)
        mdl.q = abc2complex(q_sw[i])
        # Run the solver
        run_solver(t_sw_span)
    # Logging the delta_uc
    mdl.converter.delta_ucs.append(mdl.converter.delta_uc)
```

# 3-lv SVPWM for the unbalancing neutral point

- Measuring the deviation voltage: converter.py

```
class Inverter:
    ……
# pylint: disable=R0903
def __init__(self, u_dc=540):
    ……
        # The current deviation voltage
        self.delta_uc = 0
        # The logged deviation voltage
        self.delta_ucs = []
        # Assuming the capacitor C=4700uF
        self.c = 4700e-6
    ……
def meas_delta_uc(self, phase_neutral, mdl, sol):
    self.t = sol.t
    i_s     = mdl.motor.current(sol.y[0])
    theta_m = mdl.motor.p*sol.y[1].real
    theta_m = np.mod(theta_m, 2*np.pi)
    i_ss = complex2abc(np.exp(1j * theta_m) * i_s)
    for i in range(len(phase_neutral)):
        i_np = i_ss[phase_neutral[i],:]
        # Motor mode or regenerating mode
        sgn=np.sign(mdl.speed_ref(self.t[-1])*mdl.mech.tau_L_ext(self.t[-1]))
        self.delta_uc += 1/self.c*(np.median(i_np))*(self.t[-1]-self.t[0])*sgn
```

## Harmonic Analyzer

- Functions

sm_drive.py

```
def harmonic_analyzer(self,t_start,t_end):
    '''

    "Lightweight non-uniform Fast Fourier Transform in Python".
    URL:https://github.com/jakevdp/nfft.git
```

……

- Plotting

sim.py

```
mdl.datalog.harmonic_analyzer(2,2.36)
ctrl.datalog.plot_neutralpointvoltage(mdl)
```