# S5_2019

# Demonstration and evaluation of localization mechanisms with Bluetooth Low Energy

## Prerequisites

### Software for desktop development

- SimpleLink CC13X2-26X2 SDK 3.20 or SimpleLink CC13X2-26X2 SDK 3.40
- Python 3.7 or higher

### Hardware

This module requires the following kits:

- 3x SimpleLink™ CC26x2R LaunchPad™
- 1x BOOSTXL-AoA

# Installation and basics

# How to install Simplelink-CC2640R2-SDK 3.20.00.21

Go to http://www.ti.com/tool/download/SIMPLELINK-CC2640R2-SDK/3.20.00.21

Then, download Linux Installer

Go to download folder in terminal and give permissions to file typing in terminal:

```
$ chmod +x simplelink_cc2640r2_sdk_3_20_00_21.run
```

Run in terminal

```
$ simplelink_cc2640r2_sdk_3_20_00_21.run
```

Look for the rtls_monitor tool located in :

> ti/simplelink_cc2640r2_sdk_3_20_00_21/tools/blestack/rtls_agent

Give permissions to file typing in terminal

```
$ chmod +x requirements.txt
```

Install the requirements in the *.txt file

```
$ pip3 install -r requirements.txt
```

> If error with windows-curses [windows-curses is not necessary on Python 3.7] comment such line.

Once Install virtualenv typing :

```
$ sudo pip3 install virtualenv
```

Execute virtualenv typing :

```
$ virtualenv -p python3 .venv
#or you can use
$ python3 -m venv .venv
```

Activate virtualenv typing :

```
$ . .venv/bin/activate
#or you can use
$ source .venv/bin/activate which does exactly the same thing
```

Give permissions to file typing in terminal :

```
$ chmod +x rtls_agent_cli
```

Now you can run the RTLS Agent

```
$ python3 agent/rtls_agent_cli.py
```

# To run again the virtual environment

Execute virtualenv typing :

```
$ virtualenv -p python3 .venv
#or you can use
$ python3 -m venv .venv
```

Activate virtualenv typing :

```
$ . .venv/bin/activate
#or you can use
$ source .venv/bin/activate which does exactly the same thing
```

Give permissions to file typing in terminal :

```
$ chmod +x rtls_agent_cli
```

Now you can run the AoA agent

```
$ python3 rtls_example.py
```

# How to install Simplelink-CC2640R2-SDK 3.40.00.10

Go to http://www.ti.com/tool/download/SIMPLELINK-CC2640R2-SDK/3.20.00.21

Then, download Linux Installer

Go to download folder in terminal and give permissions to file typing in terminal :

```
$ chmod +x simplelink_cc2640r2_sdk_3_40_00_10.run
```

Run in terminal

```
$ simplelink_cc2640r2_sdk_3_40_00_10.run
```

Look for the rtls_agent directory :

> /ti/simplelink_cc2640r2_sdk_3_40_00_21/tools/blestack/rtls_agent

Give permissions to file typing in terminal

```
$ chmod +x package.sh
```

File package.sh has been created as a script for Windows env and then ported over to run on a Unix environment so we need to type :

```
$ sed -i -e 's/\r$//' package.sh
```

Run de package file :

```
$ ./package.sh -c -b -u -i
```

Go to *rtls_ui* directory :

> ti/simplelink_cc2640r2_sdk_3_40_00_10/tools/blestack/rtls_agent/rtls_ui

Give permissions to file typing in terminal :

```
$ chmod +x rtls_ui
```

Run the RTLS Graphic Interface :

```
$ ./rtls_ui
```

# Flashing CC2640R2 in AoA mode

# Instructions

**Reminder**

- For the case of our cards :
- Passive  <L5000IZO>
- Master  <L5000J02>
- Slave  <L5000IZP>

  This is a tutorial to activate AOA in Code Composer Studio using LAUNCH CC2640R2 cards

In the master and slave : right click on the *_app project and then in properties

- In <Build/XDCtools> Make sure the XDC TOOL is 3.20.0.21

In the master and slave : double click on the *.ccxml file <app project/targetConfigs/CC2640R2F.ccxm>

- Decrease the JTAG operating frequency to 2.5 MHz DECREASE THE JTAG OPERATING FREQUENCY TO 2.5 MHz

In the master and slave : go to TOOLS folder <app project/TOOLS>

- Double click on <build_config.opt>
- Go to the end of the file and uncomment the line defining : <RTLS_LOCATIONING_AOA>
- Save and flash

In the passive : right click on the *_app project and then in properties

- In <Build/XDCtools> Make sure the XDC TOOL is 3.20.0.21

Double click on the *.ccxml file <app project/targetConfigs/CC2640R2F.ccxm>

- Decrease the JTAG operating frequency to 2.5 MHz

Go to project properties (Right click)

- <app project/CCS Build/ARM Compiler/Predefined Symbols>
- Click add and type <RTLS_LOCATIONING_AOA>
- Apply and close
- Save and flash

**AOA Format (RTLS)**

- {"originator": "Nwp", "type": "AsyncReq", "subsystem": "RTLS", "command": "RTLS_CMD_AOA_RESULT_ANGLE", "payload": {"angle": -59, "rssi": -47, "antenna": 2, "channel": 22}}

**AOA Format (our codes)**

- "payload": {"angle": -19, "rssi": -66, "antenna": 2, "channel": 26, "time": 1581951429.8088975, "distance": "4", "position": "-15"}

# RTLS using SDK 3.20

# Instructions

**Prerequisites**

- Follow the instructions for installing SDK 3.20
- Flash the Passive, Master and Slave
- Python 3.7 or higher

# Codes

# rtls_AoA.py

rtls_AoA.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Nov 16 17:47:15 2019

@author: Pablo Vinicio GONZALEZ RODRIGUEZ
@reference: rtls_example.py from Texas Instruments
"""
import argparse
import json
import queue
import time
from rtls import RTLSManager, RTLSNode
import multiprocessing

# Un-comment the below to get raw serial transaction logs
# import logging, sys
# logging.basicConfig(stream=sys.stdout, level=logging.DEBUG,
#                     format='[%(asctime)s] {%(filename)s:%(lineno)d} %(le

# Un-comment the below to get raw serial transaction logs
# import logging, sys
# logging.basicConfig(stream=sys.stdout, level=logging.DEBUG,
#                     format='[%(asctime)s] {%(filename)s:%(lineno)d} %(le

master_node = None
passive_nodes= []
address = None
address_type = None
timestamp = 0

parser = argparse.ArgumentParser(description='Returns the distance between
    'Returns the position of the slave relative to the passive. ' +
    'Example:    test.py \'distance\' , \'position\' and \'duration\'\''+
    'These samples were taken using Python\'')
parser.add_argument("distance", help="Displays distance between the passiv
parser.add_argument("position", help="Displays angle between the passive a
parser.add_argument("duration", help="Test duration in minutes")
args = parser.parse_args()

def run_forever():
    # Initialize, but don't start RTLS Nodes to give to the RTLSManager
    my_nodes = [RTLSNode('/dev/ttyACM0', 115200), RTLSNode('/dev/ttyACM2',
```

```
44
45      # Initialize references to the connected devices
46      master_node = None
47      passive_nodes = []
48      # Initialize references to the connected devices
49      address = None
50      address_type = None
51
52      # AoA related settings
53      aoa_run_mode = 'AOA_MODE_ANGLE'
54      aoa_cte_scan_ovs = 4
55      aoa_cte_offset = 4
56      aoa_cte_time = 20
57
58      # Auto detect AoA or ToF support related
59      aoa_supported = False
60
61      # If slave addr is None, the script will connect to the first RTLS sla
62      # that it found. If you wish to connect to a specific device
63      # (in the case of multiple RTLS slaves) then you may specify the addre
64      # explicitly as given in the comment to the right
65      slave_addr = None  # '54:6C:0E:83:3F:3D'
66
67      # Scan list
68      scanResultList = list()
69
70
71      # connection interval in units of 1.25msec configured in connection re
72      conn_interval = 80
73
74      # Initialize manager reference, because on Exception we need to stop t
75      manager = None
76      try:
77          # Start an RTLSManager instance without WebSocket server enabled
78          manager = RTLSManager(my_nodes, websocket_port=None)
79          # Create a subscriber object for RTLSManager messages
80          subscriber = manager.create_subscriber()
81          # Tell the manager to automatically distribute connection paramete
82          manager.auto_params = True
83          # Start RTLS Node threads, Serial threads, and manager thread
84          manager.start()
85
86          # Wait until nodes have responded to automatic identify command an
87          # to single master RTLSNode and list of passive RTLSNode instances
88          master_node, passive_nodes, failed = manager.wait_identified()
89
90          if len(failed):
91              print(f"ERROR: {len(failed)} nodes could not be identified. Ar
92
93          # Exit if no master node exists
94          if not master_node:
```

```
95              raise RuntimeError("No RTLS Master node connected")
96
97          # Combined list for lookup
98          all_nodes = passive_nodes + [master_node]
99
100         # Initialize application variables on nodes
101         for node in all_nodes:
102             node.seed_initialized = False
103             node.aoa_initialized = False
104
105         #
106         # At this point the connected devices are initialized and ready
107         #
108
109         # Display list of connected devices and their capabilities
110         print(f"{master_node.identifier} {', '.join([cap for cap, availabl
111
112         # Iterate over Passives and detect their capabilities
113         for pn in passive_nodes:
114             print(f"{pn.identifier} {', '.join([cap for cap, available in
115
116         # Check over aggregated capabilities to see if they make sense
117         capabilities_per_node = [[cap for cap, avail in node.capabilities.
118
119         # Assume AoA if all nodes are not ToF
120         aoa_supported = all(not ('TOF_PASSIVE' in node_caps or 'TOF_MASTER
121
122         # Check that Nodes all must be AoA
123         if not (aoa_supported):
124             raise RuntimeError("All nodes must be AoA")
125
126         # Need at least 1 passive for AoA
127         if aoa_supported and len(passive_nodes) == 0:
128             raise RuntimeError('Need at least 1 passive for AoA')
129
130         # Send an example command to each of them, from commands listed at
131         for n in all_nodes:
132             n.rtls.identify()
133
134         while True:
135             # Get messages from manager
136             try:
137                 identifier, msg_pri, msg = subscriber.pend(block=True, tim
138
139                 # Get reference to RTLSNode based on identifier in message
140                 sending_node = manager[identifier]
141
142                 if sending_node in passive_nodes:
143                     print(f"PASSIVE: {identifier} --> {msg.as_json()}")
144                 else:
145                     print(f"MASTER: {identifier} --> {msg.as_json()}")
```

```
146
147                 # If we received an error, print it.
148                 if msg.command == 'RTLS_EVT_ERROR':
149                     print(f"Received RTLS_EVT_ERROR with status: {msg.payl
150
151                 # If we received an assert, print it.
152                 if msg.command == 'RTLS_EVT_ASSERT' and msg.type == 'Async
153                     raise RuntimeError(f"Received HCI H/W Assert with code
154
155
156                 # After identify is received, we start scanning
157                 if msg.command == 'RTLS_CMD_IDENTIFY':
158                     master_node.rtls.scan()
159
160                 # Once we start scaning, we will save the address of the
161                 # last scan response
162                 if msg.command == 'RTLS_CMD_SCAN' and msg.type == 'AsyncRe
163                     # Slave address none means that we connect to any slav
164                     if slave_addr is None:
165                         address = msg.payload.addr
166                         address_type = msg.payload.addrType
167                     else:
168                         scanResultList.append(msg.payload.addr)
169                         scanResultList.append(msg.payload.addrType)
170
171                 # Once the scan has stopped and we have a valid address, t
172                 # connect
173                 if msg.command == 'RTLS_CMD_SCAN_STOP':
174                     if slave_addr is None:
175                         if address is not None and address_type is not Non
176                             master_node.rtls.connect(address_type, address
177                     elif slave_addr in scanResultList:
178                         i = scanResultList.index(slave_addr)
179                         master_node.rtls.connect(scanResultList[i + 1], sc
180                         scanResultList.clear()
181                     else:
182                         # If we didn't find the device, keep scanning.
183                         master_node.rtls.scan()
184
185                 # Once we are connected, then we can do stuff
186                 if msg.command == 'RTLS_CMD_CONNECT' and msg.type == 'Asyn
187                     if msg.payload.status == 'RTLS_SUCCESS':
188                         # Find the role based on capabilities of sending n
189                         role = 'AOA_MASTER' if sending_node.capabilities.g
190                         # Send AoA params
191                         sending_node.rtls.aoa_set_params(role, aoa_run_mod
192                                                          aoa_cte_scan_o
193                                                          aoa_cte_offset
194                                                          aoa_cte_time)
195                     else:
196                         # If the connection failed, keep scanning
```

```
                        master_node.rtls.scan()

                # Count the number of nodes that have ToF initialized
                if msg.command == 'RTLS_CMD_AOA_SET_PARAMS' and msg.payloa
                    sending_node.aoa_initialized = True
                    if all([n.aoa_initialized for n in all_nodes]):
                        # Start AoA on the master and passive nodes
                        for node in all_nodes:
                            node.rtls.aoa_start(True)

                # Wait for security seed
                if msg.command == 'RTLS_CMD_TOF_GET_SEC_SEED' and msg.payl
                    seed = msg.payload.seed
                    for node in passive_nodes:
                        node.rtls.tof_set_sec_seed(seed)

                # Wait until passives have security seed set
                if msg.command == 'RTLS_CMD_TOF_SET_SEC_SEED' and msg.payl
                    sending_node.seed_initialized = True

                try:
                    with open("Output/test.json","a") as fichier:
                        message=json.loads("["+msg.as_json()+"]")
                        message[0]["payload"]["time"]=timestamp
                        message[0]["payload"]["distance"]=args.distance
                        message[0]["payload"]["position"]=args.position
                        json.dump(message[0]["payload"], fichier)
                        print(message[0]["payload"])
                        fichier.write('\n')
                        fichier.close
                except:
                    print ('\nProblem writing on JSON file')
                    break

            except queue.Empty:
                pass

    finally:
        if manager:
            manager.stop()

if __name__ == '__main__':
    #Start RTLS as a process
    p=multiprocessing.Process(target=run_forever, name="Run_Forever")
    p.start()

    #Wait "t" minutes
    time.sleep(60*int(args.duration))

    #Terminate RTLS
    p.terminate()
```

```
248     print("\n\n RTLS AoA scan stopped after "+args.duration+" minutes \n")
249 
250     p.join()
251 
```

# exportBLETexas.py

rtls_AoA.py

```
import json
import psycopg2
import logging #Provides a logging system
import pandas as pd
import matplotlib.pyplot as plt

plt.rcParams['font.size'] = 20

class AOAData():
    position=[] #Actual angle where the slave is placed (which mark on the
    distance=[] #Distance from the passive to slave
    time=[]
    angle=[] #Measured angle (by the passive)
    rssi=[]
    antenna=[]
    channel=[]

def readData():
    mes= AOAData()
    mes1= AOAData()
    mes2= AOAData()

    try:
        #with open('/home/efisio/Documents/3Sem/S5_2019/Data/data.json') a
        with open('/home/efisio/Documents/3Sem/S5_2019/Data/dataBLE9.json'
            mes.distance=[]
            mes1.distance=[]
            mes1.position=[]
            mes1.angle=[]
            mes2.distance=[]
            mes2.position=[]
            mes2.angle=[]

            #READ THE RAW DATA

            for entry in f:
                dict = json.loads(entry)
                mes.position.append(dict.get('position', 0))
                mes.distance.append(dict.get('distance', 0))
                mes.time.append(dict.get('time', 0))
                mes.angle.append(dict.get('angle', 0))
                mes.rssi.append(dict.get('rssi', 0))
                mes.antenna.append(dict.get('antenna', 0))
```

```
                mes.channel.append(dict.get('channel', 0))
                #myList.append(mes.__dict__)

            #SEPARATE THE RANGE TESTS (BELOW 5 METERS) FROM THE REGULAR TE

            for i, line in enumerate(mes.distance):
                if int(line )< 5:

                    mes1.position.append(mes.position[i])
                    mes1.distance.append(mes.distance[i])
                    mes1.time.append(mes.time[i])
                    mes1.angle.append(mes.angle[i])
                    mes1.rssi.append(mes.rssi[i])
                    mes1.antenna.append(mes.antenna[i])
                    mes1.channel.append(mes.channel[i])
                    #print(mes.distance)

                else:
                    mes2.position.append(mes.position[i])
                    mes2.distance.append(mes.distance[i])
                    mes2.time.append(mes.time[i])
                    mes2.angle.append(mes.angle[i])
                    mes2.rssi.append(mes.rssi[i])
                    mes2.antenna.append(mes.antenna[i])
                    mes2.channel.append(mes.channel[i])

    except (Exception, psycopg2.Error) as error :
            logging.debug('Error while reading file or inserting to Postgr
            print ("Error while reading file or inserting to PostgreSQL",
    return mes1,mes2

def pointed_graph(mes):
    angle_table=pd.DataFrame({'position' : mes.position,'distance' : mes.d

    fig = plt.figure(figsize=(20,10))
    ax = plt.subplot(111)

    #PLOT ANGLE VS READ POSITION

    ax.plot(angle_table['position'], angle_table['angle'],'r+',lw=10, mark

    plt.grid(True, linestyle='-', linewidth=2)
    plt.title('Position réelle vs position mesurée',loc='right',fontsize=2
    plt.xlabel('Position réelle',fontsize=25)
    plt.ylabel('Position mesurée',fontsize=25)

    fig.tight_layout()
    fig.savefig('//home/efisio/Documents/3Sem/S5_2019/Data/position_vs_mes

def boxplot_graphs(mes):

```

```
95      fig = plt.figure(figsize=(20,10))
96      ax = plt.subplot(111)
97      print (len(mes.distance))
98      print (len(mes.position))
99      print (len(mes.angle))
100     angle_table=pd.DataFrame({'position' : mes.position,'distance' : mes.d
101
102     ax.boxplot(angle_table.loc[angle_table['position']=='-90']['angle'], p
103     ax.boxplot(angle_table.loc[angle_table['position']=='-75']['angle'], p
104     ax.boxplot(angle_table.loc[angle_table['position']=='-60']['angle'], p
105     ax.boxplot(angle_table.loc[angle_table['position']=='-45']['angle'], p
106     ax.boxplot(angle_table.loc[angle_table['position']=='-30']['angle'], p
107     ax.boxplot(angle_table.loc[angle_table['position']=='-15']['angle'], p
108     ax.boxplot(angle_table.loc[angle_table['position']=='0']['angle'], pos
109     ax.boxplot(angle_table.loc[angle_table['position']=='15']['angle'], po
110     ax.boxplot(angle_table.loc[angle_table['position']=='30']['angle'], po
111     ax.boxplot(angle_table.loc[angle_table['position']=='45']['angle'], po
112     ax.boxplot(angle_table.loc[angle_table['position']=='60']['angle'], po
113     ax.boxplot(angle_table.loc[angle_table['position']=='75']['angle'], po
114     ax.boxplot(angle_table.loc[angle_table['position']=='90']['angle'], po
115
116     ax.tick_params(direction='out', length=6, width=2, grid_alpha=0.5)
117     plt.xticks([1,2,3,4,5,6,7,8,9,10,11,12,13], ['-90','-75','-60','-45','
118     plt.yticks([-90,-75,-60,-45,-30,-15,0,15,30,45,60,75,90])
119
120     plt.grid(True, linestyle='-', linewidth=2)
121     #plt.title('Position réelle vs position mesurée',loc='right',fontsize=
122     #plt.xlabel('Position réelle (en degrés)',fontsize=25)
123     #plt.ylabel('Position mesurée',fontsize=25)
124
125     fig.tight_layout()
126     fig.savefig('//home/efisio/Documents/3Sem/S5_2019/Data/position_vs_mes
127
128 def range_graph(mes):
129     dist1=[]
130     angle1=[]
131     dist2=[]
132     angle2=[]
133     fig = plt.figure(figsize=(20,10))
134     ax = plt.subplot(111)
135
136     for i,line in enumerate(mes.position):
137         if line == '-30':
138             dist1.append(mes.distance[i])
139             angle1.append(mes.angle[i])
140         if line == '0':
141             dist2.append(mes.distance[i])
142             angle2.append(mes.angle[i])
143     angle_table1=pd.DataFrame({'position' : -30,'distance' : dist1,'angle'
144     angle_table2=pd.DataFrame({'position' : 0,'distance' : dist2,'angle' :
145
```

```
    ax.boxplot(angle_table1.loc[angle_table1['distance']=='5']['angle'], p
    ax.boxplot(angle_table1.loc[angle_table1['distance']=='10']['angle'],
    ax.boxplot(angle_table1.loc[angle_table1['distance']=='15']['angle'],
    ax.boxplot(angle_table1.loc[angle_table1['distance']=='16']['angle'],
    #ax.boxplot(angle_table1.loc[angle_table1['distance']=='17']['angle'],
    ax.boxplot(angle_table1.loc[angle_table1['distance']=='18']['angle'],
    ax.boxplot(angle_table1.loc[angle_table1['distance']=='19']['angle'],
    ax.boxplot(angle_table1.loc[angle_table1['distance']=='20']['angle'],

    plt.xticks([1, 2,3,4,5,6,7], ['5','10','15','16','18','19','20'])

    plt.grid(True, linestyle='-', linewidth=2)
    plt.title('Position mesurée vs distance (à -30 degrées)',loc='right',f
    plt.xlabel('Distance (en mètres)',fontsize=25)
    plt.ylabel('Position mesurée (en degrés)',fontsize=25)

    fig.tight_layout()
    fig.savefig('//home/efisio/Documents/3Sem/S5_2019/Data/mesure_vs_dista

    fig2 = plt.figure(figsize=(20,10))
    bx = plt.subplot(111)

    bx.boxplot(angle_table2.loc[angle_table2['distance']=='5']['angle'], p
    bx.boxplot(angle_table2.loc[angle_table2['distance']=='10']['angle'],
    bx.boxplot(angle_table2.loc[angle_table2['distance']=='15']['angle'],
    bx.boxplot(angle_table2.loc[angle_table2['distance']=='17']['angle'],
    bx.boxplot(angle_table2.loc[angle_table2['distance']=='19']['angle'],

    plt.xticks([1, 2,3,4,5], ['5','10','15','17','19'])

    plt.grid(True, linestyle='-', linewidth=2)
    plt.title('Position mesurée vs distance (à 0 degrées)',loc='right',fon
    plt.xlabel('Distance (en mètres)',fontsize=25)
    plt.ylabel('Position mesurée (en degrés)',fontsize=25)

    fig2.tight_layout()
    fig2.savefig('//home/efisio/Documents/3Sem/S5_2019/Data/mesure_vs_dist

def path_graph(mes,path):
    mes.time=mes.time[0:len(mes.angle)]
    times=[ float(x)-float(mes.time[1]) for x in mes.time ] #used to norma


    angle_table=pd.DataFrame({'time' : times,'distance' : mes.distance,'an
    angle_table=pd.DataFrame({'time' : angle_table.loc[angle_table['distan

    fig = plt.figure(figsize=(20,10))
    ax = plt.subplot(111)
    plt.title('Position mesurée vs temps',loc='right',fontsize=25)
    plt.xlabel('Temps (secondes)',fontsize=25)
    plt.ylabel('Position mesurée (en degrées)',fontsize=25)
```

```
    angle_table.plot()
    ax.plot(angle_table['time'], angle_table['angle'],'r+',lw=10, markersi
    ax.grid(True, linestyle='-', linewidth=2)

    fig.tight_layout()
    fig.savefig('//home/efisio/Documents/3Sem/S5_2019/Data/mesure_vs_temps

def get_sem(mes,test):
    fig = plt.figure(figsize=(20,10))
    cx = plt.subplot(111)
    print (len(mes.distance))
    print (len(mes.position))
    print (len(mes.angle))
    angle_table=pd.DataFrame({'position' : mes.position,'distance' : mes.d
    errors=[]

    for i in range(0,14):
        errors.extend([angle_table.loc[angle_table['position']==str(-90+(1

    cx.errorbar(angle_table.loc[angle_table['position']=='-90']['position'
    cx.errorbar(angle_table.loc[angle_table['position']=='-75']['position'
    cx.errorbar(angle_table.loc[angle_table['position']=='-60']['position'
    cx.errorbar(angle_table.loc[angle_table['position']=='-45']['position'
    cx.errorbar(angle_table.loc[angle_table['position']=='-30']['position'
    cx.errorbar(angle_table.loc[angle_table['position']=='-15']['position'
    cx.errorbar(angle_table.loc[angle_table['position']=='0']['position'],
    cx.errorbar(angle_table.loc[angle_table['position']=='15']['position']
    cx.errorbar(angle_table.loc[angle_table['position']=='30']['position']
    cx.errorbar(angle_table.loc[angle_table['position']=='45']['position']
    cx.errorbar(angle_table.loc[angle_table['position']=='60']['position']
    cx.errorbar(angle_table.loc[angle_table['position']=='75']['position']
    cx.errorbar(angle_table.loc[angle_table['position']=='90']['position']

    cx.tick_params(direction='out', length=6, width=2, grid_alpha=0.5)
    plt.xticks([0,1,2,3,4,5,6,7,8,9,10,11,12,13], ['-90','-75','-60','-45'
    plt.yticks([-90,-75,-60,-45,-30,-15,0,15,30,45,60,75,90])

    plt.grid(True, linestyle='-', linewidth=2)

    fig.tight_layout()
    fig.savefig('//home/efisio/Documents/3Sem/S5_2019/Data/errors_test_'+t
    print ("|  Position\t  |\t Error     |")
    print ("_____________________________________")
    for i in range(0,13):
        print (str(-90+(15*(i)))+"\t"+"%.5f"% errors[i])
        #print ("|"+str(-90+(15*(i)))+"\t\t  |\t "+"%.5f"% errors[i]+ "

def main():
    myList1,myList2=readData()
    #pointed_graph(myList1) #Use myList2 for dataBLE5 measures
    #boxplot_graphs(myList1) #Use myList2 for dataBLE5 measures
```

```
248         get_sem(myList1,'9') #Use myList2 for dataBLE5 measures
249         #range_graph(myList1,'4') #Use myList2 for dataBLE5 measures
250         #print(myList1.angle)
251         print('Done')
252 
253     if __name__ == "__main__":
254         main()
```