MURP_Table2

June 20, 2024

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import sympy as S
from tqdm.auto import tqdm
import pandas as pd
```

[2]:
```python
x, y, z, rx, ry, rz = S.symbols(r'x y z \alpha \beta \gamma')
L = S.symbols('L')
```

[3]:
```python
rx_mat = S.Matrix([
    [1, 0, 0],
    [0, S.cos(rx), -S.sin(rx)],
    [0, S.sin(rx), S.cos(rx)]
])
```

[4]:
```python
ry_mat = S.Matrix([
    [S.cos(ry), 0, S.sin(ry)],
    [0, 1, 0],
    [-S.sin(ry), 0, S.cos(ry)]
])
```

[5]:
```python
rz_mat = S.Matrix([
    [S.cos(rz), -S.sin(rz), 0],
    [S.sin(rz), S.cos(rz), 0],
    [0, 0, 1]
])
```

[6]:
```python
rot_mat = rz_mat @ ry_mat @ rx_mat
```

[7]:
```python
trans_vec = S.Matrix([x, y, z])
```

[8]:
```python
T = S.Matrix(np.zeros((4,4)))
T[0:3,0:3] = rot_mat
T[0:3,3] = trans_vec
T[3,3] = 1
```

[9]:
```
def begin_homogeneous(M):
    assert len(M.shape) == 2
    R,C = M.shape
    assert R == 3
    res = np.ones((4,C))
    res = S.Matrix(res)
    res[0:3,:] = M[:,:]
    return res
```

[10]:
```
def end_homogeneous(M):
    assert len(M.shape) == 2
    R,C = M.shape
    assert R == 4
    res = M[0:3,:]
    return res
```

[11]:
```
def mk_planar_antenna_matrix(COUNT,R=1):
    SEP_RAD = 2*np.pi/COUNT
    ant = [[R*np.cos(SEP_RAD/2 + i*SEP_RAD), R*np.sin(SEP_RAD/2 + i*SEP_RAD),␣
↪0] for i in range(A_ANT_COUNT)]
    ant = S.Matrix(ant).T
    assert ant.shape == (3,COUNT)
    return ant
```

[12]:
```
def dist_matrix(A, B):
    A_c = A.cols
    B_c = B.cols

    d = S.Matrix(np.zeros((A_c,B_c)))

    for i in range(A_c):
        for j in range(B_c):
            delta = A.col(i) - B.col(j)
            dot = delta.T @ delta
            d[i,j] = S.sqrt(dot[0])

    return d
```

[13]:
```
A_ANT_COUNT = 6
A_ant = mk_planar_antenna_matrix(A_ANT_COUNT,R=L)

B_ANT_COUNT = 6
B_ant = mk_planar_antenna_matrix(B_ANT_COUNT,R=L)
```

[14]:
```
dist_matrix(A_ant, B_ant)
```

[14]:

$$\left[\begin{matrix}0 & 1.0\sqrt{L^{2}} & 1.73205080756888\sqrt{L^{2}} & 2.0\sqrt{L^{2}} & 1.73205080756\\1.0\sqrt{L^{2}} & 0 & 1.0\sqrt{L^{2}} & 1.73205080756888\sqrt{L^{2}} & 2.0\sqrt{L}\\1.73205080756888\sqrt{L^{2}} & 1.0\sqrt{L^{2}} & 0 & 1.0\sqrt{L^{2}} & 1.73205080756\\2.0\sqrt{L^{2}} & 1.73205080756888\sqrt{L^{2}} & 1.0\sqrt{L^{2}} & 0 & 1.0\sqrt{L}\\1.73205080756888\sqrt{L^{2}} & 2.0\sqrt{L^{2}} & 1.73205080756888\sqrt{L^{2}} & 1.0\sqrt{L^{2}} & 0\\1.0\sqrt{L^{2}} & 1.73205080756888\sqrt{L^{2}} & 2.0\sqrt{L^{2}} & 1.73205080756888\sqrt{L^{2}} & 1.0\sqrt{L}\end{matrix}\right.$$

[15]:
```
A = A_ant
B = end_homogeneous(T @ begin_homogeneous(B_ant))
D = dist_matrix(A, B)
```

[16]:
```
meas_uwb = D.reshape(A_ANT_COUNT * B_ANT_COUNT,1)
meas_uwb_alt = meas_uwb.row_insert(36,S.Matrix([[z]]))
```

[17]:
```
subs = {
    rx: 0,
    ry: 0,
    rz: 0,
    L: .32,
}

meas_uwb = meas_uwb.subs(subs)
meas_uwb_alt = meas_uwb_alt.subs(subs)
```

[18]:
```
def mk_A(meas,states):
    A = meas.jacobian(states)

    # A has row = number of measurements
    # A has col = number of derivatives states
    R,C = A.shape
    assert R in [A_ANT_COUNT * B_ANT_COUNT, A_ANT_COUNT * B_ANT_COUNT + 1]
    assert C == len(states)

    return A
```

1 Table

[19]:
```
def add_std(a_std,b_std):
    return np.linalg.norm([a_std,b_std])

def bound_to_std(pm_a,std=2):
    return pm_a/std
```

[20]:
```
STD_UWB = .24
ALT_MEAS_STD = bound_to_std(.04,std=1)
ALT_BOUND_STD = bound_to_std(1,std=1)
```

[21]:
```
def var_zx(subs,std_alt):
    VAR_UWB = STD_UWB ** 2
    VAR_ALT = std_alt ** 2

    Sigma = np.diag([VAR_UWB]*36 + [VAR_ALT])
    A = mk_A(meas_uwb_alt,[x,y,z])
    info = np.linalg.inv(Sigma)
    G = A.T @ info @ A
    G_subs = G.subs(subs)
    G_arr = np.array(G_subs,dtype=np.float64)
    G_inv = np.linalg.inv(G_arr)
    cov = G_inv
    vx,vy,vz = cov[0,0],cov[1,1],cov[2,2]
    return vx,vy,vz

var_zx({x:5,y:0,z:0},.1)
```

[21]:
```
(0.0016065806611703791, 0.3906186614565446, 0.010000000000000002)
```

[22]:
```
NAMES = ['unconstrained', 'extemely concervative (local)', 'very concervative␣
 ↪(local)', 'concervative (local)', 'local meas only (proposed)', 'shared␣
 ↪measurements']
SIGMAS = [
    1000, # large number to allow calculation, technically should be np.inf
    add_std(.04,2),
    add_std(.04,1),
    add_std(.04,.2),
    add_std(.04,.1),
    add_std(.04,.04)
]

xs = [0,1,2.5,5,10,25] #,50]
```

[23]:
```
def mk_data_arr(wrt):
    SUBS_BASE = {x:5,y:0,z:1}
    WRT = wrt

    ys = [[] for i in range(len(SIGMAS))]

    for i in tqdm(xs):
        subs = SUBS_BASE.copy()
        subs[WRT] = i

        for j,sigma in enumerate(SIGMAS):
            ys[j].append(var_zx(subs,sigma))

    return np.array(ys)
```

[24]:
```
def mk_hvp(arr):
    h = np.linalg.norm(arr[:,:,0:2],axis=2)
    v = np.linalg.norm(arr[:,:,2:],axis=2)
    p = np.linalg.norm(arr,axis=2)
    h[h > 1e5] = np.infty # over threshold -> inf
    v[v > 1e5] = np.infty
    p[p > 1e5] = np.infty
    return np.round(h,2),np.round(v,2),np.round(p,2)
```

2 Compute HDOP, VDOP, PDOP for varying x at $(x, 0, 1)$

[25]:
```
arr_x = mk_data_arr(x)
arr_xh, arr_xv, arr_xp = mk_hvp(arr_x)
```

```
  0%|          | 0/6 [00:00<?, ?it/s]
```

[26]:
```
print('xh')
display(pd.DataFrame(arr_xh,index=NAMES,columns=xs))
print('xv')
display(pd.DataFrame(arr_xv,index=NAMES,columns=xs))
print('xp')
display(pd.DataFrame(arr_xp,index=NAMES,columns=xs))
```

xh

	0.0	1.0	2.5	5.0	10.0	25.0
unconstrained	0.03	0.05	0.16	0.57	2.23	13.79
extemely concervative (local)	0.03	0.05	0.15	0.42	1.58	9.78
very concervative (local)	0.03	0.05	0.13	0.41	1.58	9.78
concervative (local)	0.03	0.04	0.11	0.41	1.58	9.78
local meas only (proposed)	0.03	0.04	0.11	0.41	1.58	9.78
shared measurements	0.03	0.03	0.11	0.41	1.58	9.78

xv

	0.0	1.0	2.5	5.0	10.0	25.0
unconstrained	0.0	0.03	0.67	9.97	157.00	6071.21
extemely concervative (local)	0.0	0.03	0.57	2.86	3.90	4.00
very concervative (local)	0.0	0.03	0.40	0.91	1.00	1.00
concervative (local)	0.0	0.02	0.04	0.04	0.04	0.04
local meas only (proposed)	0.0	0.01	0.01	0.01	0.01	0.01
shared measurements	0.0	0.00	0.00	0.00	0.00	0.00

xp

	0.0	1.0	2.5	5.0	10.0	25.0
unconstrained	0.03	0.06	0.69	9.98	157.01	6071.22
extemely concervative (local)	0.03	0.06	0.59	2.89	4.21	10.57
very concervative (local)	0.03	0.05	0.42	1.00	1.87	9.83

concervative (local)	0.03	0.04	0.12	0.41	1.58	9.78
local meas only (proposed)	0.03	0.04	0.11	0.41	1.58	9.78
shared measurements	0.03	0.03	0.11	0.41	1.58	9.78

3 Compute HDOP, VDOP, PDOP for varying y at $(5, y, 1)$

[27]:
```
arr_y = mk_data_arr(y)
arr_yh, arr_yv, arr_yp = mk_hvp(arr_y)
```

```
  0%|          | 0/6 [00:00<?, ?it/s]
```

[28]:
```
print('yh')
display(pd.DataFrame(arr_yh,index=NAMES,columns=xs))
print('yv')
display(pd.DataFrame(arr_yv,index=NAMES,columns=xs))
print('yp')
display(pd.DataFrame(arr_yp,index=NAMES,columns=xs))
```

yh

	0.0	1.0	2.5	5.0	10.0	25.0
unconstrained	0.57	0.59	0.71	1.12	2.78	14.34
extemely concervative (local)	0.42	0.43	0.46	0.62	1.64	9.79
very concervative (local)	0.41	0.41	0.43	0.58	1.63	9.79
concervative (local)	0.41	0.41	0.42	0.57	1.62	9.79
local meas only (proposed)	0.41	0.41	0.42	0.56	1.62	9.79
shared measurements	0.41	0.41	0.42	0.56	1.62	9.79

yv

	0.0	1.0	2.5	5.0	10.0	25.0
unconstrained	9.97	10.77	15.51	39.45	245.04	6563.17
extemely concervative (local)	2.86	2.92	3.18	3.63	3.94	4.00
very concervative (local)	0.91	0.92	0.94	0.98	1.00	1.00
concervative (local)	0.04	0.04	0.04	0.04	0.04	0.04
local meas only (proposed)	0.01	0.01	0.01	0.01	0.01	0.01
shared measurements	0.00	0.00	0.00	0.00	0.00	0.00

yp

	0.0	1.0	2.5	5.0	10.0	25.0
unconstrained	9.98	10.79	15.52	39.47	245.06	6563.19
extemely concervative (local)	2.89	2.95	3.21	3.69	4.26	10.57
very concervative (local)	1.00	1.00	1.03	1.14	1.91	9.84
concervative (local)	0.41	0.41	0.42	0.57	1.62	9.79
local meas only (proposed)	0.41	0.41	0.42	0.56	1.62	9.79
shared measurements	0.41	0.41	0.42	0.56	1.62	9.79

4 Compute HDOP, VDOP, PDOP for varying y at $(5, 0, z)$

[29]:
```
arr_z = mk_data_arr(z)
arr_zh, arr_zv, arr_zp = mk_hvp(arr_z)
```

```
  0%|          | 0/6 [00:00<?, ?it/s]
```

[30]:
```
print('zh')
display(pd.DataFrame(arr_zh,index=NAMES,columns=xs))
print('zv')
display(pd.DataFrame(arr_zv,index=NAMES,columns=xs))
print('zp')
display(pd.DataFrame(arr_zp,index=NAMES,columns=xs))
```

zh

	0.0	1.0	2.5	5.0	10.0	25.0
unconstrained	0.39	0.57	0.69	1.11	2.77	14.37
extemely concervative (local)	0.39	0.42	0.59	1.02	2.62	13.72
very concervative (local)	0.39	0.41	0.52	0.90	2.36	12.47
concervative (local)	0.39	0.41	0.49	0.78	1.96	10.21
local meas only (proposed)	0.39	0.41	0.49	0.78	1.96	10.17
shared measurements	0.39	0.41	0.49	0.78	1.96	10.16

zv

	0.0	1.0	2.5	5.0	10.0	25.0
unconstrained	inf	9.97	1.93	0.78	0.49	0.41
extemely concervative (local)	4.00	2.86	1.30	0.65	0.44	0.37
very concervative (local)	1.00	0.91	0.66	0.44	0.33	0.29
concervative (local)	0.04	0.04	0.04	0.04	0.04	0.04
local meas only (proposed)	0.01	0.01	0.01	0.01	0.01	0.01
shared measurements	0.00	0.00	0.00	0.00	0.00	0.00

zp

	0.0	1.0	2.5	5.0	10.0	25.0
unconstrained	inf	9.98	2.05	1.35	2.81	14.38
extemely concervative (local)	4.02	2.89	1.43	1.21	2.66	13.73
very concervative (local)	1.08	1.00	0.84	1.00	2.38	12.48
concervative (local)	0.39	0.41	0.49	0.79	1.96	10.21
local meas only (proposed)	0.39	0.41	0.49	0.78	1.96	10.17
shared measurements	0.39	0.41	0.49	0.78	1.96	10.16

[ ]: