TricXControlArch Documentation and Manual:

(**TricXControlArch** is a control architecture, developed by Soumyadeep M. (me!), for controlling an underactuated tricopter that depends on differential thrusts for yaw control, thus eliminating the need for a tiltrotor mechanism which has thus far been used extensively in tricopters. Download source code from here:
https://drive.google.com/drive/folders/1s7xfNtkWbn8D15DFnis3TKTgtuyrHGDr?usp=sharing)

1. Run **TricXControlArch.m** and input drone characteristics, like Inertia terms and co-ordinates. Use Q and R values reasonably, Q should be reasonably higher than R. Check for stability.

2. In the script of **designTriRotor.m**, find the following code snippet, at the top:

```
% === Geometry (must match TriRotor.xml) ===
x = [0.185, -0.160, 0.045]; % [m]
y = [0.125, -0.150, 0.215]; % [m]
spin = [1, -1, 1 ]; % +1 / -1 for yaw torque

% === Mass & total hover thrust (must match XML) ===
m = 1.72; % kg
g = 9.81; % m/s^2
T_total = m * g; % total thrust to support weight [N]

% === Inertia tensor (same as XML) ===
I = [ 0.0149 -0.0018 0.0004;
-0.0018 0.0167 -0.0002;
0.0004 -0.0002 0.0284];

% === LQR weighting (rotational states only are really used) ===
% x = [p q r phi theta psi]
Q = diag([25 25 25 20 20 20]); % penalise rates + attitudes
R = diag([0.1 0.5 0.5 0.5]); % only R(2:4,2:4) used
```

Change the coordinates (x, y), mass(m), Inertia(I), Q and R as per entered in **TricXControlArch.m**.

Running **designTriRotor.m** after that should produce an output similar to what is shown below:

```
---- Design results ----
K_rot (before saving) will be computed below.
B_inv =
-92.0000 52.0000 22.1200
-112.0000 72.0000 27.3200
-20.0000 20.0000 6.2000

F0_hover =
6.7080 8.2850 1.8802

Equilibrium tau_eq = [L M N] (should have L,M ~ 0):
0.0000 0.0000 0.3033

K_rot =
7.0844 -0.0016 0.0004 6.3246 -0.0000 -0.0000
```

```
-0.0016 7.0860 -0.0002 0.0000 6.3246 0.0000
0.0004 -0.0002 7.0964 0.0000 0.0000 6.3246

LQR design complete.
```

Note that symmetrical layouts should be avoided, because it might lead to rank deficiency in the moment mixer, and B_inv might turn out to be infinity.

3. Update the aircraft on JSBSim's main XML code accordingly. In this manual, it has been named **TriRotor.xml**, and is to be placed inside the customized aircraft folder (named "**TriRotor**" here) under JSBSim.
4. The flight control systems file "**…_fcs.xml**" under the Systems folder can be left untouched, unless the user is seeking specialized flight dynamics.

5. The initial conditions of the aircraft can be defined in "**ic_trirotor.xml**".

6. In the source code of **runTriRotorUDP.m**, find the following snippet:

```
%% === Tuning knobs ===
gain_scale_final = 0.25; % final fraction of LQR gain (e.g. 0.25 = 25%)
ramp_time = 2.0; % [s] time to ramp from 0 -> gain_scale_final
zero_sum_dF = true; % enforce sum(dF) = 0 to avoid changing total thrust
invert_LM = true; % flip signs of roll/pitch torques if needed

% Torque saturation [N*m]
tau_max_roll_pitch = 0.20;
tau_max_yaw = 0.05;

% Rotor thrust limits [N]
Fmin = 0.0;
Fmax = 20.0;

fprintf("\n---- TriRotor LQR run ----\n");
fprintf("Hover trim F0 = [%.3f %.3f %.3f] N\n", F0_hover(1), F0_hover(2),
F0_hover(3));
fprintf("gain_scale_final = %.3f, ramp_time = %.2f s, zero_sum_dF = %d,
invert_LM = %d\n", ...
gain_scale_final, ramp_time, zero_sum_dF, invert_LM);
```

The value of "gain_scale_final", "ramp_time", "tau_max_roll_pitch", "tau_max_yaw", "Fmin" and "Fmax" can be tweaked as per the behavior of the drone in subsequent test runs. "Fmax" can be larger with increasing drone weight.

If ports were changed in the main XML file of the aircraft, find the following code snippet and change the ports accordingly:

```
%% === JSBSim I/O sockets ===
% UDP receive: state from JSBSim
u_rx = udpport("datagram", "IPV4", ...
"LocalPort", 5501, ...
"Timeout", 1.0); % 1 s timeout

% TCP send: thrust commands to JSBSim
jsb = tcpclient("127.0.0.1", 5502, "Timeout", 1.0);
pause(0.1); % let JSBSim settle
```

5501 is the output port, and 5502 is the input port, in the above snippet.

7. Connection between MATLAB and JSBSim can be checked by running **testTriRotorUDP.m** once.

8. Create a batch file like the following:

```
@echo off
cd /d D:\jsbsim-master\JSBSim

REM === Run JSBSim with TriRotor aircraft and init conditions ===
JSBSim.exe --aircraft=TriRotor --initfile=ic_trirotor.xml --realtime

pause
```

The location of JSBSim should be configured according to wherever it is located in the user's device. For this manual, this batch file is named "**run_trirotor.bat**".

9. In order to test the aircraft, please follow the following steps:

- Run **designTriRotor.m** once.
- Run **run_trirotor.bat** (or whatever this file has been named by the user) and if JSBSim is executing the run, type "`[t_log, x_log, F_log] = runTriRotorUDP(20);`" in the MATLAB command prompt, and wait for the runtime to get over. If the forces (F values) are realistic and not changing drastically, the control architecture is most likely working properly.
- The results can be plotted using the following on the MATLAB command prompt:

```
figure; plot(t_log, x_log);
legend('p','q','r','\phi','\theta','\psi');
figure; plot(t_log, F_log); legend('F1','F2','F3');
```

10. In the plots, \phi, \theta, and \psi represent the roll, pitch, and yaw respectively, and p, q, and r represent their rates.

11. If the curves are ugly, it doesn't necessarily mean that the control systems aren't working, because for this manual the initial conditions have been set such that the controls are applied on the drone after it is released at an altitude of 0.3ft at hover thrust, so somewhat unstable behavior is to be expected. A reasonable thrust adjustment means that the control system is most likely working fine, it tries to keep the aircraft hovering by applying differential thrusts to each rotor.