NORTHWESTERN UNIVERSITY

Kinematics and Control Implementation of Mobile Manipulator System

A THESIS

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

MASTER OF SCIENCE

Field of Mechanical Engineering

By

Wanlin Yang

EVANSTON, ILLINOIS

December 2017

Acknowledgements

I would like to express my sincere gratitude to my advisor Professor Kevin M. Lynch, who gave me the chance to be a member of the Boeing project, providing good suggestions and ideas for my master thesis research. Besides my advisor, I would like to thank the Boeing project group advisors, Prof. Edward Colgate, Prof. Todd Murphey, Dr. Ron Worth, Dr. Jarvis Schultz, Dr. Matthew Elwin and the rest group members. This thesis research is the group work with Jialei Hu, a dual-master program student from Shanghai Jiaotong University. Most of the research work will also be included in his thesis. I would also like thank my family for their financial and spiritual supports for my whole master study.

ABSTRACT

Kinematics and Control Implementation of Mobile Manipulator System

Wanlin Yang

This paper studies the kinematics and corresponding control implementation of the mobile manipulator mechanism, a Delta platform. The inverse, forward, velocity kinematics, workspace, statics, velocity and force limits are derived in this paper. The motion and force control strategy and their hardware implementation have been developed based on our Delta robot C library, and the usage of several kinds of encoder is also introduces in this paper. The results demonstrate the reliability and feasibility of the algorithms and control system.

Table of Contents

CHAPTER 1

Introduction

This mobile manipulator system aims to build several mobile manipulators to cooperatively carry a large object, such as an airplane wing. Each single mobile manipulator should provide enough force along linear directions, especially along z-axis. Closed-chain mechanisms are more suitable than opened-chain mechanisms in this condition. After comparing with other parallel mechanism, such as Steward and Merlet platform with 6 DOF, Delta robot is the finalized mechanism.

Delta mechanism is a type of parallel robots, as shown in Figure 1.1. It consists of upper and lower platforms, which are connected by three legs, and each leg is a RUU serial chain. The three R joints are actuated by motors, and the rest U joints are unsaturated in the mechanism. The Delta mechanism has three degrees of freedom on $(x, y, z)^T$ linear direction.

The coordinates, labels and parameters describing the configuration of Delta mechanism are shown in Figure 1.1. The center of the lower platform is the origin of space frame, and the center of the upper platform is the origin of body frame. In this paper, the position of end-effector means the vector $\vec{p} = (x, y, z)^T$ from space frame to body frame. The angle of a joint is described as the angle of the corresponding leg to vertical.

Our goal is to analyze the kinematics of the Delta platform and develop an embedded control system to manipulate three motors of Delta platform following the input trajectory of end-effector by user, while the controller can send back the real-time position of motors to a visible interface at the same time.

Four separate microcontrollers, which are NU32s in this research project, work together through SPI communication protocol. One of them is the SPI master that receives the instructions from human and calculates motion of each motor, and the other three are SPI slaves that performs the desired motion sent from master in a feedback control loop.

The controller may not be NU32 in future industrial implementations, so the algorithms and functions should be able to be ported over other hardware. C functions in this research satisfy the command of modularity for expansion.

This paper starts with the derivation of inverse, forward, and velocity kinematics, as well as their corresponding implementations on MATLAB. Then we clarifies the motion and force control strategies of the system. And finally the usage of our software and the C functions are illustrated piece by piece. All of the related codes are posted on: `https://github.com/WanlinYang/Delta_Mechanism`

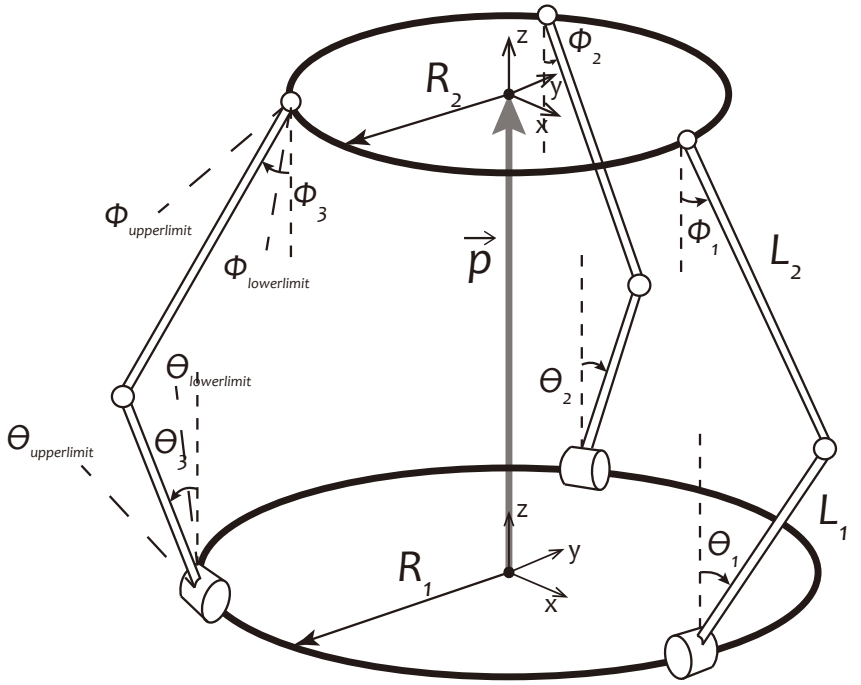

Figure 1.1. Delta Mechanism

CHAPTER 2

Related Work

Robert L. Williams [**1**] described a possible solution to the inverse and velocity kinematics of Delta robot. But they were using the up-side-down triangle Delta platform that different from ours, and their coordinates are also different. So we should figure out the solutions and combine the acceptable motion ranges in our research.

Some research and patents of compliant platform are helpful for designing the mechanism that serially connects legs and the lower platform by torsion springs. Y. Zheng et al [**2**] provides a six-degree-of-freedom quasi-zero-rigidity vibration isolation system based on a Stewart platform. The system has the advantages of being high in bearing capacity and low in resonant frequency, and is suitable for vibration isolation of flywheel systems and optical cameras in the spaceflight field and vibration isolation of equipment such as precision machine tools and precision measuring systems in the civil field. D. Malchev et al [**3**] designed a shock isolation structure for mounting a radar system to a supporting surface on board of a vessel that includes a platform on which the radar system can be attached. The damping elements are oriented in a truss configuration with first ends of the damping elements connected to the supporting surface for universal movement and with second ends of the damping elements connected to the platform for universal movement. Each of the damping elements includes a magnetorheological or electrorheological fluid damper.

A compliant platform may have multiple solutions based on different given parameters. H. Tari et al [**4**] studies a compliant Stewart platform whose six legs balance external wrench. Five major problem types are recognized based on known and unknown parameters. The number of solutions is varying according to the known and unknown parameter types, and in some cases, only a small part of them are valid that possess positive displaced leg length components. Y. Moon et al [**5**] solve for the length of springs and pistons, given the position of end-effector and external wrench, to reach a static equilibrium state. As for multiple solutions, the minimum energy constraint problem and path tracking problem is given, to reduce the number of solution to one. Stiffness synthesis problem is to solve for the instantaneous displacement of the leg connectors and the spring constants of all springs such that the stiffness matrix matches with a given desired stiffness matrix value, which is same as our stiffness problem. Y. Li et al [**6**] divided the platform into two subsystems, parallel links and machine frame. The stiffness of the whole system is the superposition of that of the subsystems. And they tested the stiffness thorough FEM method.

CHAPTER 3

Kinematics and Stiffness

This section illustrates some basic algorithms of Delta mechanism, including inverse kinematics, forward kinematics, velocity kinematics, workspace, and statics. These are basic tools for control algorithm and manipulator mechanism designers. All those algorithms are written in MATLAB as well as C code.

3.1. Inverse Kinematics (IK)

Inverse kinematics focus on finding the values of angles $(\theta_1, \theta_2, \theta_3)$ given the position of end-effector (x, y, z) in space frame. In this section, equations of IK are derived firstly and then performed in MATLAB.

3.1.1. IK Equations

From Figure 1.1, the vector-loop equations can be written as:

$$\mathbf{p} = \mathbf{R}_{1i} + \mathbf{L}_{1i} + \mathbf{L}_{2i} - \mathbf{R}_{2i}, \qquad i = 1, 2, 3 \tag{3.1}$$

$\mathbf{p}$ is the end-effector position:

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.2}$$

$\mathbf{R}_{1i}$ and $\mathbf{R}_{2i}$ are vectors in the upper and lower platform:

$$\mathbf{R}_{11} = \begin{bmatrix} R_1 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{R}_{12} = \begin{bmatrix} R_1 \cos 120^\circ \\ R_1 \sin 120^\circ \\ 0 \end{bmatrix}, \quad \mathbf{R}_{13} = \begin{bmatrix} R_1 \cos 240^\circ \\ R_1 \sin 240^\circ \\ 0 \end{bmatrix} \tag{3.3}$$

$$\mathbf{R}_{21} = \begin{bmatrix} R_2 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{R}_{22} = \begin{bmatrix} R_2 \cos 120^\circ \\ R_2 \sin 120^\circ \\ 0 \end{bmatrix}, \quad \mathbf{R}_{23} = \begin{bmatrix} R_2 \cos 240^\circ \\ R_2 \sin 240^\circ \\ 0 \end{bmatrix} \tag{3.4}$$

$\mathbf{L}_{1i}$ are vectors of the lower legs:

$$\mathbf{L}_{11} = \begin{bmatrix} L_1 \sin\theta_1 \\ 0 \\ L_1 \cos\theta_1 \end{bmatrix}, \quad \mathbf{L}_{12} = \begin{bmatrix} L_1 \sin\theta_2 \cos 120^\circ \\ L_1 \sin\theta_2 \sin 120^\circ \\ L_1 \cos\theta_2 \end{bmatrix}, \quad \mathbf{L}_{13} = \begin{bmatrix} L_1 \sin\theta_3 \cos 240^\circ \\ L_1 \sin\theta_3 \sin 240^\circ \\ L_1 \cos\theta_3 \end{bmatrix} \tag{3.5}$$

$\mathbf{L}_{2i}$ are vectors of the upper legs, whose length is a constant L_2. Square equation 3.1 and shift the terms on the right-hand side:

$i = 1$:

$$\begin{aligned} 0 =& {L_2}^2 - {L_1}^2 - [(x + R_2 - R_1)^2 + y^2 + z^2] \\ &+ 2zL_1 \cos\theta_1 + 2L_1[x + R_2 - R_1] \sin\theta_1 \end{aligned} \tag{3.6}$$

$i = 2$:

$$\begin{aligned} 0 =& {L_2}^2 - {L_1}^2 - [(x + \cos 120^\circ(R_2 - R_1)]^2 - [y + \sin 120^\circ(R_2 - R_1)]^2 \\ &- z^2 + 2zL_1 \cos\theta_2 + 2L_1[x \cos 120^\circ + (R_2 - R_1)\cos^2 120^\circ \\ &+ y \sin 120^\circ + (R_2 - R_1)\sin^2 120^\circ] \sin\theta_2 \end{aligned} \tag{3.7}$$

$i = 3$:

$$\begin{aligned} 0 =& {L_2}^2 - {L_1}^2 - [(x + \cos 240^\circ(R_2 - R_1)]^2 - [y + \sin 240^\circ(R_2 - R_1)]^2 \\ &- z^2 + 2zL_1 \cos\theta_3 + 2L_1[x \cos 240^\circ + (R_2 - R_1)\cos^2 240^\circ \\ &+ y \sin 240^\circ + (R_2 - R_1)\sin^2 240^\circ] \sin\theta_3 \end{aligned} \tag{3.8}$$

Notice equation 3.6-3.8 that they are all in the form $E_i \cos\theta_i + F_i \sin\theta_i + G_i$=0. This equation contains $\sin\theta_i$ and $\cos\theta_i$ together and usually it can be solved by Tangent Half-Angle Substitution.[**1**]

Tangent Half-Angle Substitution:

$$\cos\theta_i = \frac{1 - \tan^2\frac{\theta_i}{2}}{1 + \tan^2\frac{\theta_i}{2}} \qquad \sin\theta_i = \frac{2\tan\frac{\theta_i}{2}}{1 + \tan^2\frac{\theta_i}{2}} \tag{3.9}$$

Substitute them into $E_i \cos\theta_i + F_i \sin\theta_i + G_i$=0:

$$\begin{aligned} & E_i \frac{1 - \tan^2\frac{\theta_i}{2}}{1 + \tan^2\frac{\theta_i}{2}} + F_i \frac{2\tan\frac{\theta_i}{2}}{1 + \tan^2\frac{\theta_i}{2}} + G_i = 0 \\ & (G_i - E_i)\tan^2\frac{\theta_i}{2} + 2F_i \tan\frac{\theta_i}{2} + (G_i + E_i) = 0 \\ & \tan\frac{\theta_i}{2} = \frac{-F_i \pm\sqrt{E_i^2 + F_i^2 - G_i^2}}{G_i - E_i} \end{aligned} \tag{3.10}$$

So

$$\theta_i = 2\arctan\left(\frac{-F_i \pm\sqrt{E_i^2 + F_i^2 - G_i^2}}{G_i - E_i}\right) \tag{3.11}$$

From equation 3.11, we can get two solutions and both of them are mathematically correct. But in practical situations, one possible solution can be chosen with some given constraints. In the code the constrain of the angle is set from -45° to 90°.

3.1.2. IK Example in Matlab

`[thetalist,S] = DeltaIkin( R1,R2,L1,L2,p )`
The function calculates the inverse kinematics given parameters of Delta platform and position of end-effector in space frame. If the input parameters is valid, then `S` returns 1, otherwise returns 0.

For example, when `R1 = 0.2, R2 = 0.1, L1 = 0.2, L2 = 0.3` and the position `p = [0,0,0.3]`, the outputs are `thetalist = [0.8427, 0.8427, 0.8427]`, `S = 1`. And the calculation results can be used to plot the 3D configuration, shown as Figure 3.1.

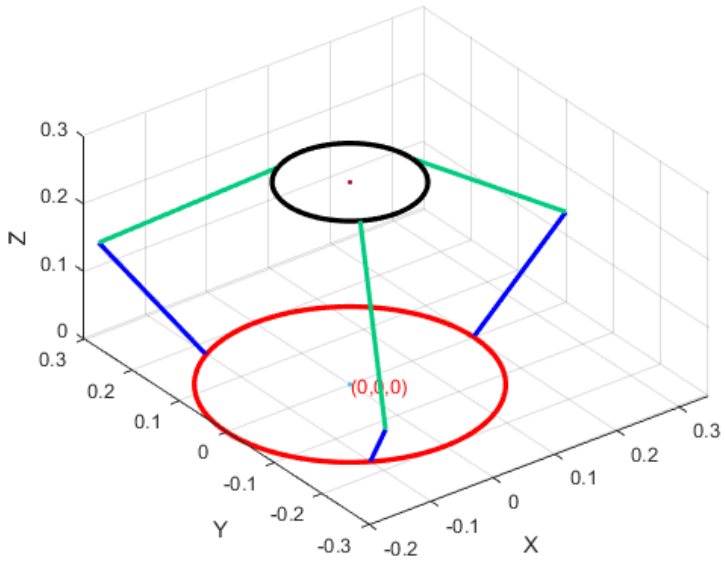

Figure 3.1. 3D plot of the configuration in the IK case

3.2. Forward Kinematics (FK)

Forward kinematics focus on finding the end-effector position (x, y, z) given joint angles $(\theta_1, \theta_2, \theta_3)$. Different from inverse kinematics that can be solved from geometric properties, forward kinematics can only be solved by numerical method. In this section, equations of FK and Newton-Raphson method are listed firstly and then performed in MATLAB.

3.2.1. FK Equations

Given joint angles $(\theta_1, \theta_2, \theta_3)$, vectors of lower legs $\mathbf{L}_{1i}$ $(i = 1, 2, 3)$ can be determined from equation 3.5. Vectors of upper and lower platform are known from equation 3.3 and 3.4.

And vector of end-effector $\mathbf{p}= (x, y, z)^T$ is unknown. So the equation of upper legs $\mathbf{L}_{2i}$ $(i = 1, 2, 3)$ can be written as:

$$\mathbf{L}_{2i} = \mathbf{p} - \mathbf{R}_{1i} + \mathbf{R}_{2i} - \mathbf{L}_{1i} \qquad i = 1, 2, 3 \tag{3.12}$$

The length of upper legs is known, so the equation of FK can be written as:

$$L_2^2 = (\mathbf{p} - \mathbf{R}_{1i} + \mathbf{R}_{2i} - \mathbf{L}_{1i})^T(\mathbf{p} - \mathbf{R}_{1i} + \mathbf{R}_{2i} - \mathbf{L}_{1i}) \qquad i = 1, 2, 3 \tag{3.13}$$

Equation 3.13 contains 3 independent linear equations and 3 unknown parameters x, y, z, thus the FK problem reduces to solving those equations through numerical method.

3.2.2. Newton-Raphson Method

The equations of forward kinematics could be written as equation $f(x) = 0$, where $f : \mathbb{R}^3 \to \mathbb{R}^3$. Assume x_0 is the initial guess to the solution. Then write first-order Taylor expansion of $f(x)$:

$$f(x) = f(x_0) + \frac{\partial f}{\partial x}(x_0)(x - x_0) \tag{3.14}$$

Where x and $f(x)$ are both 3×1 vectors, and $\dfrac{\partial f(x)}{\partial x}$ is a 3×3 matrix:

$$\frac{\partial f(x)}{\partial x} = \begin{pmatrix} \dfrac{\partial f_1(x)}{\partial x_1} & \dfrac{\partial f_1(x)}{\partial x_2} & \dfrac{\partial f_1(x)}{\partial x_3} \\ \dfrac{\partial f_2(x)}{\partial x_1} & \dfrac{\partial f_2(x)}{\partial x_2} & \dfrac{\partial f_2(x)}{\partial x_3} \\ \dfrac{\partial f_3(x)}{\partial x_1} & \dfrac{\partial f_3(x)}{\partial x_2} & \dfrac{\partial f_3(x)}{\partial x_3} \end{pmatrix} \tag{3.15}$$

Solve equation 3.14 for x:

$$x = x_0 - \big(\frac{\partial f}{\partial x}(x_0)\big)^{-1} f(x_0) \tag{3.16}$$

This x could be used as a new guess of the equation. So the iteration equation can be written as:

$$x_{k+1} = x_k - \big(\frac{\partial f}{\partial x}(x_k)\big)^{-1} f(x_k) \tag{3.17}$$

Iterate the x until a the stopping criterion is satisfied: $|f(x_{k+1} - f(x_k)| \leq \epsilon$ for some user defined small value ϵ.

3.2.3. FK Example in MATLAB

```
p = DeltaFkin( R1,R2,L1,L2,thetalist,p0 )
```

This function calculates the forward kinematics given parameters Delta platform, three angles of R joints and the initial guess of the position of end-effector.

For example, when `R1 = 0.2, R2 = 0.1, L1 = 0.2, L2 = 0.3`, and joint angles `thetalist = [pi/6, pi/6, pi/6], p0 = [0,0,0.5]`. The output is `p = [0,0,0.3968]`. And the 3D plot of this case is shown in Figure 3.2.

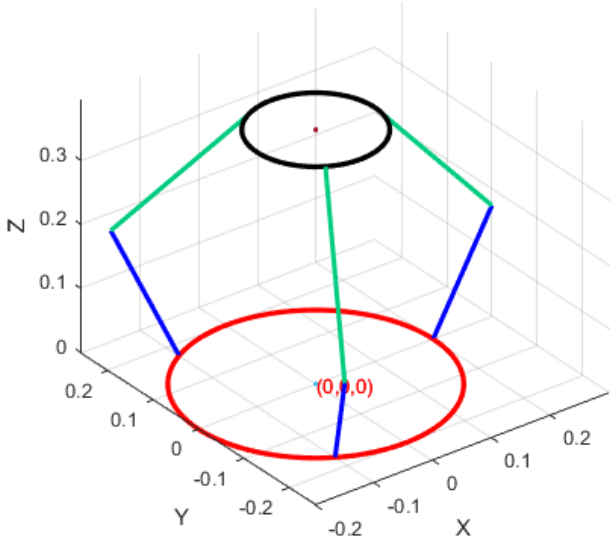

Figure 3.2. 3D plot of the configuration in the FK case

3.3. Jacobian

3.3.1. Equations

To find the 3×3 Jacobian of the delta mechanism, taking derivatives of equation 3.6-3.8 is recommended[**2**]:

$i = 1$:

$$\begin{aligned} 0 = &-2[(x + R_2 - R_1)\dot{x} + y\dot{y} + z\dot{z}] + 2\dot{z}L_1 \cos\theta_1 - 2zL_1 \sin\theta_1\dot{\theta}_1 + 2L_1\dot{x}\sin\theta_1 \\ &+ 2L_1(x + R_2 - R_1)\cos\theta_1\dot{\theta}_1 \end{aligned} \tag{3.18}$$

$i = 2$:

$$\begin{aligned} 0 = &-2[(x + \cos 120^\circ(R_2 - R_1)]\dot{x} - 2[y + \sin 120^\circ(R_2 - R_1)]\dot{y} - 2z\dot{z} + 2\dot{z}L_1\cos\theta_2 \\ &- 2zL_1\sin\theta_2\dot{\theta}_2 + 2L_1(\dot{x}\cos 120^\circ + \dot{y}\sin 120^\circ)\sin\theta_2 + 2L_1[x\cos 120^\circ + \\ &(R_2 - R_1)\cos^2 120^\circ + y\sin 120^\circ + (R_2 - R_1)\sin^2 120^\circ]\cos\theta_2\dot{\theta}_2 \end{aligned} \tag{3.19}$$

$i = 3$:

(3.20)
$$\begin{aligned} 0 = &-2[(x + \cos 240^\circ(R_2 - R_1)]\dot{x} - 2[y + \sin 240^\circ(R_2 - R_1)]\dot{y} - 2z\dot{z} + 2\dot{z}L_1 \cos\theta_3 \\ &- 2zL_1 \sin\theta_3\dot{\theta_3} + 2L_1(\dot{x}\cos 240^\circ + \dot{y}\sin 240^\circ)\sin\theta_3 + 2L_1[x\cos 240^\circ \\ &+ (R_2 - R_1)\cos^2 240^\circ + y\sin 240^\circ + (R_2 - R_1)\sin^2 240^\circ]\cos\theta_3\dot{\theta_3} \end{aligned}$$

It can be written in the matrix form:

(3.21)
$$\mathbf{A}\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix} = \mathbf{B}\begin{bmatrix}\dot{\theta_1}\\ \dot{\theta_2}\\ \dot{\theta_3}\end{bmatrix}$$

where:

(3.22)
$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{21} & a_{22} & a_{32} \\ a_{31} & a_{23} & a_{33} \end{bmatrix}$$

$$\begin{aligned} a_{11} &= x + R_2 - R_1 - L_1\sin\theta_1 \\ a_{21} &= x + \cos 120^\circ(R_2 - R_1) - L_1\cos 120^\circ\sin\theta_2 \\ a_{31} &= x + \cos 240^\circ(R_2 - R_1) - L_1\cos 240^\circ\sin\theta_3 \\ a_{12} &= y \\ a_{22} &= y + \sin 120^\circ(R_2 - R_1) - L_1\sin 120^\circ\sin\theta_2 \\ a_{32} &= y + \sin 240^\circ(R_2 - R_1) - L_1\sin 240^\circ\sin\theta_3 \\ a_{13} &= z - L_1\cos\theta_1 \\ a_{23} &= z - L_1\cos\theta_2 \\ a_{33} &= z - L_1\cos\theta_3 \end{aligned}$$

(3.23)
$$\mathbf{B} = \begin{bmatrix} b_1 & 0 & 0 \\ 0 & b_2 & 0 \\ 0 & 0 & b_3 \end{bmatrix}$$

$$
\begin{aligned}
b_1 =& -zL_1\sin\theta_1 + (x + R_2 - R_1)L_1\cos\theta_1 \\
b_2 =& -zL_1\sin\theta_2 + [x\cos 120^\circ + (R_2 - R_1)\cos^2 120^\circ + y\sin 120^\circ \\
&+ (R_2 - R_1)\sin^2 120^\circ]\cos\theta_2 \\
b_3 =& -zL_1\sin\theta_3 + [x\cos 240^\circ + (R_2 - R_1)\cos^2 240^\circ + y\sin 240^\circ \\
&+ (R_2 - R_1)\sin^2 240^\circ]\cos\theta_3
\end{aligned} \tag{3.24}
$$

From the definition of Jacobian,

$$\mathbf{J} = \mathbf{A}^{-1}\mathbf{B} \tag{3.25}$$

3.3.2. Jacobian Example in Matlab

`J = DeltaJacobian( R1,R2,L1,L2,p )`
This function calculates the 3×3 Jacobian matrix given parameters Delta platform and the current position (x, y, z) of end-effector in space frame.

For example, when `R1 = 0.2, R2 = 0.1, L1 = 0.2, L2 = 0.3`, and end-effector position `p = [0,0,0.4]`. The output is

$$
\mathtt{J} = \begin{matrix} 0.1907 & -0.0953 & -0.0953 \\ 0.1907 & 0.1651 & -0.1651 \\ -0.0836 & -0.0836 & -0.0836 \end{matrix}
$$

3.4. Workspace

Workspace is a set of configurations of the end-effector that the Delta platform can reach when satisfying several constraints. Those constraints are labeled as $\theta_{lowerlimit}$, $\theta_{upperlimit}$, $\phi_{lowerlimit}$, $\phi_{upperlimit}$, as shown in Figure 1.1. The plot of workspace is generated by iteration method in four steps:

(1) Before the iteration process, a space that big enough containing the workspace should be decided. This big space includes hundreds or thousands of dots $\mathbf{p}_i = (x_i, y_i, z_i)$, describing the possible positions of end-effector that will be tested successively later.

(2) During a single loop of the iteration process, each dot $\mathbf{p}_i$ in the big space is plugged into the IK function successively to calculate out a set of joint angles $(\theta_{i1}, \theta_{i2}, \theta_{i3})$, $(\phi_{i1}, \phi_{i2}, \phi_{i3})$, which are angles of the lower and upper platform respectively.

(3) Also in the same loop, the values of joint angles $(\theta_{i1}, \theta_{i2}, \theta_{i3}), (\phi_{i1}, \phi_{i2}, \phi_{i3})$ are compared with the constraints $\theta_{lowerlimit}, \theta_{upperlimit}, \phi_{lowerlimit}, \phi_{upperlimit}$. If

$$\theta_{lowerlimit} \leqslant \theta_{ij} \leqslant \theta_{upperlimit} \ \ \& \ \ \phi_{lowerlimit} \leqslant \phi_{ij} \leqslant \phi_{upperlimit} \quad (j = 1, 2, 3)$$

is satisfied, the dot $\mathbf{p}_i$ will be recognized as an element of workspace and plotted out at the position (x_i, y_i, z_i).

(4) Repeat the step (2) (3) and remain each plotted dot, until all of the dots in the big space referred in step (1) are calculated and checked.

This 4-step workspace algorithm is used in both 2D and 3D workspace cases.

3.4.1. 2D Workspace

In the 2D case, the basic big space is a vertical plane. The parameters of the Delta platform are $R_1 = 0.18, R_2 = 0.062, L_1 = 0.2, L_2 = 0.31$, and the constraints are $\theta_{lowerlimit} = 10°, \theta_{upperlimit} = 80°, \phi_{lowerlimit} = 25°, \phi_{upperlimit} = 50°$ The workspace plot satisfying the constraints is shown in Figure 3.3.

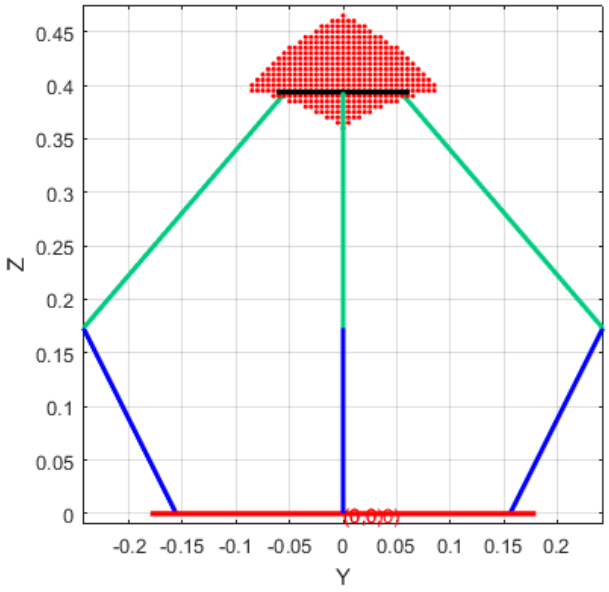

Figure 3.3. 2D workspace of Delta platform

3.4.2. 3D Workspace

In the 3D case, the basic big space is a cube in 3D space. The resolution of the 3D plot is lower than that of the 2D plot because of the compute efficiency consideration. The 3D workspace plot is shown in Figure 3.4.

3.5. Velocity and Force Limits

Since the 3×3 Jacobian matrix has been derived in previous sections, the velocity and force limits of the end-effector can be mapped through Jacobian from the limits of joint

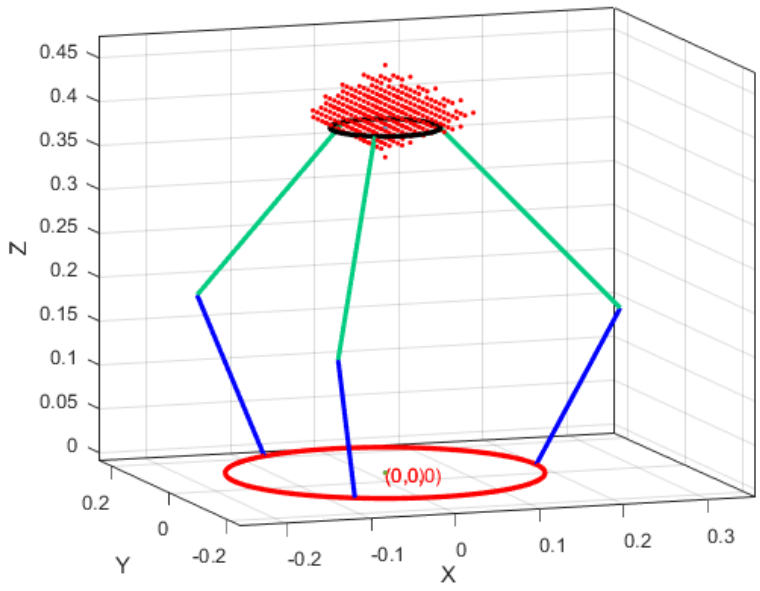

Figure 3.4. 3D workspace of Delta platform

velocities. Both the velocity and force limits of joints are represented as a unit sphere in θ_1-θ_2-θ_3 space, and limits of end-effector are mapped in x-y-z space.

3.5.1. Velocity Limits

In order to efficiently analyze the velocity limits of the end-effector, four points at the corner of the 2D workspace are selected as the representative points. Four velocity ellipsoids mapped from the unit circle through Jacobian are plotted at those points by the formula:

$$\dot{x} = J(\theta)\dot{\theta}$$

Where $\dot{\theta}$ is a 3×1 angular velocity vector of joints, and $\dot{x}$ is a 3×1 linear velocity vector of the end-effector. The plot of the velocity limits is shown in Figure 3.5. A piece of line near the ellipsoids representing a standard velocity serves as the mapping scale for estimating the value.

3.5.2. Force limits

Similar as the plot of velocity limits, the force ellipsoids are plotted out in the same way as the velocity ones by the formula:

$$f_{e\text{-}e} = J(\theta)^{-T}\tau$$

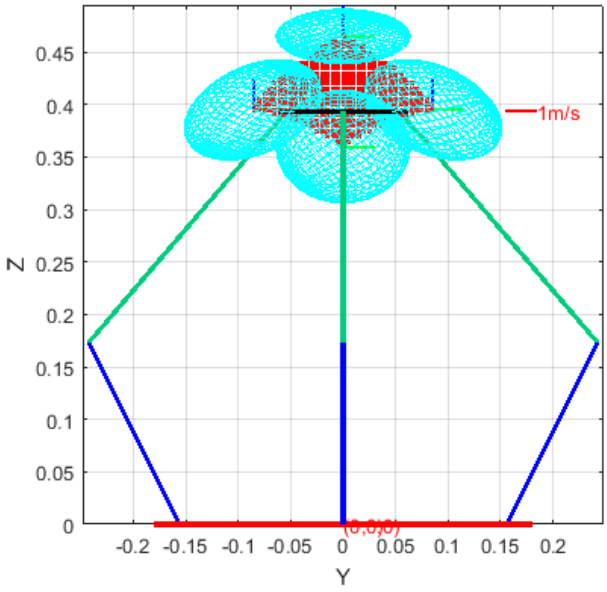

Figure 3.5. Velocity limits at representative points in workspace

Where τ is a 3×1 torque vector of joints, and $f_{e\text{-}e}$ is a 3×1 force vector of the end-effector. The The plot of the velocity limits is shown in Figure 3.6. There is also a piece of line representing standard force for estimation.

3.6. Stiffness

The three R joints connecting legs and the lower platform are designed as three torsional springs, and each spring connects a motor in series. Torque from a motor is transfered to leg through a torsional springs. Therefore, the angular positions of the torsional springs directly reflect the force and stiffness of the end-effector. Forth at the end-effector could be solved by Jacobian as previously mentioned, and stiffness of the platform is expressed by stiff matrix.

3.6.1. 3D Stiffness Matrix

The formula of stiff matrix follows Hooke's law

$$\begin{bmatrix}\Delta f_x\\ \Delta f_y\\ \Delta f_z\end{bmatrix} = \begin{bmatrix}k_{11} & k_{12} & k_{13}\\ k_{21} & k_{22} & k_{23}\\ k_{31} & k_{32} & k_{33}\end{bmatrix}\begin{bmatrix}\Delta x\\ \Delta y\\ \Delta z\end{bmatrix} \tag{3.26}$$

To calculate the stiffness matrix at a specific configuration, we exert a small displacement (0.001m in our example) to the end-effector, and $\Delta\mathbf{f}$ is known by calculating the difference of forces before and after the displacement. Then the 3×3 stiffness matrix could be

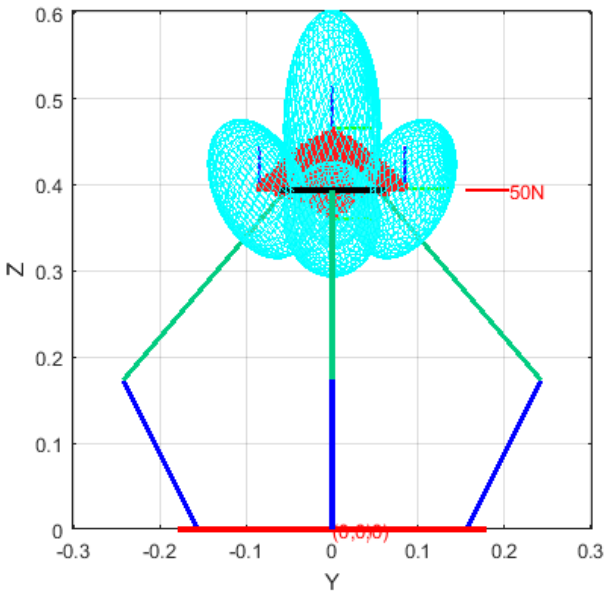

Figure 3.6. Force limits at representative points in workspace

determined by solving equation 3.26.
The corresponding function in MATLAB is `k_matrix = DeltaStiffnessXYZ(R1, R2, L1, L2, p, K, restangle, displacement)`. Where `K` is the stiffness (N/m) of torsional springs, `restangle` is the zero position of the platform, and `displacement` is 0.001m in our example.

3.6.2. 2D Stiffness matrix

Similar as analyzing the workspace, sometimes we are interested in the stiffness on Y-Z plane, and the 2×2 stiffness matrix can be calculated by

$$\begin{bmatrix} \Delta f_y \\ \Delta f_z \end{bmatrix} = \begin{bmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta z \end{bmatrix} \tag{3.27}$$

The method of solving equation 3.27 is same as that of equation 3.26.
The corresponding function in MATLAB is `k_matrix = DeltaStiffnessYZ(R1, R2, L1, L2, p, K, restangle, displacement)`.

3.6.3. Stiffness Ellipse

In order to directly reflect the end-effector stiffness of the platform at a particular position given the spring stiffness and the zero position, we plot out the ellipse according to the eigenvalues of its 2D stiffness matrix, shown as Figure 3.7. In this plot, the platform is

stiffer along z-axis and softer along x and y axis. There is also a reference length with value in the plot for estimation.

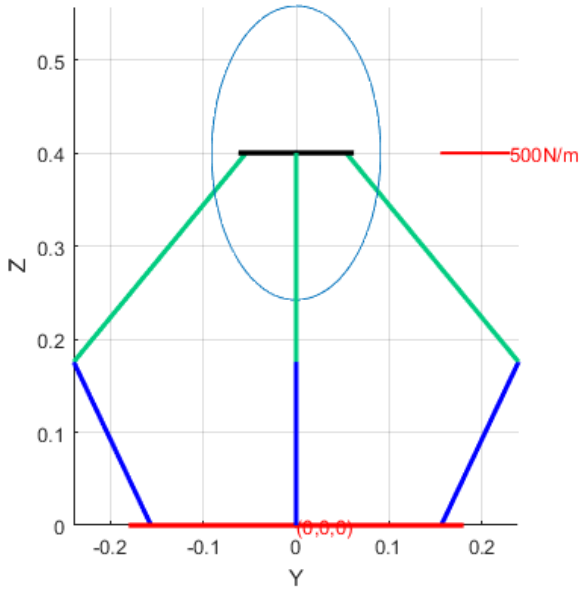

Figure 3.7. Stiffness ellipse at a particular position

CHAPTER 4

Control Strategy

The embedded control system is divided into two main parts: mobile manipulator controller (MMC) and joint controller (JC), shown as Figure 4.1. And MMC receives instructions from the central computer (CC) operated by user. Central computer is just an interface between user and controller, and does noK calculation. This chapter illustrates the structure and work flow of the control system.

4.1. Motion Control

Motion control of the Delta platform includes a series of calculation of inverse kinematics and feedback control of joint angles. User inputs the desired euclidean position (x_d, y_d, z_d) of end-effector, then the controller calculates the correspondingly desired joint angles $(\theta_{1d}, \theta_{2d}, \theta_{3d})$. And reaching the desired angles is performed separately by several PID feedback controllers.

If we want to perform a trajectory by the end-effector, we need input an array of end-effector positions varying with time. After calculation the array of joint angles varying with time, the joint controller will guide the joints move along reference angles.

4.2. Force Control

Since the motors are serially connect to the legs through torsion springs, the force control of joints can be finished by angular motion control of torsion springs. We are always interested in the linear force exerted by the end-effector, where the target object directly contacts with. User inputs the desired force $f_d = (f_x, f_y, f_z)$ of end-effector, and controller calculates the desired torques $\tau_d = (\tau_1, \tau_2, \tau_3)$ through the equation:

$$\tau_d = J(\theta)^T f_d$$

The force control should also consider the dead load of the upper platform and legs. So before the platform working exists a calibration step, where user moves the end-effector to the home position and controller records the zero angles of torsion springs $\theta_0 = (\theta_{1-0}, \theta_{2-0}, \theta_{3-0})$.

The home position of torsion springs θ_0 is known in the previous calibration step, then the controller just needs to calculate the desired angle θ_d by $\theta_d - \theta_0 = \tau_d/k$, where k is the stiffness of torsion springs. Then the PID feedback motion controller will make the spring move to the desired angle θ_d.

4.3. Structure of Control System

The structure of the mobile manipulator and communication protocol is shown in Figure 4.1. The embedded controller consists of four microcontrollers, i.e. NU32s. One of them serves as the mobile manipulator controller (MMC) and the other three serve as joint controller (JC). The MMC is connected to the central computer (CC) and three JCs separately.

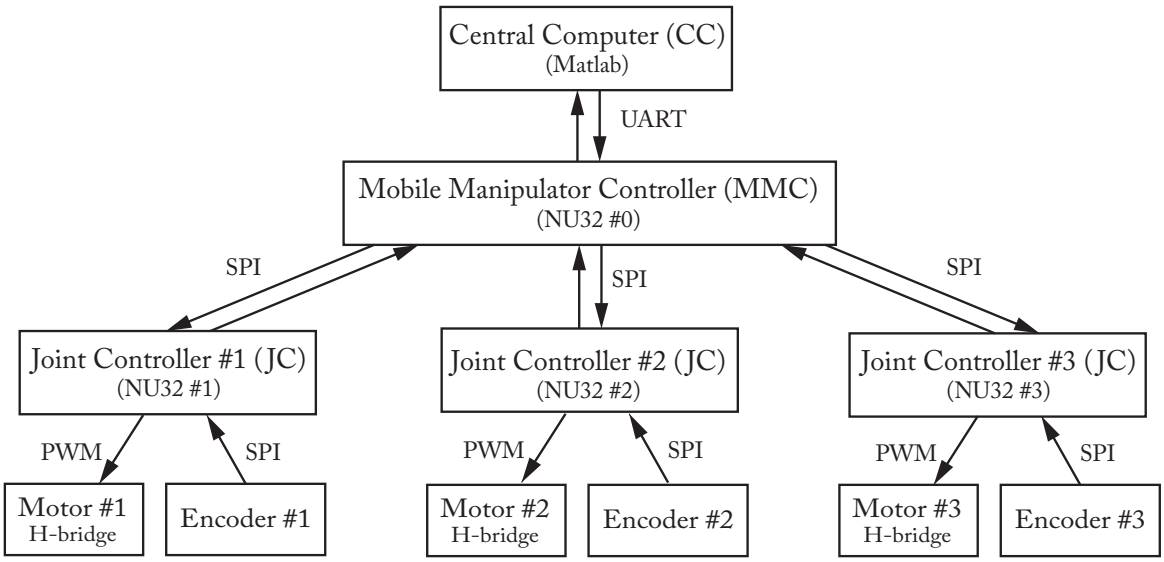

Figure 4.1. Structure and communication of control system

In this study, the MMC and JCs are NU32s that communicate through SPI protocol, where MMC is the SPI master and JCs are SPI slaves. MMC also communicates with CC through UART protocol.

JC adopts a nested PID loop to control the current and motion, and also incorporates functions of SPI communication.

4.4. Work Flow

Five steps are included in the motion control process, as shown in Figure 4.2:

(1) User inputs the desired trajectory of the end-effector of Delta mechanism as an array of positions $\mathbf{p}_d = (x_d, y_d, z_d)$.

(2) MMC receives the desired position of end-effector and apply the data into IK function to generate a list of motors' angles $(\theta_{1d}, \theta_{2d}, \theta_{3d})$ varying with time, and the they will be the reference angles in the next step.

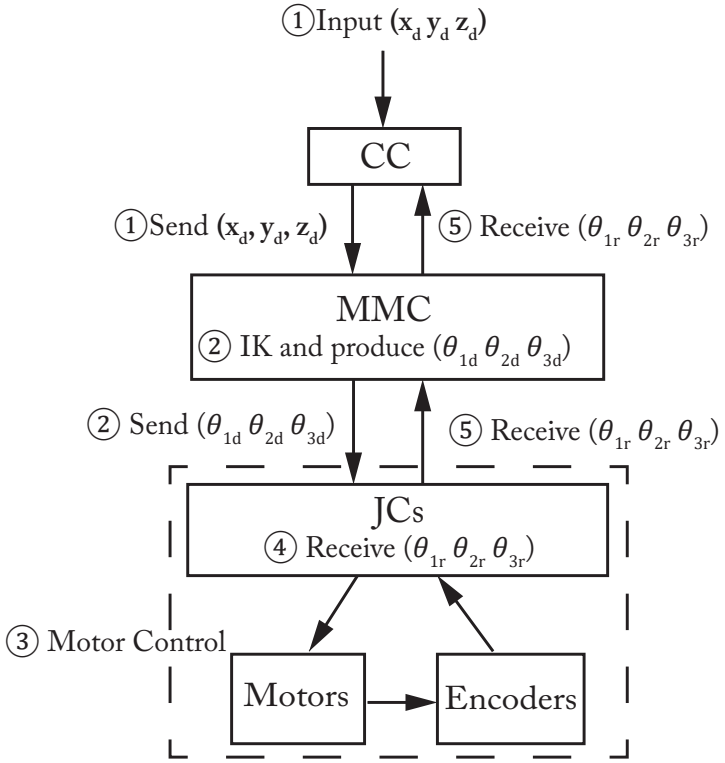

Figure 4.2. Work flow of control system

(3) MMC sends the reference angles to three JCs, and those JCs receive the angles as the reference to perform the control algorithm to manipulate three motors separately.

(4) Each JC receives the real-time angles of each corresponding motor $(\theta_{1r}, \theta_{2r}, \theta_{3r})$ while the motor is moving, and save the angles into an array. The array will be sent back to the MMC once finishing the motion.

(5) MMC receives the array of the real angles and sends the array to computer. The the computer will process and plot the data finally.

CHAPTER 5

SPI Communication

SPI (Serial Peripheral Interface) is a fast method of digital communication between devices. As shown in Figure 4.1, SPI is used for NU32-NU32 and NU32-encoder communication.

5.1. NU32-NU32 Communication

5.1.1. Connection Strategy

SPI is a master-slave architecture. In this case, MMC is the master and JCs are slaves. MMC connects with three JCs at the same time, while slave select ($\overline{\text{SS}}$) and slave check ($\overline{\text{SC}}$) pin are used simultaneously for activating each slave consecutively. The connection circuits and data flow direction are shown in Figure 5.1.

SPI3 is adopted in the NU32-NU32 communication. Three pins are included in each SPI communication for data transfer between one master and one slave: SDO (Serial Data Out), SDI (Serial Data In) and SCK (System Clock). Since the master can only communicate with one slave each time, slave select $\overline{\text{SS}}$ (active low) and slave check $\overline{\text{SC}}$ (valid low) pins are necessary to ensure that only one slave is active each time.

SS pins on master are general digital IO pins that output either 3.3V or 0V. $\overline{\text{SS}}$ pin on slave, D9 pin, is the specified control pin of SPI3. For a slave, When the input $\overline{\text{SS}}$ is high, SPI3 is turned off. When the input $\overline{\text{SS}}$ is low, SPI3 is active and able to transmit data.

For some practical reason, three digital IO pins on master and one IO pin on slave are configured as $\overline{\text{SC}}$ pins. Usually reading and writing IO pin for logical judgment is more convenient and precise in program. Therefore the $\overline{\text{SC}}$ as well as $\overline{\text{SS}}$ pin are working together in NU32-NU32 SPI communication.

5.1.2. Work Flow

Figure 5.2 5.3, and 5.4 illustrate the communication process between MMC and JCs while the SS pins and SC pins are working together. Three different types of communication are included in the protocol: MMC sends data to JCs, JCs send data to MMC, and communication while JC is performing task.

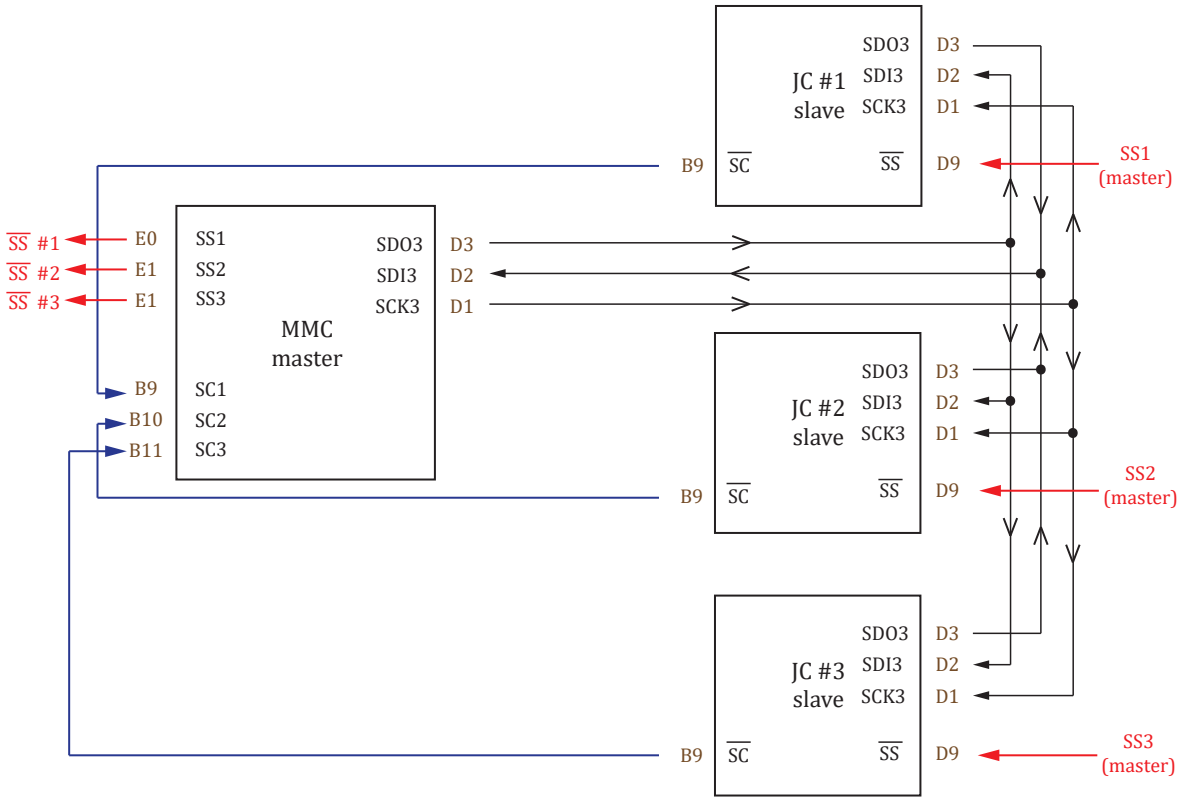

Figure 5.1. SPI communication diagram among NU32s. SS stands for "Slave Select". SC stands for "Slave Check". Pin numbers of NU32 are labeled brown.

5.2. NU32-Encoder Communication

The final design of the mobile manipulator includes 3 torsional springs and a 3-DOF gimbal mechanism, and both of them should be attached with high-precision encoders. Each spring attaches two 23-bit absolute encoder, and the gimbal attaches three 12-bit absolute encoders. Both of the encoders can be read through SPI communication on NU32.

5.2.1. 23-bit Spring Encoder

The AS38-H39E series encoder is a high-resolution optical absolute encoder produced by Broadcom, which offers 23-bit single-turn and 16-bit multi-turn counts, hence a combined 39-bit high resolution. For the Delta mechanism, the rotation angle of springs are limited within 360°, thus only the later 23-bit single-turn part is useful.

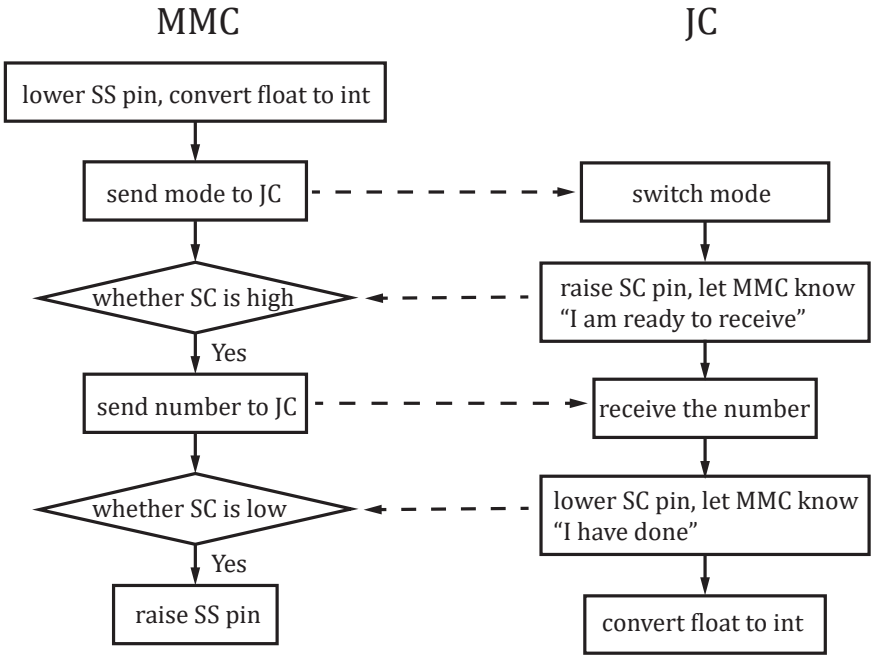

Figure 5.2. MMC sends data to JC

The encoder has multiple communication protocols, but only the SSI (Synchronous Serial Interface) interface protocol is adopted here. And a SN65LBC179 transceiver is necessary to read the encoder. The wiring diagram is shown in Figure 5.5.

In this case, SPI4 on NU32 connecting to transceiver receives the data from encoder. The received data should be a 39-bit binary number, where the last 23 bits are the desired single-turn counts, as illustrated in data sheet. NU32 has 16 and 32 bits transfer mode, corresponding with 16-bit and 32-bit data per sent, both less than 39 bits. So the 39-bit number should be divided into at least two parts to transmit, then combining together by bit shifting operation.

5.2.2. 12-bit Gimbal Encoder

The AEAT-6012 magnetic encoder is a 12-bit resolution encoder produced by AVAGO, which provides the angular detection within 360°. This encoder uses 3-wire SSI interface protocol, including $\overline{\text{CS}}$, SCK and DO. The default level into $\overline{\text{CS}}$ should be high, and the data transmitting operation can only be triggered once generating a falling edge in $\overline{\text{CS}}$.

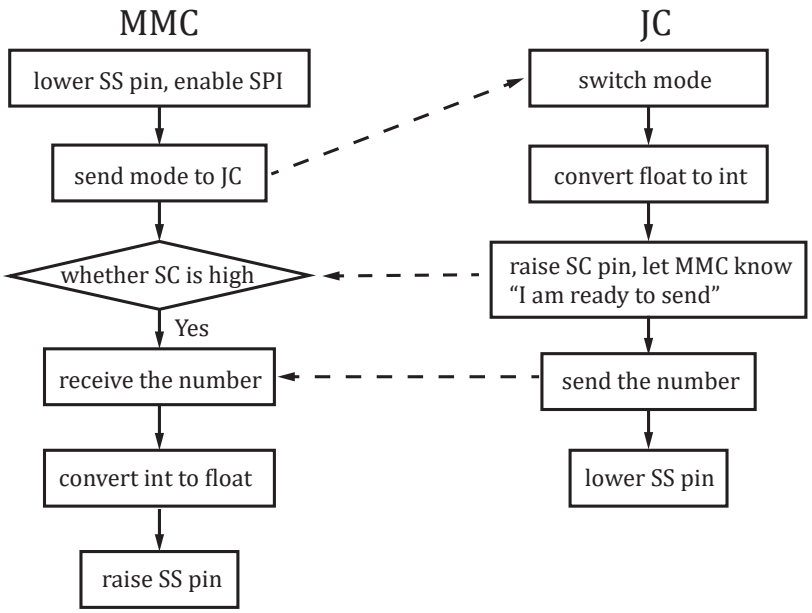

Figure 5.3. JC sends data to MMC

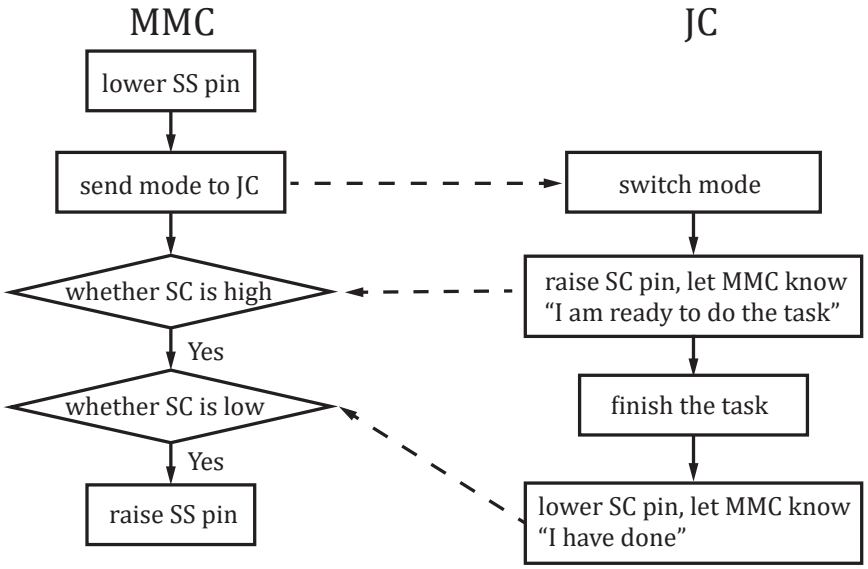

Figure 5.4. Communication while JC is performing task

The encoder can also be read by SPI communication in NU32. The operation of reading three gimbal encoders should be performed by MMC using one SPI channel. Wiring diagram is shown in Figure 5.6.

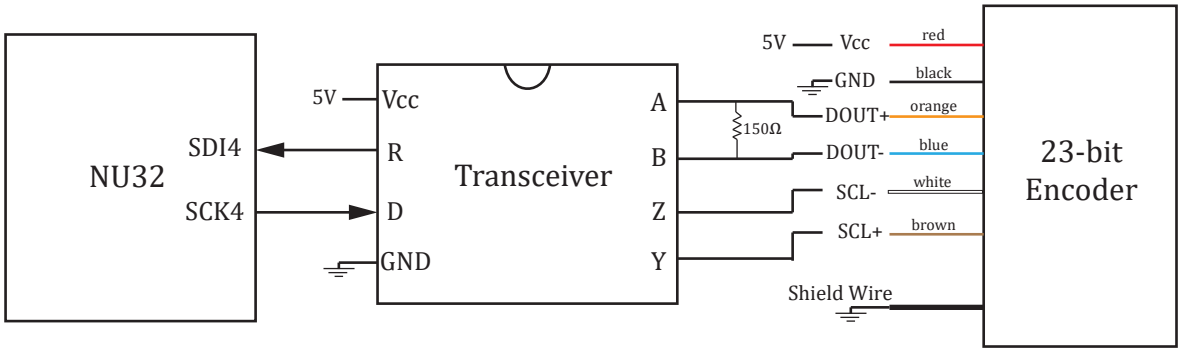

Figure 5.5. Wiring diagram of the 23-bit absolute encoder

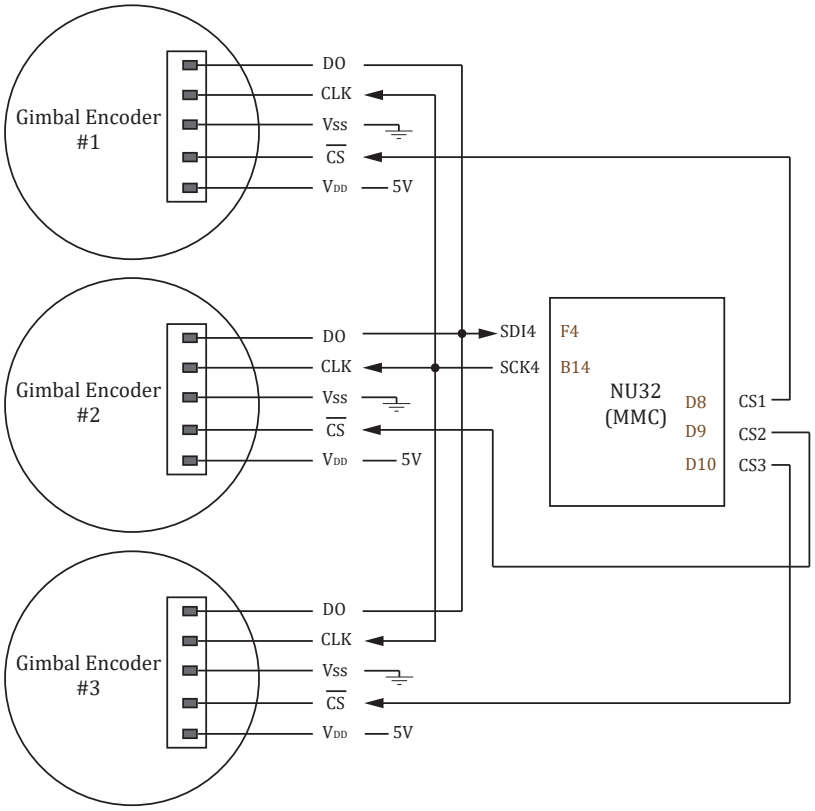

Figure 5.6. Wiring diagram of the 12-bit absolute encoder

The NU32 lowers one CS pin each time and reads three encoders consecutively with a high frequency, to obtain the real time revolve angle of the gimbal mechanism. The received angles of the gimbal will be sent to central computer.

CHAPTER 6

Circuit and Wiring

The JC in the demo is same as the motor controller of the ME333 final project. The circuit diagram of one brush motor controller, including H-bridge, current-sense amplifier, and decoder is shown in Figure 6.1.

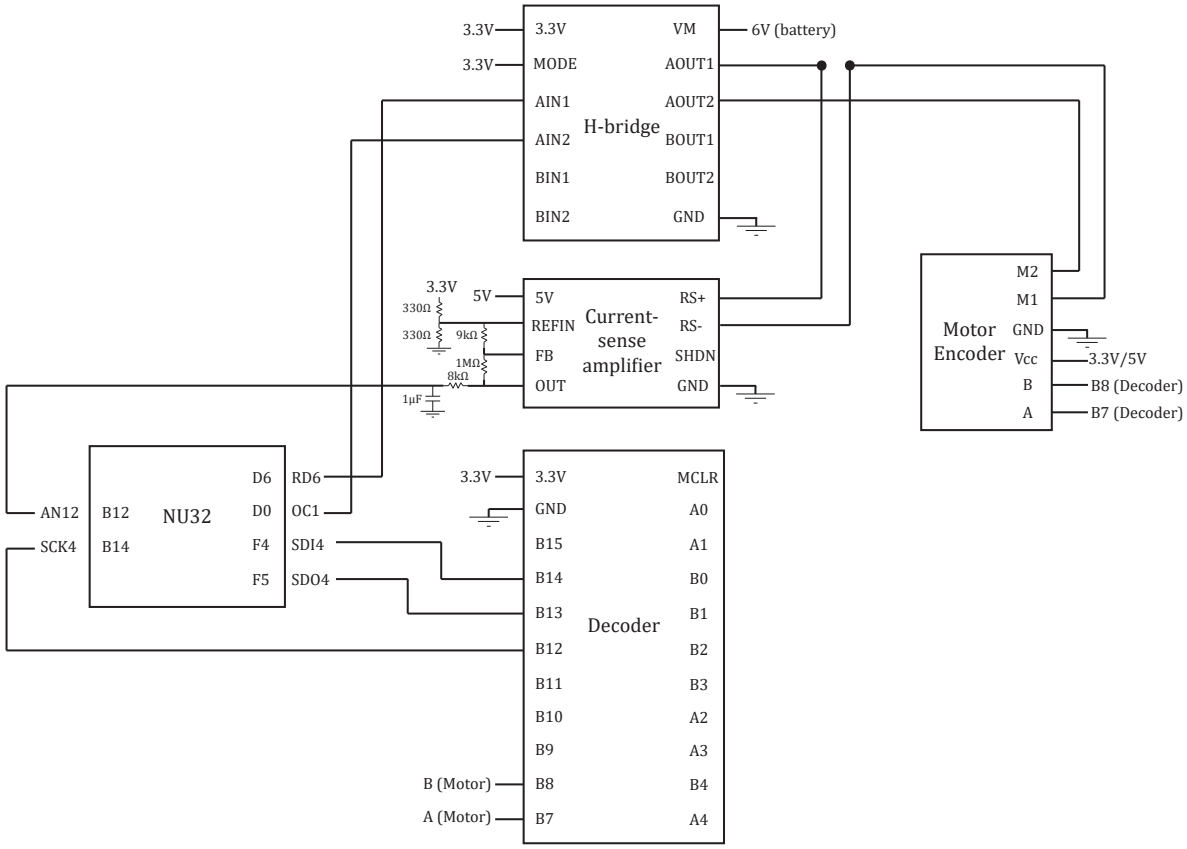

Figure 6.1. Circuit diagram of brush motor controller

CHAPTER 7

Software

The target of this project is to build a motor control system, including the interface in computer to transmit data. Users can monitor the state and manipulate the motion through the interface. Several operation modes and a Matlab interface are constructed to satisfy the demand.

7.1. Operation List

`a:Calibration.` Move manipulator to home position, and calculate the joint torques keeping the position. Then load the article and do the same thing. Now the MMC can estimate the weight of the article. These joint torques can be used as feedforward gravity compensation for controllers.

`b:Read gimbal encoders.` Print three gimbal encoder angles. The data are read directly by the MMC with 12 bits resolution.

`c:Reset gimbal encoders.` The present angles of the gimbal encoders are defined as the zero positions. No output exists in this mode but you can call command `b` after this to confirm the result.

`d:Set desired torques to 3 JCs.` Prompt for three desired torque, in Nm, that each JC should satisfy. The MMC will send them to each JC.

`e:Read current torques from 3 JCs.` Print three current torques from three JCs.

`f:Set stiffness of springs.` Prompt for stiffness of springs, in Nm/rad. And MMC will send it to JCs.

`g:Reset spring abs encoders.` The present angles of spring encoders are defined as the zero positions. No output exists in this mode but you can call command `h` after this to confirm the result.

`h:Read spring abs encoders.` Print six spring encoder angles. The data are read by JCs(each has two absolute encoders) with 23 bits resolution.

`i:Set geometry parameters.` Input parameters of Delta robot, including R_1, R_2, L_1 and L_2.

`j:Set PWM.` Prompt the user for three desired PWM duty cycle, specified in the range $[-100, 100]$. Each JC motor will switch to PWM mode and implements the constant

PWM. An input -100 means that the motor is full on in the clockwise direction, 100 means full on in the counterclockwise direction, and 0 means that the motor is unpowered.

`k:Reset incremental encoders.` The present angles of motor incremental encoders are defined as the zero positions. No output exists in this mode but you can call command `l` after this to confirm the result.

`l:Read incremental encoders.` Print three motor encoder angles. The data are read by JCs (each has one motor incremental encoder).

`u:Set control gains.` Set control gains, including current gains K_p, K_i and control gains K_p, K_i and K_d.

`p:Set JCs to IDLE Mode.` Unpower the motors. The JCs are set to IDLE mode.

`o:move end-effect to an position.` Prompt R1, R2, L1, L2, x, y, z from the user. R1, R2, L1, L2 are parameters of your mechanism and (x, y, z) is the desired position. The MMC will calculate the joint angles and send them to each JC. Then three motors will move to the specified angle immediately and hold the angle.

`n:load cubic trajectory.` Prompt for the time and position values describing a series of one or more steps in the end-effector position. Input format is shown below:

```
Enter cubic trajectory of x:  [time1, x1; time2, x2; ...]
Enter cubic trajectory of y:  [time1, y1; time2, y2; ...]
Enter cubic trajectory of z:  [time1, z1; time2, z2; ...]
```

The maximum value of time is 9 seconds. Then the data will be sent to and store in MMC for IK calculation and later plotting.

`y:Set JCs to TRACK Mode.` Execute trajectory stored in JCs. This command set JCs to TRACK mode, and JCs attempts to track the reference trajectory previously stored in JCs. After finishing the trajectory, JCs switch to HOLD mode. Please wait until the motion is finished.

`r:Read actual anglelist data from JCs.` Get data of TRACK mode. After TRACK mode, three actual angle lists are stored in JCs, and three reference angle lists are stored in MMC. In this mode, all these data are sent to MATLAB for plotting.

7.1.1. MATLAB Interface

MATLAB is able to open serial ports, allowing communication with NU32 and plotting data. Following is the MATLAB interface menu:

```
 a:  Calibration                          b:  Read gimbal encoders
 c:  Reset gimbal encoders                d:  Set desired torques to 3 JCs
 e:  Receive current torques from 3 JCs   f:  Set Stiffness to 3 JCs
 g:  Reset spring abs encoders            h:  Read spring abs encoders
 i:  Set geometry parameters              j:  Set PWM
 k:  Reset incremental encoders           l:  Read incremental encoders
 u:  Set control gains                    p:  Set JCs to IDLE mode
 o:  Move end-effect to an position       n:  Load cubic trajectory
 y:  Set JCs to TRACK mode                r:  Read actual anglelist data from JCs
 q:  Quit client
ENTER COMMAND:
```

User can type in the single character, which may lead the user to input more information. All of the operations in the control system are accessible through the MATLAB interface. For example, a user types in command `o`, then the following command in MATLAB asks user to input the desired values of x, y and z that indicate the target end-effector position. Then the microcontrollers manipulate the motor to rotate following the angles that generated from IK functions in MMC. Once finishing the entire motion process, MATLAB will receive the real motion data from MMC. And the user can continue the next step of data processing.

7.1.2. Operating Modes in C

The following list is the operating modes that necessary for JCs and MMC. The detailed explanations are also included in the comments of C files. Since spring encoders did not exist when constructing the software, the modes `0x01000004` to `0x01000006` are commented out in C files.

`0x01000000` JC receives a desired torque from MMC.

`0x01000001` JC Sends current torque to MMC.

`0x01000002` JC receives a nominal torque from MMC.

`0x01000003` JC receives stiffness from MMC.

`0x01000004` JC Sends data of the first spring encoder to MMC.

`0x01000005` JC Sends data of the second spring encoder to MMC.

`0x01000006` Reset two spring encoders.

`0x01000007` JC receives PWM from MMC.

`0x01000008` Resent incremental encoder.

`0x01000009` JC Sends data of incremental encoder to MMC.

`0x01000010` Set motor mode to IDLE.

`0x01000011` Set motor mode to HOLD.

`0x01000012` JC receives the length of the array that contains the angle track of motors from MMC.

`0x01000013` JC receives the the array that contains the angle track of motors from MMC.

`0x01000014` Set motor mode to TRACK.

`0x01000015` JC sends the array that contains the real angle tracks to MMC.

`0x01000016` Set K_p of current controller.

`0x01000017` Set K_i of current controller.

`0x01000018` Set K_p of position controller.

`0x01000019` Set K_i of position controller.

`0x01000020` Set K_d of position controller.

CHAPTER 8

Conclusion and Future Work

In this paper we found the analytical solution to inverse and velocity kinematics as well as the iterative solution to forward kinematics of Delta platform, and also put forward motion and force control strategy for the mechanism. We tested our algorithms on both MATLAB software in mechanism design and on C code in embedded controllers. The kinematic functions work well in both the parameter optimization in design and PID real-time control, which testified the reliability and efficiency of our codes. Besides, we also figured out the usage and protocols of encoders, and incorporated the encoders into our embedded control system.

We are still solving the dynamic problems that can contribute to the feed-forward force control. Our research is a valuable attempt of the mobile manipulator and its corresponding control system. In the future, the control algorithms and codes will be applied over other controllers other than NU32. The Delta mechanism will also be connected with the mobile base, and multiple mobile manipulators will cooperatively work with each other.

References

[1] Robert L. Williams II, The Delta Parallel Robot: Kinematics Solutions, Mechanical Engineering, Ohio University, Oct. 2016

[2] Y. Zheng, Q. Li, X. Zhang, Y. Luo, Y. Zhang, and S. Xie, Six-degree-of-freedom quasi-zero-rigidity vibration isolation system based on Stewart platform. CN patent CN105041961A, July 2015

[3] D. Malchev, and H. Veerkamp, Shock-isolation structure. US patent US9068622B2, Nov. 2009

[4] H. Tari, H. Su, and J. Hauenstein, Classification and complete solution of the kineto-statics of a compliant Stewart-Gough platform, Mechanism and Machine Theory 49 (2012) 177-186

[5] Y. Moon, C.D. Crane III, and R.G. Roberts, Reverse kinetostatic and stiffness of a spatial tensegrity-based compliant mechanism, Mechanism and Machine Theory 70 (2013) 320-337

[6] Y. Li, J. Wang, L, Wang, Stiffness Analysis Of A Stewart-Platform Based Parallel Kinematic Machine, International Conference on Robotics and Automation (ICRA) 2002