Parallel axis theorem

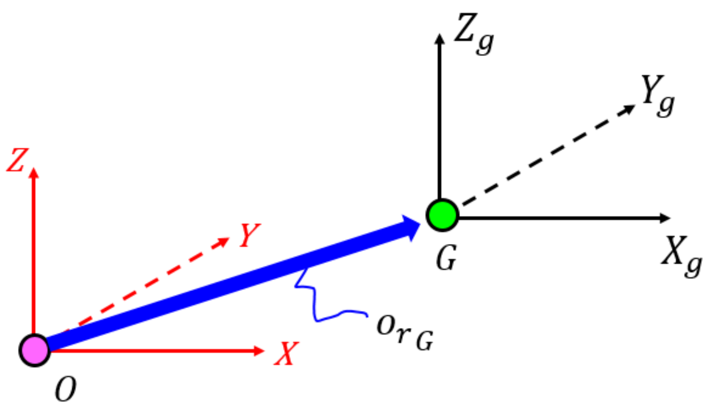
What we're going to do:

In this FAQ, we're going to define the 3 dimensional version of the popular parallel axis theorem.

`Bradley Horton : 01-Mar-2016, bradley.horton@mathworks.com.au`

Starting the derivation:

Consider the following:



The *G-frame* represents a body fixed frame positioned at the Centre of mass(CM) of the body. We know the Inertia matrix for this *G-frame* and write it as:

$$I_G = \begin{pmatrix} {}^{G}I_{XX} & {}^{G}I_{XY} & {}^{G}I_{XZ} \\ {}^{G}I_{XY} & {}^{G}I_{YY} & {}^{G}I_{YZ} \\ {}^{G}I_{XZ} & {}^{G}I_{YZ} & {}^{G}I_{ZZ} \end{pmatrix} \quad \text{where} \quad \begin{aligned} {}^{G}I_{ZZ} &= \int (x_g^2 + y_g^2)\, dm \\ {}^{G}I_{XY} &= \int (-1 * x_g * y_g)\, dm \end{aligned} \quad \text{etc.}$$

And recall that because G is at the CM, we have: $0 = \int x_g \; dm = \int y_g \; dm = \int z_g \; dm$

The *0-frame* is parallel to the *G-frame*. We wish to determine the inertia matrix of the body relative to the *0-frame*. The position of the *G-frame* relative to the *0-frame* and expressed in components of the *0-frame*, is given by ${}^{0}\vec{r}_G$ where:

$${}^{0}\vec{r}_G = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

So let's look at ${}^{0}I_{ZZ}$:

$${}^{0}I_{ZZ} = \int (x_0^2 + y_0^2)\, dm = \int \left((a + x_g)^2 + (b + y_g)^2 \right) dm = \int (a^2 + 2ax_g + x_g^2 + b^2 + 2by_g + y_g^2)\, dm$$

$$ {}^0I_{ZZ} = \int (a^2 + b^2) dm + \int (x_g^2 + y_g^2)\, dm + 2a \int x_g dm + 2b \int y_g dm $$

So finally: ${}^0I_{ZZ} = m.(a^2 + b^2) \quad + \quad {}^GI_{XX}$

Now let's look at ${}^0I_{XY}$:

$$ {}^0I_{XY} = \int (-1 * x_0 * y_0)\, dm = \int (-1 * (a + x_g) * (b + y_g) dm $$

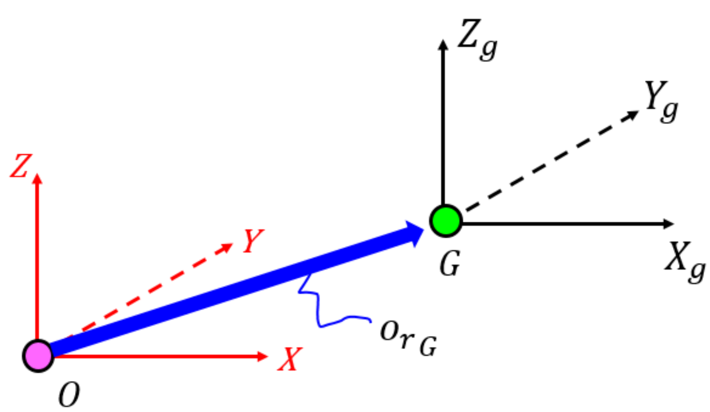
$$ {}^0I_{XY} = -1 * \int ab + ay_g + bx_g + x_g y_g dm = \int (-1 * a * b) dm \quad + \quad {}^GI_{XY} \quad + \quad \int -ay_g dm + \int -bx_g dm $$

So finally: ${}^0I_{XY} = m.(-1 * a * b) \quad + \quad {}^GI_{XY}$

And by applying the same process we can compute similar values for ${}^0I_{yy}$ and ${}^0I_{XZ}$, etc. So let's summarise what we've just derived:

Concluding the derivation:

Recall the diagram presented in the previous section:



where $\overrightarrow{{}^0r_G} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$ and

$$ {}^0I_{ZZ} = \int (x_0^2 + y_0^2)\, dm $$
$$ {}^0I_{XY} = \int (-1 * x_0 * y_0)\, dm $$
etc.

From the derivations in the previous section we can now write:

$$ {}^0I = \begin{pmatrix} {}^0I_{XX} & {}^0I_{XY} & {}^0I_{XZ} \\ {}^0I_{XY} & {}^0I_{YY} & {}^0I_{YZ} \\ {}^0I_{XZ} & {}^0I_{YZ} & {}^0I_{ZZ} \end{pmatrix} = {}^GI \quad + \quad m \times \begin{pmatrix} (b^2 + c^2) & (-ab) & (-ac) \\ (-ab) & (a^2 + c^2) & (-bc) \\ (-ac) & (-bc) & (a^2 + b^2) \end{pmatrix} $$

The implementation:

I've implemented the above formula into a MATLAB class called **inertia_parallel_local_to_desired_CLS**. Here's the implementation

```
dbtype inertia_parallel_local_to_desired_CLS
```

```
classdef inertia_parallel_local_to_desired_CLS

    properties  (SetAccess = protected)
        I_local          = [];
        mass             = [];
        pos_CM_in_desired = [];
    end

    methods
        function  OBJ = inertia_parallel_local_to_desired_CLS( ...
                                     pos_CM_rel_to_desired, I_LOC, mass)
            OBJ.I_local           = I_LOC;
            OBJ.mass              = mass;
            OBJ.pos_CM_in_desired = pos_CM_rel_to_desired;
        end
        %------------------------------------------------------------------
        function I_desired = calc_I_GLOB(OBJ)
            x_c  =  OBJ.pos_CM_in_desired(1);
            y_c  =  OBJ.pos_CM_in_desired(2);
            z_c  =  OBJ.pos_CM_in_desired(3);

           A     =  [ (y_c^2 + z_c^2),         -x_c*y_c,         -x_c*z_c;
                             -x_c*y_c,  (x_c^2 + z_c^2),         -y_c*z_c;
                             -x_c*z_c,         -y_c*z_c,  (x_c^2 + y_c^2);
                    ];

            I_desired = OBJ.I_local + (OBJ.mass * A );
        end
        %------------------------------------------------------------------
    end

end
```

The above implementation will work for either NUMERIC or symbolic variables. Let's look at an example:

An example (SYMBOLIC):

```
syms m I_xx I_xy I_xz I_yy I_yz I_zz a b c

% define the LOCAL G-frame inertias
gI = [ I_xx, I_xy, I_xz;
       I_xy, I_yy, I_yz;
       I_xz, I_yz, I_zz ];

% define the position of G relative to the 0-frame (and in units of the 0-frame)
r = [a,b,c].';

% create an instance of the inertia_local_to_desired_CLS class
OBJ = inertia_parallel_local_to_desired_CLS(r, gI, m);
```

```
% compute the INERTIA relative to the 0-frame
OBJ.calc_I_GLOB()
```

ans =

$$\begin{pmatrix} I_{xx} + m\,(b^2 + c^2) & I_{xy} - a\,b\,m & I_{xz} - a\,c\,m \\ I_{xy} - a\,b\,m & I_{yy} + m\,(a^2 + c^2) & I_{yz} - b\,c\,m \\ I_{xz} - a\,c\,m & I_{yz} - b\,c\,m & I_{zz} + m\,(a^2 + b^2) \end{pmatrix}$$