Stewart-Gough Platform Kinematics

Jak_o_Shadows

December 24, 2015

1 Coordinate System Definitions

Two coordinate systems are used. The first corresponds to the base plate, and the second the moveable upper platform. For each coordinate system, $\hat{z}$ is upwards, and $\hat{x}$, $\hat{y}$ are orthogonal within each plate. These can be seen in Figure 1 below.

Denoting the coordinate systems as follows

- Base $\boldsymbol{B}$
- Platform $\boldsymbol{P}$

A set of coordinates would then be denoted as below if in the base coordinate system

$$\boldsymbol{x}^{\boldsymbol{B}} = \begin{bmatrix} x_{coord} \\ y_{coord} \\ z_{coord} \end{bmatrix}$$

2 Sensor/Actuator Locations

A Stewart-Gough platform has 6 actuators or sensors (henceforce sensor). These are typically attached radially around the edge of the base and platform.
Let:

- $\boldsymbol{b}_i^{\boldsymbol{B}} = \begin{bmatrix} x_i & y_i & z_i \end{bmatrix}^T$ be the attachment location of the i_{th} sensor to the base. Note it is in the base coordinate system.
- $\boldsymbol{p}_i^{\boldsymbol{P}} = \begin{bmatrix} x_i & y_i & z_i \end{bmatrix}^T$ be the attachment location of the i_{th} sensor to the platform. Note it is in the platform coordinate system.

3 Coordinate Transformation

To do FK or IK, it is necessary to know both ends of each sensor in the same coordinate system. As the platform coordinate system would (usually) be moving with regards to the world, the sensor platform attachment positions are converted to the base coordinate system.

To fully describe the $\boldsymbol{P}$ coordinate system with respect to $\boldsymbol{B}$, a translation and rotation is needed. Let

$$\boldsymbol{a} = \begin{bmatrix} x_p & y_p & z_p & \alpha & \beta & \gamma \end{bmatrix}^T$$

where:

Figure 1: Coordinate system definition

- α: Roll angle - rotation about x-axis
- β: Pitch angle - rotation about y-axis
- γ: Yaw angle - rotation about z-axis
- x, y, z: Position of origin of P coordinate system

3.1 Translation Effects

Hence transform platform attachment coordinates to base coordinate system.

$$\boldsymbol{p_i^P} \longrightarrow \boldsymbol{p_i^B}$$

The purely translational effect is achieved through:

$$\boldsymbol{p_i^B} = \boldsymbol{p_i^P} - \begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix}$$

However the effect of the rotation of the platform is more complicated.

3.2 Euler Angles & Rotation Matrices

DO THIS SECTION FIX MEDO THIS SECTION FIX MEDO THIS SECTION FIX MEDO THIS SECTION FIX ME 3-2-1 rotation. http://ntrs.nasa.gov/search.jsp?R=19770024290 Rotation matrices about the three coordinate axis are:

- $\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ gives rotation matrix, for θ_1, $\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_1 & -\sin\theta_1 \\ 0 & \sin\theta_1 & \cos\theta_1 \end{bmatrix}$
- $\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$ gives rotation matrix, for θ_2, $\begin{bmatrix} \cos\theta_2 & 0 & \sin\theta_2 \\ 0 & 1 & 0 \\ -\sin\theta_2 & 0 & \cos\theta_2 \end{bmatrix}$
- $\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ gives rotation matrix, for θ_3, $\begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 \\ \sin\theta_3 & \cos\theta_3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Hence, as $\theta_1 = \alpha$, $\theta_2 = \beta$, $\theta_3 = \gamma$, performing a 3-2-1 rotation gives the rotation matrix:

$$R_{zyx} = \begin{bmatrix} \cos\theta_1\cos\theta_2 & \cos\theta_1\sin\theta_2\sin\theta_3 - \sin\theta_1\cos\theta_2 & \cos\theta_1\sin\theta_2\cos\theta_3 + \sin\theta_1\sin\theta_3 \\ \sin\theta_1\cos\theta_2 & \sin\theta_1\sin\theta_2\sin\theta_3 + \cos\theta_1\cos\theta_3 & \sin\theta_1\sin\theta_2\cos\theta_3 - \cos\theta_1\sin\theta_3 \\ -\sin\theta_2 & \cos\theta_2\sin\theta_3 & \cos\theta_2\cos\theta_3 \end{bmatrix}$$

Using the transformation matrix is just:

$$\boldsymbol{x}_{uvw} = R_{zyx}\boldsymbol{p_i^P}$$

3.3 Translation and Rotation

Hence the platform sensor attachment locations in the base coordinate system are:

$$\boldsymbol{p_i^B} = \left(R_{zyx}\boldsymbol{p_i^P}\right) + \left(\boldsymbol{p_i^B} - \begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix}\right)$$

However, FK and IK only need the length of the sensors. Denoting this $\bar{\boldsymbol{x}}$:

$$\bar{\boldsymbol{x}} = \begin{bmatrix} \bar{x} \\ \bar{y} \\ \bar{z} \end{bmatrix} = \boldsymbol{p_i^B} - \boldsymbol{b_i^B}$$

The length of this vector is simply:

$$l_i = \|\bar{\boldsymbol{x}}\| = \sqrt{\bar{x}^2 + \bar{y}^2 + \bar{z}^2}$$

4 Kinematics

Both forward and inverse kinematics (FK, IK) both involve finding the position of the platform in the base coordinate system, and subsequently finding the sensor lengths. The above derivation is used for both.

4.1 Inverse Kinematics

The inverse kinematics problem is finding the actuator lengths for a given platform position. This is simply finding l_i as above. Note that l_i should be checked against possible actuator range to determine whether the platform position is feasible.

It is possible to rewrite l_i directly in terms of the elements of the rotation matrix R_{zyx} —this allows l_i to be written directly as a function of $\boldsymbol{a}, \boldsymbol{p}_i^{\boldsymbol{P}}, \boldsymbol{b}_i^{\boldsymbol{B}}$ This may be advantageous when programming in an environment that does not allow matrix algebra. See Nguyen, Zhou & Antrazi 1991 for details.

4.2 Forward Kinematics

The basic idea behind forward kinematics of a stewart-gough platform is to find the position of the platform such that the model sensor lengths are the same as your sensor values. This leads to a quadratic optimisation problem.

4.2.1 Objective Function

The objective functions for this problem are related to l_i. Denoting L_i as the input length of each sensor, the functions to be minimised are:

$$f_i(\boldsymbol{a}) = -\left(l_i^2 - L_i^2\right)$$

$$f_i(\boldsymbol{a}) = -\left(\bar{x_i}^2 + \bar{y_i}^2 + \bar{z_i}^2 - L_i^2\right)$$

As these functions are unimodal (as quadratic functions), optimisation will provide the global minimum (simplifying things a bit —it gets more complicated in higher dimensions, as we are here). It is reasonable (guessing) to seek to minimise the sum of these functions, ie.

$$f(\boldsymbol{a}) = \sum_{i=1}^{6} f_i(\boldsymbol{a})$$

Optimisation will have occurred when this value is within a specified tolerance

4.2.2 Constraints

It is reasonable to define a series of constraints on the allowed length of each sensor in the model. However, as these constraints take the form

$$\bar{x_i}^2 + \bar{y_i}^2 + \bar{z_i}^2 \geq 0$$

they are difficult to include in an optimisation scheme.

Figure 2: Example of Newton-Raphson iteration

4.2.3 Solving using Newton-Raphson Iteration

Newton-Raphson iteration is method of solving for a zero (or root) of a function. By taking steps along the tangent of the function at the current point, new points are generated. When the function value is zero, no step is taken, and thus the algorithm ceases. In practice, iteration is stopped when the function value is small, or when the step to be taken is small. The objective function $f_i(\boldsymbol{a})$ represents the difference in length between the sensor inputs and the sensor lengths found using the position of the platform. Note that this can have multiple solutions; for Newton-Raphson iteration, starting guess is important. Not only does it determine whether the "correct" solution is found, it also effects whether the algorithm will converge, as well as convergence speed.

Newton-Raphson iteration approaches the zero by taking linear steps along the tangent, as seen in Figure 2. The slope of the tangent at $\boldsymbol{a}$ is $\nabla_a f_i$. Taking a step of size $f_i(\boldsymbol{a})$:

$$\boldsymbol{a}_{new} = \boldsymbol{a} + \left[\nabla_a f_i(\boldsymbol{a})\right]^{-1} f_i(\boldsymbol{a})$$

Note that $\left[\nabla_a f_i(\boldsymbol{a})\right]^{-1} f_i(\boldsymbol{a})$ is equivalent to solving the problem $\nabla_a f_i(\boldsymbol{a})\,\delta = f_i(\boldsymbol{a})$ for δ, which is easily solved using existing libraries (eg Matlab, NumPy, etc).

4.2.3.1 Derivation of $\nabla_a f_i(\boldsymbol{a})$

Hence calculate $\nabla_a f_i(\boldsymbol{a})$. As $f_i(\boldsymbol{a}) = -\left(\bar{x_i}^2 + \bar{y_i}^2 + \bar{z_i}^2 - L_i^2\right)$:

$$\nabla_a f_i = \begin{bmatrix} 2\bar{x}_i \\ 2\bar{y}_i \\ 2\bar{z}_i \\ 2\bar{x}_i\frac{\partial \bar{x_i}}{\partial \alpha} + 2\bar{y}_i\frac{\partial \bar{y_i}}{\partial \alpha} + 2\bar{z}_i\frac{\partial \bar{z_i}}{\partial \alpha} \\ 2\bar{x}_i\frac{\partial \bar{x_i}}{\partial \beta} + 2\bar{y}_i\frac{\partial \bar{y_i}}{\partial \beta} + 2\bar{z}_i\frac{\partial \bar{z_i}}{\partial \beta} \\ 2\bar{x}_i\frac{\partial \bar{x_i}}{\partial \gamma} + 2\bar{y}_i\frac{\partial \bar{y_i}}{\partial \gamma} + 2\bar{z}_i\frac{\partial \bar{z_i}}{\partial \gamma} \end{bmatrix} = \begin{bmatrix} 2\bar{x}_i \\ 2\bar{y}_i \\ 2\bar{z}_i \\ 2\left(\bar{x}_i\frac{\partial \bar{x_i}}{\partial \alpha} + \bar{y}_i\frac{\partial \bar{y_i}}{\partial \alpha} + \bar{z}_i\frac{\partial \bar{z_i}}{\partial \alpha}\right) \\ 2\left(\bar{x}_i\frac{\partial \bar{x_i}}{\partial \beta} + \bar{y}_i\frac{\partial \bar{y_i}}{\partial \beta} + \bar{z}_i\frac{\partial \bar{z_i}}{\partial \beta}\right) \\ 2\left(\bar{x}_i\frac{\partial \bar{x_i}}{\partial \gamma} + \bar{y}_i\frac{\partial \bar{y_i}}{\partial \gamma} + \bar{z}_i\frac{\partial \bar{z_i}}{\partial \gamma}\right) \end{bmatrix}$$

Note that the only components of $\boldsymbol{x_i}$ which are dependant on the angles α, β, γ come from the $\boldsymbol{x_{uvw}} = R_{zyx}\boldsymbol{p_i^P}$ terms. Hence using the normal derivative rules, the partial derivatives of R_{zyx} with respect to the variables in α are:

$$\frac{\partial R_{zyx}}{\partial \alpha} = \begin{bmatrix} 0 & \cos(\alpha)\cos(\gamma)\sin(\beta) + \sin(\alpha)\sin(\gamma) & \cos(\gamma)\sin(\alpha) - \cos(\alpha)\sin(\beta)\sin(\gamma) \\ 0 & \cos(\gamma)\sin(\alpha)\sin(\beta) - \cos(\alpha)\sin(\gamma) & -\cos(\alpha)\cos(\gamma) - \sin(\alpha)\sin(\beta)\sin(\gamma) \\ 0 & \cos(\beta)\cos(\gamma) & -\cos(\beta)\sin(\gamma) \end{bmatrix}$$

$$\frac{\partial R_{zyx}}{\partial \beta} = \begin{bmatrix} -\cos(\alpha)\sin(\beta) & \cos(\alpha)\cos(\beta)\sin(\gamma) & \cos(\alpha)\cos(\beta)\cos(\gamma) \\ -\sin(\alpha)\sin(\beta) & \cos(\beta)\sin(\alpha)\sin(\gamma) & \cos(\beta)\cos(\gamma)\sin(\alpha) \\ -\cos(\beta) & -\sin(\beta)\sin(\gamma) & -\cos(\gamma)\sin(\beta) \end{bmatrix}$$

$$\frac{\partial R_{zyx}}{\partial \gamma} = \begin{bmatrix} -\cos(\beta)\sin(\alpha) & -\cos(\alpha)\cos(\gamma) - \sin(\alpha)\sin(\beta)\sin(\gamma) & \cos(\alpha)\sin(\gamma) - \cos(\gamma)\sin(\alpha)\sin(\beta) \\ \cos(\alpha)\cos(\beta) & \cos(\alpha)\sin(\beta)\sin(\gamma) - \cos(\gamma)\sin(\alpha) & \cos(\alpha)\cos(\gamma)\sin(\beta) + \sin(\alpha)\sin(\gamma) \\ 0 & 0 & 0 \end{bmatrix}$$

Note that as $\nabla_a \boldsymbol{x}_{uvw} = \nabla_a R_{zyx}\boldsymbol{p_i^P}$ these lead to the following. Denoting p_{ix}^P as the x component of $\boldsymbol{p_i^P}$ (and etc):

$$\frac{\partial \boldsymbol{x}_{uvw}}{\partial \alpha} = \begin{bmatrix} (\cos(\alpha)\cos(\gamma)\sin(\beta) + \sin(\alpha)\sin(\gamma))p_{iy}^P + (\cos(\gamma)\sin(\alpha) - \cos(\alpha)\sin(\beta)\sin(\gamma))p_{iz}^P \\ (\cos(\gamma)\sin(\alpha)\sin(\beta) - \cos(\alpha)\sin(\gamma))p_{iy}^P + (-\cos(\alpha)\cos(\gamma) - \sin(\alpha)\sin(\beta)\sin(\gamma))p_{iz}^P \\ \cos(\beta)\cos(\gamma)p_{iy}^P - \cos(\beta)\sin(\gamma)p_{iz}^P \end{bmatrix}$$

$$\frac{\partial \boldsymbol{x}_{uvw}}{\partial \beta} = \begin{bmatrix} -\cos(\alpha)\sin(\beta)p_{ix}^P + \cos(\alpha)\cos(\beta)\sin(\gamma)p_{iy}^P + \cos(\alpha)\cos(\beta)\cos(\gamma)p_{iz}^P \\ -\sin(\alpha)\sin(\beta)p_{ix}^P + \cos(\beta)\sin(\alpha)\sin(\gamma)p_{iy}^P + \cos(\beta)\cos(\gamma)\sin(\alpha)p_{iz}^P \\ -\cos(\beta)p_{ix}^P - \sin(\beta)\sin(\gamma)p_{iy}^P - \cos(\gamma)\sin(\beta)p_{iz}^P \end{bmatrix}$$

$$\frac{\partial \boldsymbol{x}_{uvw}}{\partial \gamma} = \begin{bmatrix} -\cos(\beta)\sin(\alpha)p_{ix}^P + (-\cos(\alpha)\cos(\gamma) - \sin(\alpha)\sin(\beta)\sin(\gamma))p_{iy}^P + (\cos(\alpha)\sin(\gamma) - \cos(\gamma)\sin(\alpha)\sin(\beta))p_{iz}^P \\ \cos(\alpha)\cos(\beta)p_{ix}^P + (\cos(\alpha)\sin(\beta)\sin(\gamma) - \cos(\gamma)\sin(\alpha))p_{iy}^P + (\cos(\alpha)\cos(\gamma)\sin(\beta) + \sin(\alpha)\sin(\gamma))p_{iz}^P \\ 0 \end{bmatrix}$$

4.2.3.2 $\nabla_a f_i(\boldsymbol{a})$ is

It is not obvious, however that this simplifies $\nabla_a f_i(\boldsymbol{a})$. Denote u_i, v_i, w_i as the components of $\boldsymbol{x}_{uvw}$.

$$\nabla_a f_i(\boldsymbol{a}) = \begin{bmatrix} 2\bar{x}_i \\ 2\bar{y}_i \\ 2\bar{z}_i \\ 2p_{iy}^P \left[(\bar{x}_i - u_i)\left[R_{zyx}\right]_{1,3} + (\bar{y}_i - v_i)\left[R_{zyx}\right]_{2,3} + (\bar{z}_i - w_i)\left[R_{zyx}\right]_{3,3} \right] \\ 2\left[((\bar{x}_i - u_i)\cos\alpha + (\bar{y}_i - v_i)\sin\alpha)\, w_i - \left(p_{ix}^P \cos\beta + p_{iy}^P \sin\beta \sin\gamma\right)(\bar{z}_i - w_i) \right] \\ 2\left[(\bar{x}_i - u_i)\, v_i + (\bar{y}_i - v_i)\, u_i \right] \end{bmatrix}$$

Remember Newton-Raphson iteration can be written as:

$$\boldsymbol{a}_{new} = \boldsymbol{a} + \left[\nabla_a f_i(\boldsymbol{a})\right]^{-1} f_i(\boldsymbol{a})$$

Convergence is achieved when there is no significant change in the value of $\boldsymbol{a}$ —when $\sum_{i=1}^{6} \left[\nabla_a f_i(\boldsymbol{a})\right]^{-1} f_i(\boldsymbol{a}) \leq$ a tolerance.

Newton-Raphson iteration, as used in this instance, is not actually finding the minimum of a generic quadratic function. It is actually finding a zero (or root) of the function. However due to the nature of this problem, as well as it's constraints, these come to the same thing.

5 Code

Python code for both inverse and forward kinematics is available. This code heavily uses the NumPy library, and is vectorized (not using for loops) to a reasonable extent —hopefully enough to allow real time use.

- Download Inverse Kinematics
- Download Forward Kinematics

INVERSE KINEMATICS NOT AVAILABLE YET LINK LINK LINK LINK LINK LINK

6 References

Nguyen C. C, Zhou Z, Antrazi S. S 1991, *Efficient Computation of Forward Kinematics and Jacobian Matrix of a Stewart Platform-based Manipulator: IEEE Proceedings of Southeastcon '91*, Williamsburg, VA, 7-10 April 1991.