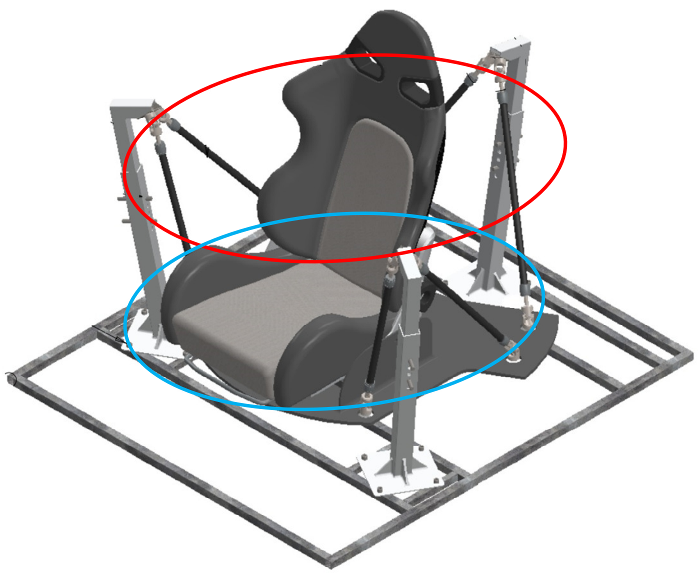

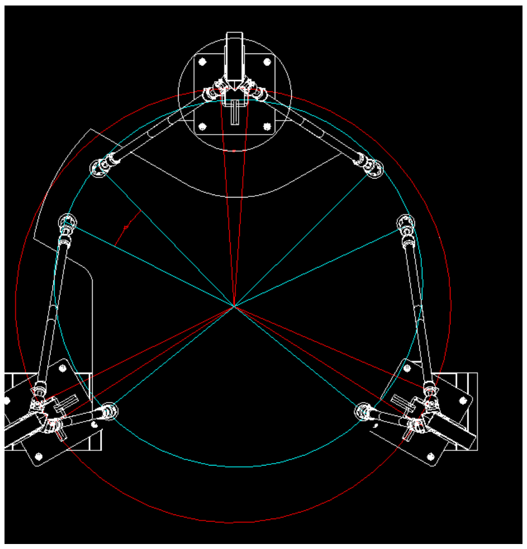

Variables used in transform code (see diagrams above):

base radius is radius of cyan circle
baseAngles are angles of cyan lines
Platform radius is radius of red circle
platformAngles are angles of red lines
Note that some error is introduced because circles are not concentric and the planes are not parallel

Here is the java class for the effector platform definition:

```
    public static class effectorDef {
        float baseAngles[] ;
        float baseRadius;         // mm
        float platformAngles[] ;
        float platformRadius;     // mm
        float actuatorLen[];      // min, max millimetres
        float initialHeight;      // mm distance between base & platform when centered
        float maxTranslation;     // mm
        float maxRotation;        // angle in degrees
        float minActuatorLen;     // mm
        float maxActuatorLen;     // mm
        public effectorDef() {
            baseAngles = new float[6];
            platformAngles = new float[6];
            actuatorLen = new float[2];
        }
    }
```

```
If you want to plug in actual numbers:
baseRadius = 442
platformRadius=540
initialHeight = 728
maxTranslation = 40
maxRotation= 25
actuatorLen = [666,790]
platformAngles = [147,154,266,274,26,33]
baseAngles= [140,207,226,314,334,40]
```

Here is the Java transform code:

```
import java.lang.Math;

/*  Modules for rotation and translation transforms */

public class PlatfromTransforms {
    private PVector translation, rotation, initialHeight;
    private PVector[] baseJoint, platformJoint, q, l, A;
    private float baseRadius, platformRadius;
    private float maxTranslation, maxRotationRadians;
    private float minLen;
    private float maxLen;

    public PlatfromTransforms() {
    }

    public void begin(EffectorInterface.effectorDef def) {
        minLen = def.minActuatorLen;
        maxLen = def.maxActuatorLen;
        translation = new PVector();
        initialHeight = new PVector(0, 0, def.initialHeight );
        rotation = new PVector();
        baseJoint = new PVector[6];
        platformJoint = new PVector[6];

        q = new PVector[6];
        l = new PVector[6];
        A = new PVector[6];
        baseRadius = def.baseRadius;
        platformRadius = def.platformRadius;
        maxTranslation = def.maxTranslation;
        maxRotationRadians = (float)toRadians(def.maxRotation);

        for (int i = 0; i < 6; i++) {
            float mx = (float) (baseRadius * cos(toRadians(def.baseAngles[i])));
            float my = (float) (baseRadius * sin(toRadians(def.baseAngles[i])));
            baseJoint[i] = new PVector(mx, my, 0);
        }
        for (int i=0; i<6; i++) {
            float mx = (float)(platformRadius * cos(toRadians(def.platformAngles[i])));
            float my = (float)(platformRadius * sin(toRadians(def.platformAngles[i])));
            platformJoint[i] = new PVector(mx, my, 0);
            q[i] = new PVector(0,0,0);
            l[i] = new PVector(0,0,0);
            A[i] = new PVector(0,0,0);
        }
        calcQ();
    }

    public void applyTranslationAndRotation(PVector t, PVector r) {
        rotation.set(PVector.mult(r, maxRotationRadians));
        translation.set(PVector.mult(t,maxTranslation));
        calcQ();
    }

    public float getRawLength(int index) {
         float v = l[index].mag();
         float vnormalized = -1 + ((v - minLen) * 2 / (maxLen - minLen));
         return (vnormalized) ;
    }
    private void calcQ() {
        for (int i=0; i<6; i++) {
        // rotation
          q[i].x = (float)(cos(rotation.z)*cos(rotation.y)*platformJoint[i].x +
             (-sin(rotation.z)*cos(rotation.x)+
             cos(rotation.z)*sin(rotation.y)*sin(rotation.x))*platformJoint[i].y +
             (sin(rotation.z)*sin(rotation.x)+
             cos(rotation.z)*sin(rotation.y)*cos(rotation.x))*platformJoint[i].z);

              q[i].y = (float) (sin(rotation.z)*cos(rotation.y)*platformJoint[i].x +
                (cos(rotation.z)*cos(rotation.x)+
                sin(rotation.z)*sin(rotation.y)*sin(rotation.x))*platformJoint[i].y +
                (-cos(rotation.z)*sin(rotation.x)+
                sin(rotation.z)*sin(rotation.y)*cos(rotation.x))*platformJoint[i].z);

              q[i].z = (float) (-sin(rotation.y)*platformJoint[i].x +
                      cos(rotation.y)*sin(rotation.x)*platformJoint[i].y +
                      cos(rotation.y)*cos(rotation.x)*platformJoint[i].z);

              // translation
              q[i].add(PVector.add(translation, initialHeight));
              l[i] = PVector.sub(q[i], baseJoint[i]);
            }
       }
}
```