In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from qutip import *
import time

font = {'size'   : 18}

plt.rc('font', **font)
```

In [2]:
```
"""
bloch_vector: bloch vector of spin-1/2 moment
r: vector from spin to target position in nm

returns field in units of GHz * hbar/mu_B
"""
def dipole_field(bloch_vector, r):
    dist = np.linalg.norm(r)
    direction = r/dist

    mu0 = 4*np.pi*pow(10,-7) #vacuum permeability
    muB = 9.274*pow(10,-24) #bohr magneton SI
    hbar = 1.054571817*pow(10,-34) #hbar SI
    mag = -1*muB*bloch_vector #magnetic moment of a single electron

    return muB * mu0/(4*np.pi) * 1/hbar * (3*np.dot(mag,direction)*direction - mag)/(pow(dist,3) * pow(10,-27)) * pow(10,-9)
```

In [3]:
```
"""
Bext: external field strength (G)
times: list of times (1/GHz) over which to run experiment **todo - check if you're off by 2pi
modes: list of [[k, amplitude]] of magnon modes
a: inter-site magnon spacing (nm)
dist: sensor distance to index = 0 magnon site
indices: integer multiples of a at which we consider magnon field
max_magnon: maximum point in field-free exchange magnon dispersion (GHz)
mw_rabi: Rabi frequency associated with MW drive
mw_frequencies: list of CW frequencies over which we can sweep to look for the resonance (GHz)
"""
def run_expt(Bext, times, modes, a, dist, indices, max_magnon, mw_rabi, mw_frequencies, output = False):

    hbar = 1.054571817*pow(10,-34) #hbar SI
    gamma = -28 #electron gyromagnetic ratio GHz/T

    def magnon_single_mode_field(Bext,t,k,a,dist,amplitude,indices,max_magnon):
        B_total = np.array([0.,0.,0.])
        #get frequency in ghz from magnon dispersion relation
        omega = max_magnon/2*(1 - np.cos(k*a)) + 2.8*pow(10,-3)*Bext*np.sqrt(1-amplitude**2)
        for index in indices:
            #bloch vector precession
            bloch_vector = np.array([amplitude*np.cos(index*k*a - omega*t),amplitude*np.sin(index*k*a - omega*t), np.sqrt(1-amplitude**2)])
            direction = np.array([-1*index*a, 0, dist])
            B_total += dipole_field(bloch_vector, direction)

        return B_total

    def total_coupling_field(t, Bext, a, dist, indices, modes, max_magnon):
        total_field = np.array([0.,0.,0.])

        for mode in modes:
            total_field += magnon_single_mode_field(Bext,t,mode[0],a,dist,mode[1],indices, max_magnon)

        return total_field

    D = 2.87
    B_diag = Bext * 2* 1.4 * pow(10,-3)

    H0 = Qobj([[0,0,0],[0,D-B_diag,0],[0,0,D+B_diag]])

    nv_init = Qobj([0,1,0])
    states = []

    """
    args to pass: mw_rabi, mw_frequency, Bext, a, dist, indices, modes, max_magnon
    """

    def spc(t, args):
        f = total_coupling_field(t, args['Bext'], args['a'], args['dist'], args['indices'], args['modes'], args['max_magnon'])
        return f[0]/2+f[1]/(2j) + args['mw_rabi']*np.cos(args['mw_frequency']*t)

    def smc(t,args):
        f = total_coupling_field(t, args['Bext'], args['a'], args['dist'], args['indices'], args['modes'], args['max_magnon'])
        return f[0]/2-f[1]/(2j) + args['mw_rabi']*np.cos(args['mw_frequency']*t)

    def szc(t, args):
        f = total_coupling_field(t, args['Bext'], args['a'], args['dist'], args['indices'], args['modes'], args['max_magnon'])
        return f[2]

    for omega in mw_frequencies:
        Sp = Qobj([[0,np.sqrt(2),0],[0,0,0],[np.sqrt(2),0,0]])
        Sm = Qobj([[0,0,np.sqrt(2)],[np.sqrt(2),0,0],[0,0,0]])
        Sz = Qobj([[0,0,0],[0,-1,0],[0,0,1]])


        H = [H0,[Sp,spc],[Sm,smc], [Sz, szc]]
        st = time.time()

        out = sesolve(H, nv_init, times,args={'mw_rabi':mw_rabi, 'mw_frequency':omega, 'Bext':Bext, 'a':a, 'dist':dist, 'indices':indices, 'modes':modes, 'max_magnon':max_magnon})
        end = time.time()
        args = {'mw_rabi':mw_rabi, 'mw_frequency':omega, 'Bext':Bext, 'a':a, 'dist':dist, 'indices':indices, 'modes':modes, 'max_magnon':max_magnon}
        f = total_coupling_field(0, args['Bext'], args['a'], args['dist'], args['indices'], args['modes'], args['max_magnon'])
        if output:
            print(omega, f, end-st)
        states.append(out.states)
    return states
```

In [4]:
```
#Total magnon + MW field used to drive system leading to off diagonal matrix element
def drive_only(t, args):
    mw_rabi = 0.2 #100 MHz Rabi frequency of drive field
    mw_frequency = args['omega']

    return mw_rabi*np.cos(mw_frequency*t)
```

```python
nv_init = Qobj([0,1,0])
amps_no_magnons = []

for omega in np.linspace(25,35,50):
    Bext = 20
    B_diag = Bext * 2* 1.4 * pow(10,-3)
    D = 2.87

    H0 = Qobj([[0,0,0],[0,D-B_diag,0],[0,0,D+B_diag]])
    Hr1 = Qobj([[0,np.sqrt(2),np.sqrt(2)],[np.sqrt(2),0,0],[np.sqrt(2),0,0]])

    H = [H0,[Hr1, drive_only]]

    t = np.linspace(0,100,500)
    out = sesolve(H, nv_init, t,args={'omega':omega})
    #plt.plot([abs(out.states[i][0])**2 for i in range(len(out.states))])

    amps_no_magnons.append(max([abs(out.states[i][0])**2 for i in range(len(out.states))]))
```

In [6]:
```python
def magnon_single_mode_field(Bext,t,k,a,dist,amplitude,indices,max_magnon):
        B_total = np.array([0.,0.,0.])
        #get frequency in ghz from magnon dispersion relation
        omega = max_magnon/2*(1 - np.cos(k*a)) + 2.8*pow(10,-3)*Bext*np.sqrt(1-amplitude**2)
        for index in indices:
            #bloch vector precession
            bloch_vector = np.array([amplitude*np.cos(index*k*a - omega*t),amplitude*np.sin(index*k*a - omega*t), np.sqrt(1-amplitude**2)])
            direction = np.array([-1*index*a, 0, dist])
            B_total += dipole_field(bloch_vector, direction)

        return B_total

def total_coupling_field(t, Bext, a, dist, indices, modes, max_magnon):
    total_field = np.array([0.,0.,0.])

    for mode in modes:
        total_field += magnon_single_mode_field(Bext,t,mode[0],a,dist,mode[1],indices, max_magnon)

    return total_field
```

In [8]:
```python
k_results = {}
for k in np.linspace(np.pi/3,2*np.pi,10):
    print(k)
    omega_res = 100*(1 - np.cos(k/2))
    out = run_expt(20, np.linspace(0,100,500), [[k,0.1]], 0.5,0.5, np.linspace(-5,5,11), 200, 0.5, np.linspace(omega_res - 2,omega_res +12,50))
    k_results[k] = [max([abs(out[i][j][0])**2 for j in range(len(out[i]))]) for i in range(len(out))]
```

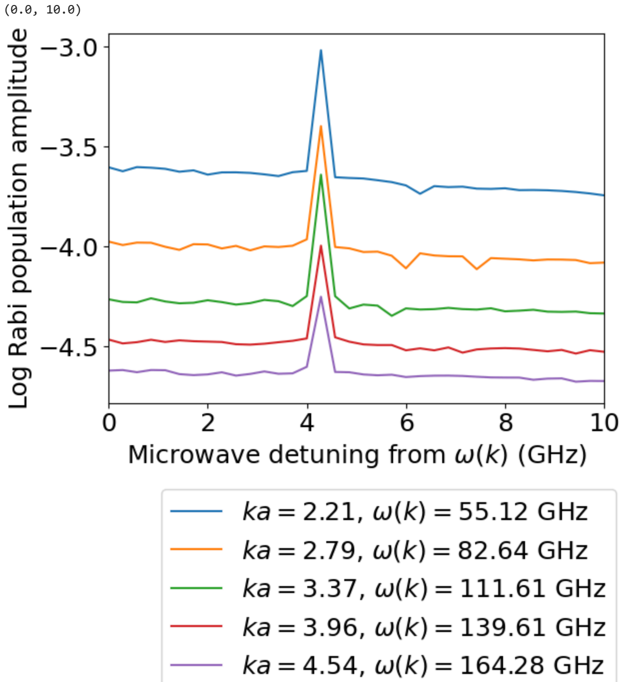
```
1.0471975511965976
1.6289739685280409
2.210750385859484
2.7925268031909276
3.3743032205223704
3.956079637853814
4.537856055185257
5.1196324725167
5.701408889848143
6.283185307179586
```

In [8]:
```python
k_results.keys()
```

Out[8]:
```
dict_keys([1.0471975511965976, 1.6289739685280409, 2.210750385859484, 2.7925268031909276, 3.3743032205223704, 3.956079637853814, 4.537856055185257, 5.1196324725167, 5.701408889848143,
6.283185307179586])
```

In [20]:
```python
for key in list(k_results.keys())[2:7]:
    omega_res = 100*(1 - np.cos(key/2))
    plt.plot(np.linspace(-2,12,50), [np.log10(i) for i in k_results[key]], label = "$ka = " + str(round(key,2))+ " $, $\omega(k) = $" + str(round(omega_res,2)) + " GHz")
plt.legend(bbox_to_anchor=(1.05, -0.2))
plt.xlabel("Microwave detuning from $\omega(k)$ (GHz)")
plt.ylabel("Log Rabi population amplitude")
plt.xlim(0,10)
```

Out[20]:
```
(0.0, 10.0)
```



In [16]:
```
for k in np.linspace(np.pi/3,2*np.pi,10)[4:]:
    print(k)
    omega_res = 100*(1 - np.cos(k/2))
    out = run_expt(20, np.linspace(0,100,500), [[k,0.1]], 0.5,9, np.linspace(-5,5,11), 200, 0.5, np.linspace(omega_res - 2,omega_res +12,50))
    nv_dist_k_results[k] = [max([abs(out[i][j][0])**2 for j in range(len(out[i]))]) for i in range(len(out))]
```

```
3.3743032205223704
3.956079637853814
4.537856055185257
5.1196324725167
5.701408889848143
```

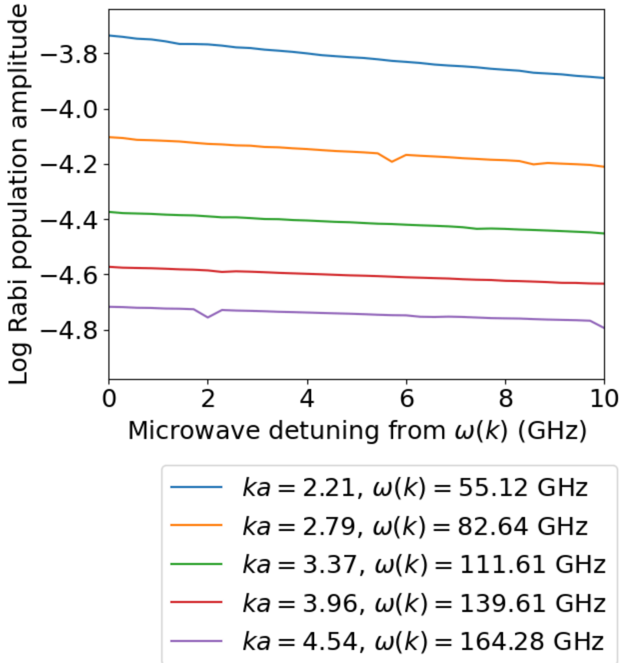
```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In [16], line 4
      2 print(k)
      3 omega_res = 100*(1 - np.cos(k/2))
----> 4 out = run_expt(20, np.linspace(0,100,500), [[k,0.1]], 0.5,9, np.linspace(-5,5,11), 200, 0.5, np.linspace(omega_res - 2,omega_res +12,50))
      5 nv_dist_k_results[k] = [max([abs(out[i][j][0])**2 for j in range(len(out[i]))]) for i in range(len(out))]

Cell In [3], line 70, in run_expt(Bext, times, modes, a, dist, indices, max_magnon, mw_rabi, mw_frequencies, output)
     67 H = [H0,[Sp,spc],[Sm,smc], [Sz, szc]]
     68 st = time.time()
---> 70 out = sesolve(H, nv_init, times,args={'mw_rabi':mw_rabi, 'mw_frequency':omega, 'Bext':Bext, 'a':a, 'dist':dist, 'indices':indices, 'modes':modes, 'max_magnon':max_magnon})
     71 end = time.time()
     72 args = {'mw_rabi':mw_rabi, 'mw_frequency':omega, 'Bext':Bext, 'a':a, 'dist':dist, 'indices':indices, 'modes':modes, 'max_magnon':max_magnon}

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\qutip\solver\sesolve.py:106, in sesolve(H, psi0, tlist, e_ops, args, options, **kwargs)
    104 H = QobjEvo(H, args=args, tlist=tlist)
    105 solver = SESolver(H, options=options)
--> 106 return solver.run(psi0, tlist, e_ops=e_ops)

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\qutip\solver\solver_base.py:158, in Solver.run(self, state0, tlist, args, e_ops)
    153 stats['preparation time'] += time() - _time_start
    155 progress_bar = progress_bars[self.options['progress_bar']](
    156     len(tlist)-1, **self.options['progress_kwargs']
    157 )
--> 158 for t, state in self._integrator.run(tlist):
    159     progress_bar.update()
    160     results.add(t, self._restore_state(state, copy=False))

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\qutip\solver\integrator\integrator.py:201, in Integrator.run(self, tlist)
    187 """
    188 Integrate the system yielding the state for each times in tlist.
    189 
   (...)
    198     The state of the solver at each ``t`` of tlist.
    199 """
    200 for t in tlist[1:]:
--> 201     yield self.integrate(t, False)

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\qutip\solver\integrator\scipy_integrator.py:110, in IntegratorScipyAdams.integrate(self, t, copy)
    108 self._check_handle()
    109 if t != self._ode_solver.t:
--> 110     self._ode_solver.integrate(t)
    111 return self.get_state(copy)

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\scipy\integrate\_ode.py:433, in ode.integrate(self, t, step, relax)
    430     mth = self._integrator.run
    432 try:
--> 433     self._y, self.t = mth(self.f, self.jac or (lambda: None),
    434                           self._y, self.t, t,
    435                           self.f_params, self.jac_params)
    436 except SystemError as e:
    437     # f2py issue with tuple returns, see ticket 1187.
    438     raise ValueError(
    439         'Function to integrate must not return a tuple.'
    440     ) from e

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\scipy\integrate\_ode.py:1009, in vode.run(self, f, jac, y0, t0, t1, f_params, jac_params)
   1005     jac = _vode_banded_jac_wrapper(jac, self.ml, jac_params)
   1007 args = ((f, jac, y0, t0, t1) + tuple(self.call_args) +
   1008         (f_params, jac_params))
-> 1009 y1, t, istate = self.runner(*args)
   1010 self.istate = istate
   1011 if istate < 0:

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\qutip\solver\integrator\scipy_integrator.py:69, in IntegratorScipyAdams._mul_np_vec(self, t, vec)
     67 state = _data.dense.fast_from_numpy(vec)
     68 column_unstack_dense(state, self._size, inplace=True)
---> 69 out = self.system.matmul_data(t, state)
     70 column_stack_dense(out, inplace=True)
     71 return out.as_ndarray().ravel()

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\qutip\core\cy\qobjevo.pyx:1099, in qutip.core.cy.qobjevo.QobjEvo.matmul_data()

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\qutip\core\cy\qobjevo.pyx:1111, in qutip.core.cy.qobjevo.QobjEvo.matmul_data()

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\qutip\core\cy\_element.pyx:167, in qutip.core.cy._element._BaseElement.matmul_data_t()

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\qutip\core\cy\_element.pyx:367, in qutip.core.cy._element._EvoElement.coeff()

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\qutip\core\cy\coefficient.pyx:220, in qutip.core.cy.coefficient.FunctionCoefficient.__call__()

Cell In [3], line 58, in run_expt.<locals>.szc(t, args)
     57 def szc(t, args):
---> 58     f = total_coupling_field(t, args['Bext'], args['a'], args['dist'], args['indices'], args['modes'], args['max_magnon'])
     59     return f[2]

Cell In [3], line 33, in run_expt.<locals>.total_coupling_field(t, Bext, a, dist, indices, modes, max_magnon)
     30 total_field = np.array([0.,0.,0.])
     32 for mode in modes:
---> 33     total_field += magnon_single_mode_field(Bext,t,mode[0],a,dist,mode[1],indices, max_magnon)
     35 return total_field

Cell In [3], line 25, in run_expt.<locals>.magnon_single_mode_field(Bext, t, k, a, dist, amplitude, indices, max_magnon)
     23     bloch_vector = np.array([amplitude*np.cos(index*k*a - omega*t),amplitude*np.sin(index*k*a - omega*t), np.sqrt(1-amplitude**2)])
     24     direction = np.array([-1*index*a, 0, dist])
---> 25     B_total += dipole_field(bloch_vector, direction)
     27 return B_total

Cell In [2], line 16, in dipole_field(bloch_vector, r)
     13 hbar = 1.054571817*pow(10,-34) #hbar SI
     14 mag = -1*muB*bloch_vector #magnetic moment of a single electron
---> 16 return muB * mu0/(4*np.pi) * 1/hbar * (3*np.dot(mag,direction)*direction - mag)/(pow(dist,3) * pow(10,-27)) * pow(10,-9)

File <__array_function__ internals>:180, in dot(*args, **kwargs)

KeyboardInterrupt: 
```

In [15]:
```
nv_dist_k_results.keys()
```

Out[15]: dict_keys([1.0471975511965976, 1.6289739685280409, 2.210750385859484, 2.7925268031909276])

In [19]:
```
for key in list(nv_dist_k_results.keys())[2:7]:
    omega_res = 100*(1 - np.cos(key/2))
    plt.plot(np.linspace(-2,12,50), [np.log10(i) for i in nv_dist_k_results[key]], label = "$ka = " + str(round(key,2))+ " $, $\omega(k) = $" + str(round(omega_res,2)) + " GHz")
plt.legend(bbox_to_anchor=(1.05, -0.2))
plt.xlabel("Microwave detuning from $\omega(k)$ (GHz)")
plt.ylabel("Log Rabi population amplitude")
plt.xlim(0,10)
```

Out[19]:
```
(0.0, 10.0)
```

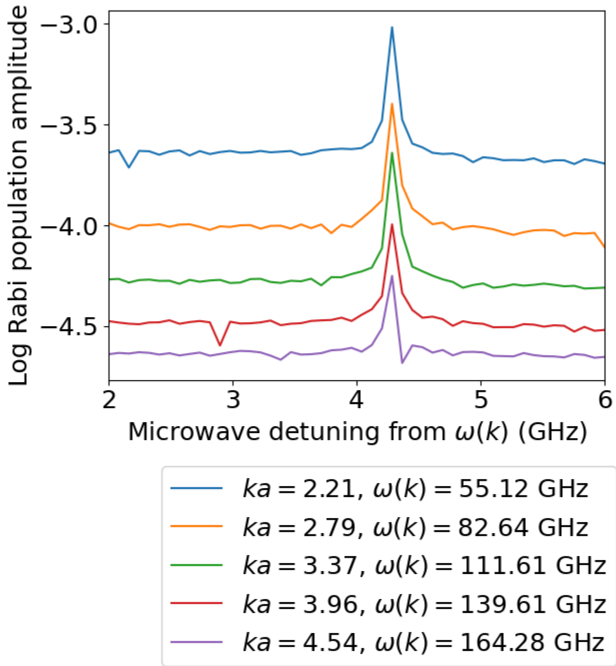


In [21]:
```
k_results_finer = {}
for k in np.linspace(np.pi/3,2*np.pi,10)[2:7]:
    print(k)
    omega_res = 100*(1 - np.cos(k/2))
    out = run_expt(20, np.linspace(0,100,500), [[k,0.1]], 0.5,0.5, np.linspace(-5,5,11), 200, 0.5, np.linspace(omega_res +2,omega_res +6,50))
    k_results_finer[k] = [max([abs(out[i][j][0])**2 for j in range(len(out[i]))]) for i in range(len(out))]
```

```
2.210750385859484
2.7925268031909276
3.3743032205223704
3.956079637853814
4.537856055185257
```

In [24]:
```
for key in list(k_results_finer.keys()):
    omega_res = 100*(1 - np.cos(key/2))
    plt.plot(np.linspace(2,6,50), [np.log10(i) for i in k_results_finer[key]], label = "$ka = " + str(round(key,2))+ " $, $\omega(k) = $" + str(round(omega_res,2)) + " GHz")
plt.legend(bbox_to_anchor=(1.05, -0.2))
plt.xlabel("Microwave detuning from $\omega(k)$ (GHz)")
plt.ylabel("Log Rabi population amplitude")
plt.xlim(2,6)
```

Out[24]:
```
(2.0, 6.0)
```

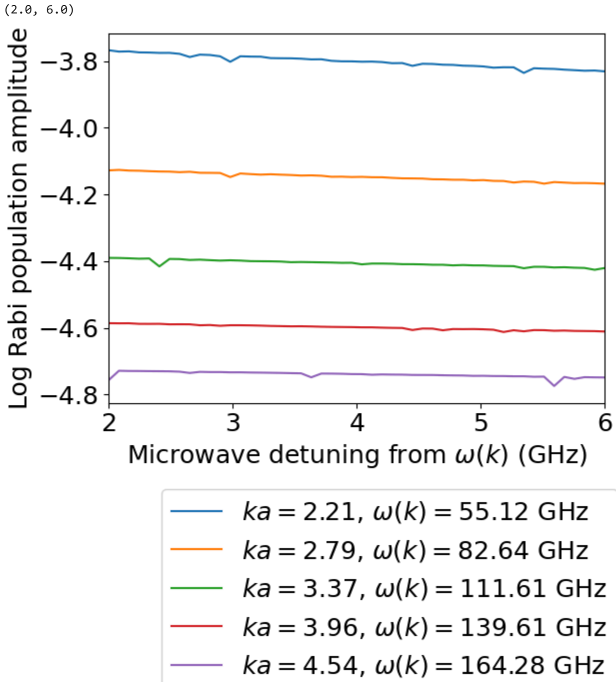


In [25]:
```
nv_dist_k_results_finer = {}
for k in np.linspace(np.pi/3,2*np.pi,10)[2:7]:
    print(k)
    omega_res = 100*(1 - np.cos(k/2))
    out = run_expt(20, np.linspace(0,100,500), [[k,0.1]], 0.5,9, np.linspace(-5,5,11), 200, 0.5, np.linspace(omega_res + 2,omega_res +6,50))
    nv_dist_k_results_finer[k] = [max([abs(out[i][j][0])**2 for j in range(len(out[i]))]) for i in range(len(out))]
```

```
2.210750385859484
2.7925268031909276
3.3743032205223704
3.956079637853814
4.537856055185257
```

In [26]:
```
for key in list(nv_dist_k_results_finer.keys()):
    omega_res = 100*(1 - np.cos(key/2))
    plt.plot(np.linspace(2,6,50), [np.log10(i) for i in nv_dist_k_results_finer[key]], label = "$ka = " + str(round(key,2))+ " $, $\omega(k) = $" + str(round(omega_res,2)) + " GHz")
plt.legend(bbox_to_anchor=(1.05, -0.2))
plt.xlabel("Microwave detuning from $\omega(k)$ (GHz)")
plt.ylabel("Log Rabi population amplitude")
plt.xlim(2,6)
```

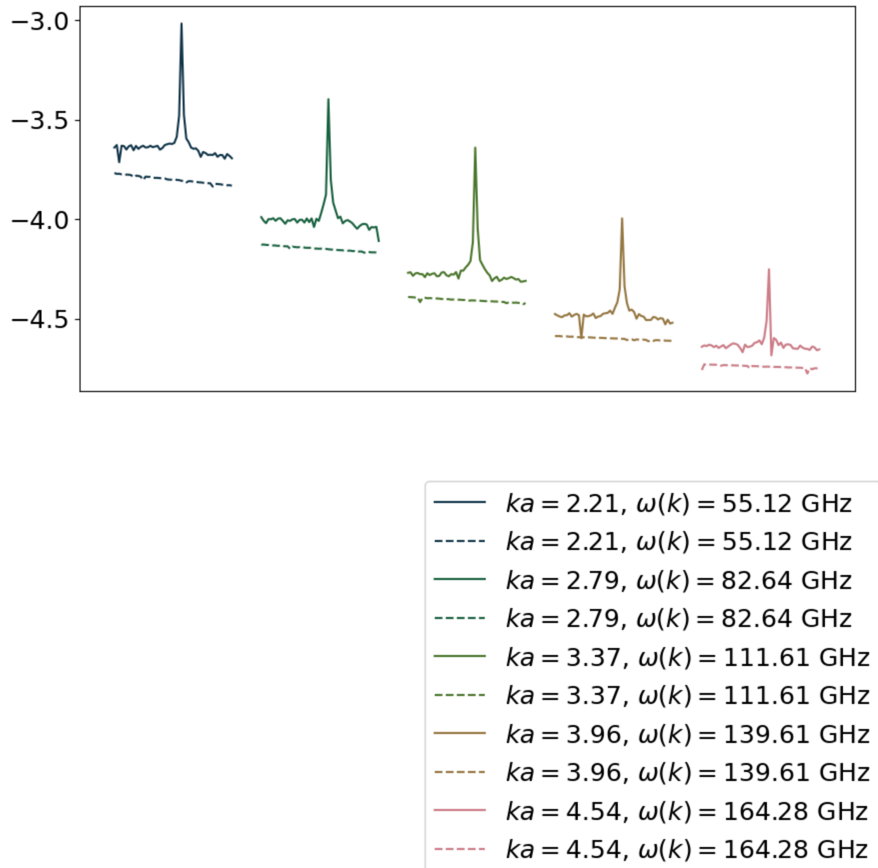
Out[26]: (2.0, 6.0)



In [40]:
```
fig,ax = plt.subplots(figsize = (10,5))
#plt.xlim(50,170)
ki = 0

cmap = plt.get_cmap('cubehelix')
my_colors = [cmap(int(i)) for i in np.linspace(50,150,5)]

for key in list(k_results_finer.keys()):
    omega_res = 100*(1 - np.cos(key/2))
```

```
    plt.plot(np.linspace(5*ki+2,5*ki+6,50), [np.log10(i) for i in k_results_finer[key]], label = "$ka = " + str(round(key,2))+ " $, $\omega(k) = $" + str(round(omega_res,2)) + " GHz",c
    plt.plot(np.linspace(5*ki+2,5*ki+6,50), [np.log10(i) for i in nv_dist_k_results_finer[key]], "--",label = "$ka = " + str(round(key,2))+ " $, $\omega(k) = $" + str(round(omega_res,2
    ki += 1
plt.legend(bbox_to_anchor=(1.05, -0.2))
plt.xticks([])
```

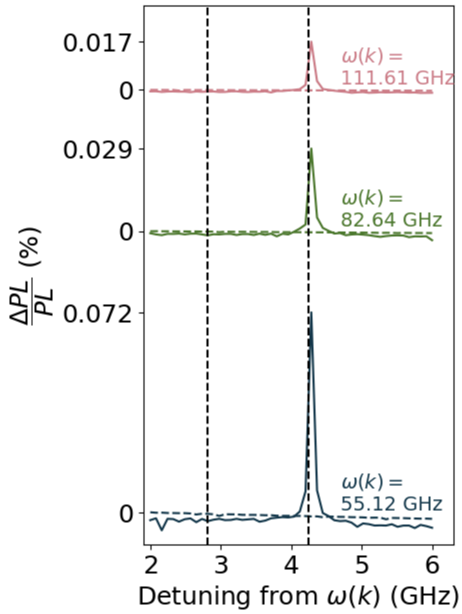
Out[40]: ([], [])



In [108…

```
fig,ax = plt.subplots(figsize = (4,7))

ki = 0

cmap = plt.get_cmap('cubehelix')
my_colors = [cmap(int(i)) for i in np.linspace(50,150,3)]
y_off_total = 0
curr_y_off = 0.1
yticks = []
ylabels = []
for key in list(k_results_finer.keys())[:3]:
    omega_res = 100*(1 - np.cos(key/2))
    mcc_dist_results = k_results_finer[key]
    mcc_contrast = [(i-np.mean(mcc_dist_results))/(1-np.mean(mcc_dist_results))*100 + y_off_total for i in mcc_dist_results]
    nv_dist_results = nv_dist_k_results_finer[key]
    nv_contrast = [(i-np.mean(nv_dist_results))/(1-np.mean(nv_dist_results))*100+ y_off_total for i in nv_dist_results]
    yticks.append(max(mcc_contrast)[0])
    yticks.append(max(nv_contrast)[0])
    ylabels.append(str(round(max(mcc_contrast)[0] - y_off_total,3)))
    ylabels.append("0")
    plt.text(4.7,y_off_total+0.002,"$\omega(k) = $" + "\n" + str(round(omega_res,2)) + " GHz",color = my_colors[ki],size = 14)
    y_off_total += curr_y_off
    curr_y_off = 0.05
    plt.plot(np.linspace(2,6,50), mcc_contrast, label = "$ka = " + str(round(key,2))+ " $, $\omega(k) = $" + str(round(omega_res,2)) + " GHz",color = my_colors[ki])
    plt.plot(np.linspace(2,6,50), nv_contrast, "--",label = "$ka = " + str(round(key,2))+ " $, $\omega(k) = $" + str(round(omega_res,2)) + " GHz",color = my_colors[ki])

    ki += 1
#plt.legend(bbox_to_anchor=(1.05, -0.2))
plt.xlim(1.9,6.3)
plt.xticks([2,3,4,5,6])
plt.yticks(yticks, labels=ylabels)
plt.ylabel("$\dfrac{\Delta PL}{PL}$ (%)")
plt.plot([2.81+1.43,2.81+1.43],[-0.1,0.18],"--",color = "black")
plt.plot([2.81,2.81],[-0.1,0.18],"--",color = "black")
plt.ylim([-0.01,0.18])
plt.xlabel("Detuning from $\omega(k)$ (GHz)")
```

Out[108]: Text(0.5, 0, 'Detuning from $\\omega(k)$ (GHz)')


0.017
0
$\omega(k) =$
111.61 GHz
0.029
0
$\omega(k) =$
82.64 GHz
$\frac{\Delta PL}{PL}$ (%)
0.072
$\omega(k) =$
55.12 GHz
0
2 3 4 5 6
Detuning from $\omega(k)$ (GHz)

In [ ]: