A brief guide to the Falcon robot by Novint

G. Aiello and G. Rauter*

***Bio Inspired RObotics in Medicine Lab**, University of Basel, Basel*

October 2016

E-mail: gregorio.aiello@unibas.ch

Abstract

In this report are described the main properties of the Falcon robot By Novint (shown in Fig. 1). First the architecture of the robot is introduced as presented by the inventor Lung-Wen Tsai in his paper [1] , then both the forward and inverse kinematics are covered. The characterization is briefly presented with information about the calibration, the performances, and the workspace as presented by S. Martin [2] . The last part is a tutorial on how to interface the Falcon with your computer and how to use the Falcon API to design your own software.

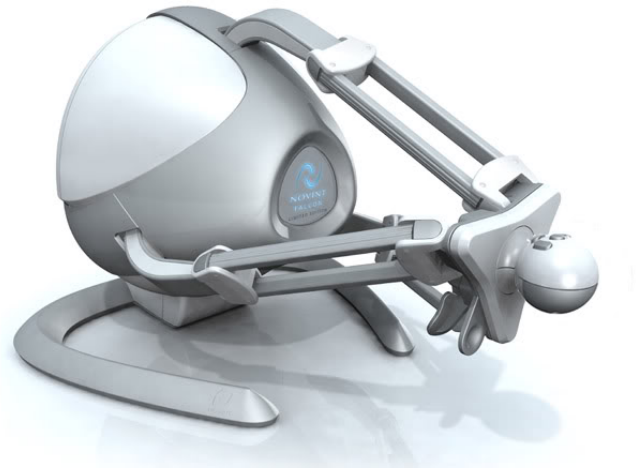

Figure 1: The Falcon robot by Novint

Architecture

The Falcon robot has three degrees of freedom and the moving platform is constrained to only translational motion. The main advantages of this parallel manipulator are that all of the actuators can be attached directly to the base (the robot is presented in Fig. 2), closed-form solutions are available for the forward kinematics, the moving platform maintains the same orientation throughout the entire workspace, and it can be constructed with only revolute joints. Closed-form solutions for both the forward and inverse kinematics problems are presented. It is shown that the inverse kinematics problem has up to four real solutions, and the forward kinematics problem has up to sixteen real solutions.

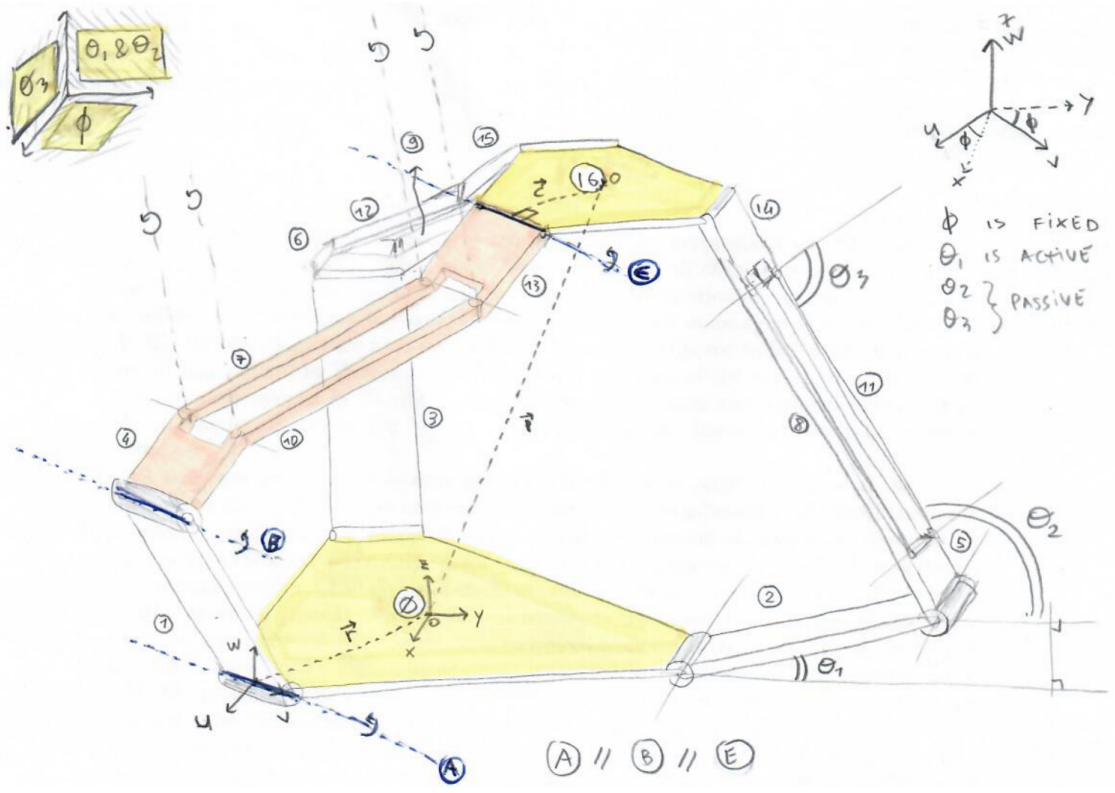

Figure 2: Schematic drawing of the Falcon's architecture

A reference frame **(XYZ)** is attached to the fixed base at point (∅), located at the center of the fixed platform, the x and y axes lie in the same plane as defined by the axes of A_1, A_2, and A_3 (the axis of the active revolute joints attached to the fixed base). The angle ϕ_i

for the i^{th} leg ($i = 1, 2, 3$) as shown in Fig. 2.1 defines the angular orientation of the leg relative to the **(XYZ)** frame on the fixed platform. Another coordinate system $\mathbf{(U}_i\mathbf{V}_i\mathbf{W}_i\mathbf{)}$ is attached to the fixed base at A_i for each leg, such that the u_i-axis is perpendicular to the axis of rotation of the joint at A_i and at an angle ϕ_i from the x-axis, while being in the plane of the fixed platform. The v_i-axis is along the joint axis of A_i (as presented in Fig. 2).

The vector $\bar{p}$ is the position vector of point **P** in the **(XYZ)** coordinate frame, where **P** is attached at the center of the moving platform (point (16) in Fig. 2).

The general mobility of the mechanism is found using the Grubler-Kutzbach equation:[3]

$$\mathbf{F} = \lambda(n - j - 1) + \sum_{i=1}^{j} f_i \tag{1}$$

with:

- **F**: degrees of freedom of the mechanism
- λ: motion parameter, for spatial mechanisms λ=6, for planar and spherical mechanisms λ=3, and for planar prismatic mechanisms λ=2. In our case obviously λ=6.
- n: number of links. In our case $n = (5 \times 3) + 2 = 17$
- j: number of joints. In our case $j = 7 \times 3 = 21$
- f_i: degrees of freedom associated with the j^{th} link, we use rotational joints only thus in our case $\forall i,\ f_i = 1$

Finally our general mobility is:

$$\mathbf{F} = 6(17 - 21 - 1) + \sum_{i=1}^{21} 1 = -30 + 21 = -9 \tag{2}$$

Negative general mobility indicates a statically indeterminate (over-constrained) structure, however due to the arrangement of the links and joints, many of the constraints imposed by the joints are redundant and the resulting mechanism does have three translational degrees

of freedom. Observe that the three revolute joints at $\mathbf{A}_i$, $\mathbf{B}_i$, and $\mathbf{E}_i$ have parallel axes (for the same i^{th} leg), hence the i^{th} leg provides two constraints on the rotation of the moving platform about the z and u_i axes, the latter means that two legs are enough to rotationally lock the moving platform. This leaves the mechanism with three translational degrees of freedom.

Inverse Kinematics

The inverse kinematics aims to obtain the joint variables starting from the position of the moving platform in Cartesian coordinates, in a formal way:

$$f^{-1} : \bar{p} \mapsto \bar{\theta} \tag{3}$$

$$\bar{p}_i = [p_{xi}, p_{yi}, p_{zi}]^T \tag{4}$$

$$\bar{\theta}_i = [\theta_{1i}, \theta_{2i}, \theta_{3i}]^T \tag{5}$$

$$i = 1, 2, 3 \tag{6}$$

First we transform the position of $\mathbf{P}$ from $\mathbf{(XYZ)}$ to the $\mathbf{(U}_i\mathbf{V}_i\mathbf{W}_i\mathbf{)}$ coordinates system (graphic interpretation in Fig. 2):

$$\tilde{p}_i = \begin{pmatrix} \cos\phi_i & -\sin\phi_i & 0 \\ \sin\phi_i & \cos\phi_i & 0 \\ 0 & 0 & 1 \end{pmatrix} \bar{p}_i + \begin{pmatrix} -r \\ 0 \\ 0 \end{pmatrix}$$

The meaning of $\tilde{p}_i = [p_{ui}, p_{vi}, p_{wi}]^T$ is shown in Fig. 3, it is convenient to use this formulation since ϕ and r are both fixed. The coordinates of the point (16) expressed in the $\mathbf{(U}_i\mathbf{V}_i\mathbf{W}_i\mathbf{)}$ coordinates system are:

$$\begin{cases} p_{ui} = a\cos\theta_{1i} + [d + e + b\sin\theta_{3i}]\cos\theta_{2i} - c \\ p_{vi} = b\cos\theta_{3i} \\ p_{wi} = a\sin\theta_{1i} + [d + e + b\sin\theta_{3i}]\sin\theta_{2i} \end{cases}$$

From the equation of p_{vi} we can immediately find 2 solutions for θ_{3i}:

$$\theta_{3i} = \pm\cos^{-1}\left(\frac{p_{vi}}{b}\right) \tag{7}$$

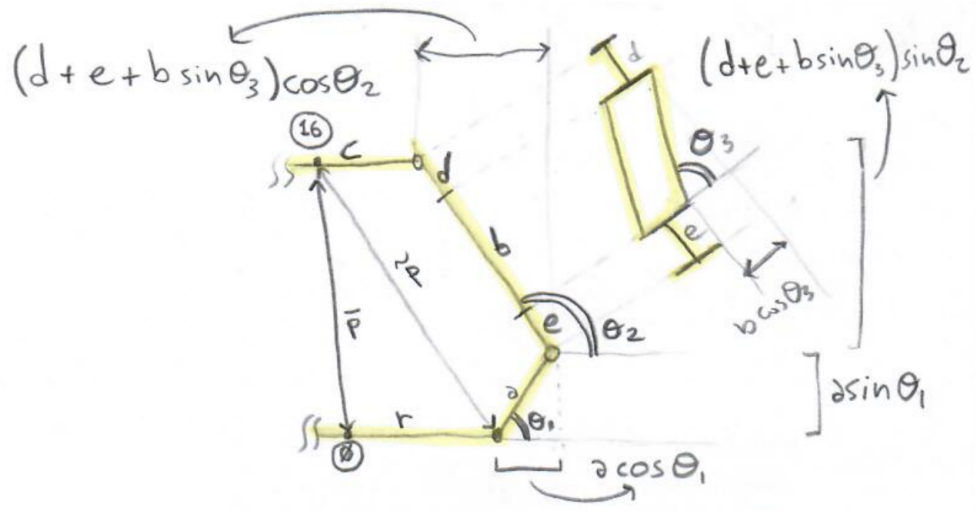

Figure 3: Schematic drawing of a generic leg of the Falcon.

$$\begin{cases} a\cos\theta_{1i} = [P_u + c - (d + e + b\sin\theta_{3i})\cos\theta_{2i}] \\ a\sin\theta_{1i} = [P_w - (d + e + b\sin\theta_{3i})\sin\theta_{2i}] \end{cases}$$

We can substitute $P_u + c = \hat{P}_u$ and $d + e + b\sin\theta_{3i} = k$, square the equations and sum them using the Pythagorean identity rule:

$$[\hat{P}_u - k\cos\theta_{2i}]^2 + [P_w - k\sin\theta_{2i}]^2 = a^2 \tag{8}$$

$$\hat{P_u}^2 + \underline{k^2\cos^2\theta_{2i}} - 2\hat{P}_u k\cos\theta_{2i} + P_w^2 + \underline{k^2\sin^2\theta_{2i}} - 2P_w k\sin\theta_{2i} = a^2 \tag{9}$$

$$\hat{P_u}^2 + P_w^2 + k^2 - 2k[\hat{P}_u\cos\theta_{2i} + P_w\sin\theta_{2i}] = a^2 \tag{10}$$

At this point we can use the parametrization of the trigonometric functions: $t = \tan\frac{\theta_2}{2}$, $\sin\theta_{2i} = \frac{2t}{1+t^2}$, and $\cos\theta_{2i} = \frac{1-t^2}{1+t^2}$. Again we can substitute $\hat{P_u}^2 + P_w^2 - a^2 = \hat{h}$ obtaining:

$$h + k^2 - 2k\left[\hat{P_u}\frac{1-t^2}{1+t^2} + P_w\frac{2t}{1+t^2}\right] = 0 \tag{11}$$

$$h(1+t^2) + k^2(1+t^2) - 2k\left[\hat{P_u}(1-t^2) + P_w(2t)\right] = 0 \tag{12}$$

$$t^2\left(h + k^2 - 2k\hat{P_u}\right) + t\left(4kP_w\right) + \left(h + k^2 - 2k\hat{P_u}\right) = 0 \tag{13}$$

$$at^2 + bt + c = 0 \tag{14}$$

The latter is a standard second order equation and it has 2 solutions of θ_{2i} for each of the two solutions of θ_{3i}, θ_{1i} is found backsubstituting θ_{2i} and θ_{3i} into the first equation, hence the number of solution for each leg is four.

Direct Kinematics

The objective of the forward kinematics solution is to define a mapping from the known set of the actuated joint angles to the unknown position of the moving platform ($f : \bar{\theta} \mapsto \bar{p}$). For this manipulator the joint angles that are considered known are the angles formed by the input links and the base of the manipulator, θ_{11}, θ_{12}, and θ_{13}. The unknown position of the moving platform is described by the position vector $\bar{p}$, which defines the location of **P** at the center of the moving platform in the **XYZ** coordinate frame. The three approaches used to solve the direct kinematics of a parallel manipulator are the **Dialytic Elimination**, the **Polynomial Continuation**, and the **Grobner bases**[4].

The analytical solution of the forward kinematics is tedious and in the end using the Dialytic Elimination method it presents 16 feasible solutions,[1] in the following I will present a general iterative algorithm to solve numerically the direct kinematics as presented in [5].

Fast Numerical Methods for Direct Kinematics

The unknown current pose will be close to the previous one found with direct kinematics, this idea allows to parse the multiple solutions and get the right one in a computationally light way.

A classic way for solving a NL system of equations is the Newton iterative method, let's assume that the coordinates $\mathbf{X}$ of the moving platform are related to the joint variables vector $\mathbf{q}$:

$$\mathbf{q} = f(\mathbf{X}) \tag{15}$$

considering $\mathbf{X_0}$ is an estimate of the solution, the Newton method at iteration k is defined as:

$$\mathbf{X_{k+1}} = \mathbf{X_k} + \mathbf{\Lambda}\left[\mathbf{q} - f(\mathbf{X})\right] \tag{16}$$

the most common stop techniques are the number of iterations (reiterate while $k <$*threshold*) or the error threshold (stop iterations when $|\mathbf{q} - f(\mathbf{X})| < \epsilon$).

There exist different choices of the map $\mathbf{\Lambda}$, for example:

- Newton-Raphson scheme: $\mathbf{\Lambda} = \mathbf{J} = \frac{\partial f^{-1}}{\partial \mathbf{q}} \cdot (\mathbf{X_k})$, this algorithm is first in the class of **Householder's methods**, succeeded by **Halley's method**.
- Damped Newton scheme: $\mathbf{\Lambda} = \alpha^k \cdot \mathbf{J} = \alpha^k \cdot \frac{\partial f^{-1}}{\partial \mathbf{q}} \cdot (\mathbf{X_k})$

Let us assume that for a given method there are two performance parameters c,r such that:

$$e_{k+1} \leq c \cdot e_k^r \tag{17}$$

r is called the **rate of convergence** and for the Newton-Raphson scheme $r = 2$. The Jacobian matrix can be found from the inverse kinematics extracting the function $f : \mathbf{X} \mapsto \mathbf{q}$ and taking its derivative wrt the joint variables. Basically we go on iterating until the joint variables found using the inverse kinematics ($f(\mathbf{X_k})$) on the estimate position of the moving

platform get close enough to the real ones ($\mathbf{q}$). It is possible to express the position of the moving platform with plücker coordinates:

$$\mathbf{X} = [x, y, z, \delta_x, \delta_y, \delta_z] \tag{18}$$

with $\delta_i = \alpha v_i \quad (i = x, y, z)$, δ is the **rotation vector** in which $\mathbf{v}$ is the rotation axis and α is the rotation angle. The **instantaneous rotation vector** is given by the Rodrigues' rotation formula:

$$\Omega = \dot{\alpha}\mathbf{v} + \sin\alpha\dot{\mathbf{v}} + (1 - \cos\alpha)\mathbf{v} \times \dot{\mathbf{v}} \tag{19}$$

and for small oscillations ($\alpha \approx 0$) the values of the trigonometric functions can be simplified:

$$\Omega \approx \dot{\alpha}\mathbf{v} + \alpha\dot{\mathbf{v}} \tag{20}$$

Easy to notice that the instantaneous rotation vector (Ω) is approximately equal to the time derivative of the rotation vector (δ), the time derivative of $\mathbf{X}$ is thus the twist of the moving platform. If we go back to the definition of inverse kinematics $\mathbf{q} = f(\mathbf{X})$ and take its time derivative, we will find the relation that defines the Jacobian for parallel manipulators:

$$\dot{\mathbf{q}} = J(\dot{\mathbf{x}}) \tag{21}$$

that is the connection between the moving platform twist with joint velocities.

Robot's Jacobian

As already introduced in the previous section, the Jacobian matrix is a map from the velocities of the moving platform in Cartesian coordinates to the velocities of the actuated joints $J : \dot{\mathbf{x}} \mapsto \dot{\mathbf{q}}$, where $\dot{\mathbf{q}}$ is an m-dimensional vector that stores joint velocities, $\dot{\mathbf{x}}$ is a n-dimensional vector that stores the moving platform velocoties, and J is the $n \times m$ Jacobian matrix. In our case the Jacobian is (fortunately) good looking and it is neither fat nor

skinny, as a matter of fact it maps 3 end effector degrees of freedom to 3 motors, it is a 3×3 matrix.

Gosselin and Angeles in [6] presented a Jacobian split in two different parts, they started defining a function of joint values and moving platform positions: $F(\mathbf{q}, \mathbf{x}) = 0$, taking its derivative gives back:

$$\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\dot{\mathbf{x}} = \frac{\partial \mathbf{F}}{\partial \mathbf{q}}\dot{\mathbf{q}} \; ; \; (J_F)_{3\times 3} \; \dot{\mathbf{x}} = (J_I)_{3\times 3} \; \dot{\mathbf{q}} \; ; \; J_F \begin{bmatrix} \dot{\theta}_{1,1} \\ \dot{\theta}_{1,2} \\ \dot{\theta}_{1,3} \end{bmatrix} = J_I \begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{p}_z \end{bmatrix} \tag{22}$$

the relations just presented are intuitively connected to the Jacobian:

$$J = [J_I^{-1} \cdot J_F]_{3\times 3} \tag{23}$$

In our case we can find a closed loop equation $\overrightarrow{OP} + \overrightarrow{PC_i} = \overrightarrow{OA_i} + \overrightarrow{A_iB_i} + \overrightarrow{B_iC_i}$ as presented in Fig. 4 and Fig. 3 [7].

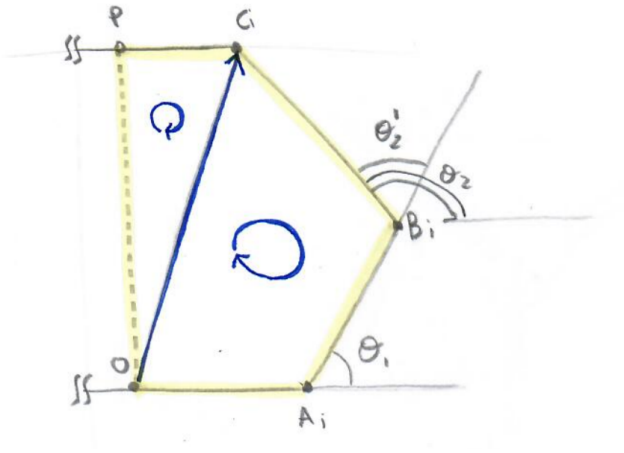

Figure 4: Side view with naming convention.

In matrix form this equation can be rewritten as:

$$\begin{bmatrix} p_x\cos\phi_i - p_y\sin\phi_i \\ p_x\sin\phi_i + p_y\cos\phi_i \\ p_z \end{bmatrix} = \begin{bmatrix} r \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} c \\ 0 \\ 0 \end{bmatrix} + a\begin{bmatrix} \cos\theta_{1i} \\ 0 \\ \sin\theta_{1i} \end{bmatrix} + \begin{bmatrix} (d+e+b\sin\theta_{3i})\cos\theta_{2i} \\ b\cos\theta_{3i} \\ (d+e+b\sin\theta_{3i})\sin\theta_{2i} \end{bmatrix} \tag{24}$$

That can be presented in the reduced form: $(\vec{\mathbf{p}}+\vec{\mathbf{c}}) = \vec{\mathbf{r}}+\vec{\mathbf{a}_i}+\vec{\mathbf{b}_i}$. Taking the time derivative leads to $\dot{\vec{\mathbf{p}}} = \dot{\vec{\mathbf{a}}}_i + \dot{\vec{\mathbf{b}}}_i$ (the constant terms drop out), every point of the moving platform has the same velocity, thus we can write $\dot{\vec{\mathbf{p}}} = \vec{\mathbf{v}}$.

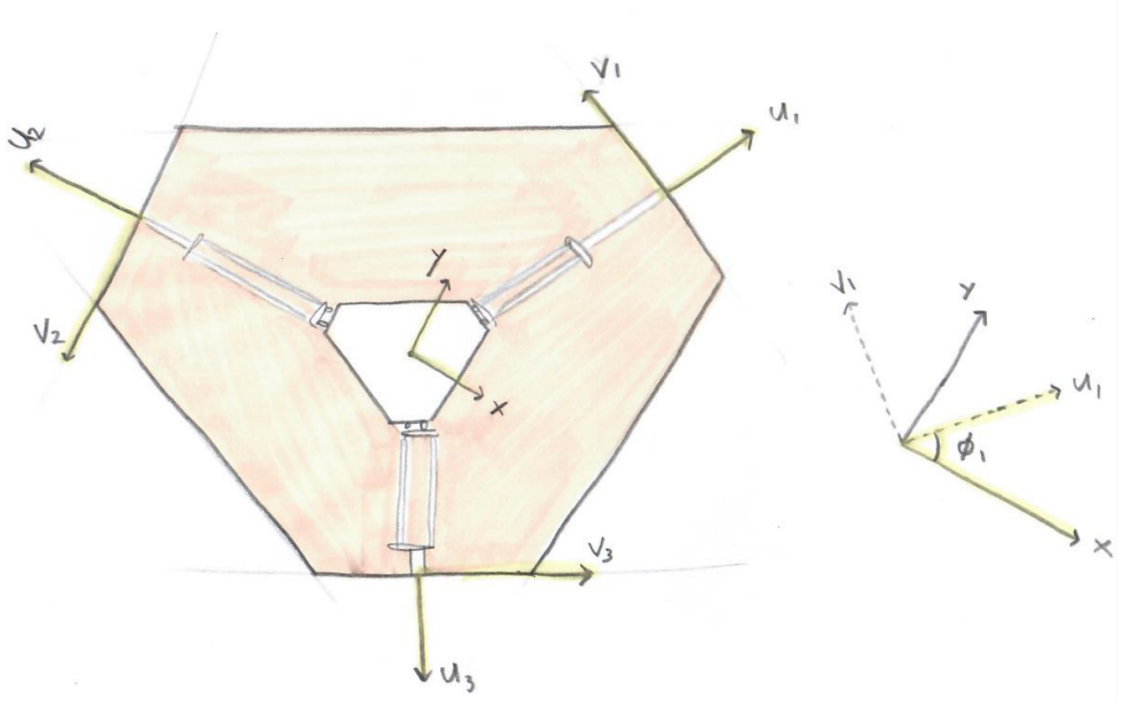

Figure 5: Upper view of the robot with axis convention.

Using the relations between linear and angular velocities we can write the latter equation as:

$$\vec{\mathbf{v}} = \vec{\omega}_{a,i} \times \vec{\mathbf{a}}_i + \vec{\omega}_{b,i} \times \vec{\mathbf{b}}_i \tag{25}$$

However the angular velocity of b_i is tough to get since it depends on $\dot{\theta}_{2i}$ and $\dot{\theta}_{3i}$, the

shortcut to get rid of that term is to scalar multiply everything by $\vec{\mathbf{b}}_i$:

$$\vec{\mathbf{b}}_i \cdot \vec{\mathbf{v}} = \vec{\mathbf{b}}_i \cdot (\vec{\omega}_{a,i} \times \vec{\mathbf{a}}_i) + \vec{\mathbf{b}}_i \cdot (\vec{\omega}_{b,i} \times \vec{\mathbf{b}}_i) \tag{26}$$

We can now expand the left-hand side of the equation with:

$$\vec{\mathbf{b}}_i \cdot \vec{\mathbf{v}} = (K(\theta_{3i}) \cos\theta_{2i})(v_x \cos\phi_i - v_y \sin\phi_i) + b \cos\theta_{3i}(v_x \sin\phi_i + v_y \cos\phi_i) + (K(\theta_{3i}) \sin\theta_{2i})v_z \tag{27}$$

with $K(\theta_{3i}) = d + e + b \sin\theta_{3i}$, reordering the coefficients:

$$\vec{\mathbf{b}}_i \cdot \vec{\mathbf{v}} = J_{ix}v_x + J_{iy}v_y + J_{iz}v_z \tag{28}$$

these are the three parameters that have entries:

$$J_{ix} = K(\theta_{3i}) \cos\theta_{2i} \cos\phi_i + b \cos\theta_{3i} \sin\phi_i \tag{29}$$

$$J_{iy} = -K(\theta_{3i}) \cos\theta_{2i} \sin\phi_i + b \cos\theta_{3i} \cos\phi_i \tag{30}$$

$$J_{iz} = K(\theta_{3i}) \sin\theta_{2i} \tag{31}$$

Going back to the right-hand side of Eq. 26 we can reorganize the angular part:

$$\vec{\mathbf{b}}_i \cdot (\vec{\omega}_{a,i} \times \vec{\mathbf{a}}_i) + \vec{\mathbf{b}}_i \cdot (\vec{\omega}_{b,i} \times \vec{\mathbf{b}}_i) = \vec{\mathbf{b}}_i \cdot (\vec{\omega}_{a,i} \times \vec{\mathbf{a}}_i) \tag{32}$$

in particular the link a_i moves in the $u_i w_i$ plane, thus $\omega_{a,i} = [0, -\dot{\theta}_{1i}, 0]^T$ and consequently:

$$\vec{\omega}_{a,i} \times \vec{\mathbf{a}}_i = det \begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ 0 & -\dot{\theta}_{1i} & 0 \\ a_{1i} & a_{2i} & a_{3i} \end{bmatrix} = -a_{3i}\dot{\theta}_{1i}\mathbf{i} + a_{1i}\dot{\theta}_{1i}\mathbf{k} \tag{33}$$

Finally the expression of the right-hand side of Eq. 26 can be reduced to:

$$\vec{\mathbf{b}}_i \cdot (\vec{\omega}_{a,i} \times \vec{\mathbf{a}}_i) = aK(\theta_{3i})\dot{\theta}_{1i}[\sin\theta_{2i}\cos\theta_{1i} - \cos\theta_{2i}\sin\theta_{1i}] = \tag{34}$$

$$= aK(\theta_{3i})\dot{\theta}_{1i}[\sin(\theta_{2i} - \theta_{1i})] \tag{35}$$

the reader can notice graphically that $\theta_{2i} = \theta_{1i} + \theta_{2'i}$ as presented in Fig. 4, hence:

$$\vec{\mathbf{b}}_i \cdot (\vec{\omega}_{a,i} \times \vec{\mathbf{a}}_i) = \dot{\theta}_{1i}K(\theta_{3i})a\sin(\theta_{2'i}) \tag{36}$$

Adding up all these pieces to Eq. 26:

$$J_{1x}v_x + J_{1y}v_y + J_{1z}v_z = \dot{\theta}_{11}K(\theta_{31})a\sin(\theta_{2'1}) \tag{37}$$

$$J_{2x}v_x + J_{2y}v_y + J_{2z}v_z = \dot{\theta}_{12}K(\theta_{32})a\sin(\theta_{2'2}) \tag{38}$$

$$J_{3x}v_x + J_{3y}v_y + J_{3z}v_z = \dot{\theta}_{13}K(\theta_{33})a\sin(\theta_{2'3}) \tag{39}$$

That allows us to finally get the two part Jacobian:

$$J_I\vec{\mathbf{v}} = J_F\dot{\vec{\theta}} \tag{40}$$

with:

$$J_I = \begin{bmatrix} K(\theta_{31})c\theta_{21}c\phi_1 + bc\theta_{31}s\phi_1 & -K(\theta_{31})c\theta_{21}s\phi_1 + bc\theta_{31}c\phi_1 & K(\theta_{31})s\theta_{21} \\ K(\theta_{32})c\theta_{22}c\phi_2 + bc\theta_{32}s\phi_2 & -K(\theta_{32})c\theta_{22}s\phi_2 + bc\theta_{32}c\phi_2 & K(\theta_{32})s\theta_{22} \\ K(\theta_{33})c\theta_{23}c\phi_3 + bc\theta_{33}s\phi_3 & -K(\theta_{33})c\theta_{23}s\phi_3 + bc\theta_{33}c\phi_3 & K(\theta_{33})s\theta_{23} \end{bmatrix} \tag{41}$$

$$J_F = \begin{bmatrix} aK(\theta_{31})as(\theta_{2'1}) & 0 & 0 \\ 0 & aK(\theta_{32})as(\theta_{2'2}) & 0 \\ 0 & 0 & aK(\theta_{33})as(\theta_{2'3}) \end{bmatrix} \tag{42}$$

Characterization

This section presents the results of work conducted by the Autonomous Systems Lab at CSIRO in characterising the Falcons geometric, inertial and actuation properties for application to model-based robotic algorithm development, the aim is to present the key kinematic and dynamic parameters of the device.

Interface

The Falcon uses an USB interface and the datagrams are read and sent by the firmware of the onboard μcontroller. Novint has released a Software Development Kit (SDK) with their Application Programming Interface (API) in order to allow ad hoc programming of the device. The interface uses a 1kHz update rate, the authors of [2] found that a 1kHz update rate was unable to be sustained over the USB interface, and typically missed commands or reads resulted in a real-world communication rate between 800Hz and 1kHz, depending on the controlling computers payload. This interface also resulted in a noticeable (2-5 samples) delay between force commands being issued and changes in the encoder measurements being received by the controlling computer. The exact cause of this delay is unknown. It is possible that some of these delays may be eliminated with custom firmware on the Falcons internal controller chip, but this has not been investigated by the authors.

Geometric and inertial Properties

Still in[2] a Falcon robot was disassembled to allow inertial and geometric characterisation. The link masses were measured with a set of scales. Link centre-of-mass was able to be measured by freely hanging the component from a pivot and comparing the pose of the link with that of a vertical plumb-line. Repetition with multiple pivot points on the component allow the centre-of-mass to be triangulated. Similarly, the mass-moment of inertia of a component can be determined by triaxial pendulum analysis, where the period of oscillation

in orthogonal planes gives rise to the inertia tensor. For most links of the Falcon, full triaxial analysis is not required due to the planar motion of the components, for example the main leg links that connect to the base can only rotate around a single pivot (and the second mass moment of inertia is suf- ficiently characterised by a single term in one plane), and the end effector is constrained to translational motions only. A sanity check was made by comparison of the experimentally derived values to those provided by a solid modelling package. A summary of the results of this characterisation are presented in Tables 1 and 2.

dimension	value (m $\times 10^{-3}$)
a	60.0
b	102.5
c	15.7
d	11.5
e	11.5
f	26.2
g	27.9
r	36.6
s	27.2

Table 1: Key geometric properties for the links of the Novint Falcon. Labelling corresponds to that shown in Figure 3.

	leg	shin bar	shin joint	end effector
mass $\times 10^{-3}$	89.59	8.37	10.42	82.73
C.M.†				
$x \times 10^{-3}$	-3.19	56.35	15.61	15.7
$y \times 10^{-3}$	23.34	0	0	26.2
$z \times 10^{-3}$	0	0	0	48.78
I_{xx}	-	-	-	
I_{yy} $\times 10^{-6}$	-	8.5	-	-
I_{zz} $\times 10^{-6}$	140	8.5	0.7	-

Table 2: Inertial properties for the links of the Novint Falcon. Units are S.I. † Centre-of-mass, defined as a triple (x,y,z) with the x-direction along the line from parent to child joints, working outwards from the base.

Motors

The Novint Falcon is actuated by three Mabuchi RS- 555PH-15280 motors whose motion can be monitored by a coaxial 4-state encoder with 320 lines per revolution. These motors are directly coupled to a 14.25 mm diameter drum, upon which a 0.5 mm diameter cable is wrapped. The cable runs around the outer edge of the main leg links at a radius of 56.0mm, and is tensioned by a spring at the inner end of the leg. This arrangement allows for a very low friction and robust actuation mechanism effectively providing a 7.62:1 (outer leg radius to drum radius, plus correction for cable width) gain to the motor output about the main legs pivot without resorting to the use of gears.

Workspace

The most common criticism of the delta-robot form is the limited workspace. The three limbs of the Falcon work in kinematic (and with an appropriate control methodology, dynamic) concert to actuate the end-effector, but each leg is limited by the reach of the connected linkages. This results in a workspace bounded by warped tri-hemispherical regions overlapping along the common longitudinal (z) axis. To quantify this volume, a number of random points were generated in Cartesian space and tested to see if they were kinematically feasible. As the number of random points were increased, so a better estimate of the workspace volume was formed. Similarly, by simple application of limit theory, we can also estimate the enclosing volume to be approximately $7.90 \times 10^{-5} m^3$. The projection of the workspace is presented in Fig. 6.

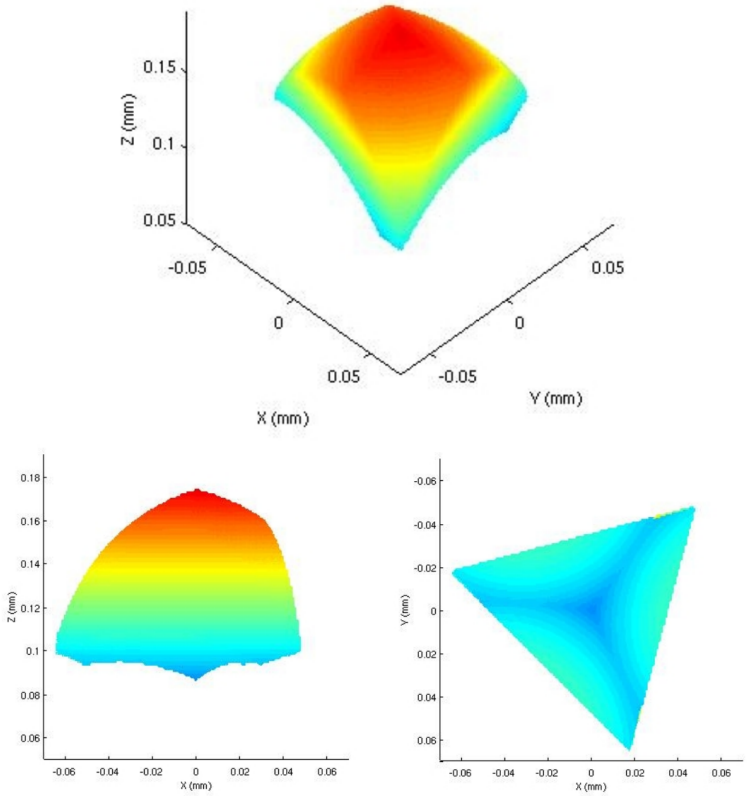

Figure 6: Workspace of the Falcon with projections on XY and YZ.

Installation

In order to interface the Falcon with your computer you need the right drivers that are available here:

- Drivers for communication

Then you will need the setup app with the tutorials:

- Setup & tutorials

And finally if you want to write your own software you would want the SDK:

- SDK

After having downloaded these files you can run them with administrator privileges.

Restart your PC to have all the Paths set up properly, connect the supply to the Falcon and connect it to your PC (to a USB 2.0 port!!).

Go to (the path automatically chosen by the installer should be the same, the only difference may be the absence of (x86) if you are working on a 32 bit computer):

- `C:\Program Files(x86)\Novint\Falcon\TestUtilities\FalconTest`

and run "FalconTest", it will open a simple GUI where you will be able to test the main features of the Falcon robot.

API setup

The interface with the μcontroller inside the Falcon and thus with the robot is made through the API provided by Novint, the latter allows a nice control of the Falcon starting a thread for the motor control and the position of the end effector. The manual for the API are available online, in particular the most important are [8] [9] . In this section it will be described a simple way to setup the libraries and dependencies in a Visual Studio 2012 new project:

1. Open VS 2012, create a New Project, select Visual C++ language and click on General, give the project a name (for example **FalconTest**) and confirm.

2. Right click on Source Files → New Item → Select C++ file and call it Main.cpp

3. Right click on Source Files → New Item → Select Header file and call it Define.h, inside define win32 writing `#define WIN32`

4. Right click on FalconTest on the left panel → Open folder in file explorer → and copy here the *include* and *lib* folders available here:

   - `C:\Program Files (x86)\Novint\HDAL_SDK_2.1.3\`

5. Right click on FalconTest on the left panel → Properties:

   - C/C++ → General → Additional Include Directories:
     - `$(ProjectDir)\Include;`
   - Linker → General → Additional Library Directories:
     - `$(ProjectDir)\Lib;`
   - Linker → Input → Additional Dependencies → write `hdl.lib:`

6. copy the .dll files available here:

   - `C:\Program Files (x86)\Novint\HDAL_SDK_2.1.3\bin`

   into the folder where the .sln is and into the Debug/Release directory.

Now in the main file try to write a simple test code to check if the libraries are correctly included:

```
#include"Define.h"
#include<stdio.h>
#include"hdl\hdl.h"
```

```
#include"hdlu\hdlu.h"
int main(){
   printf("Falcon dependencies ok");
   getchar();
   return 0;}
```

Press F5 and check if the code is compiled without the return of errors.

Haptik Library setup

There are other APIs available online that allow to build an interface with the Falcon, the most important ones are:

- Haptik Library by SSSA, Pisa
- Libnifalcon

For a first introduction to the robot I suggest to stick to the API provided by Novint since they are simple and definitely enough to build interesting stuff.

Code Example

In the following is attached a basic example that prints the position of the moving platform and compensate the effect of gravity, it is divided with the classic structure of a C++ code:

- main.cpp
- className.cpp
- className.h

```
// Gregorio Aiello, BIROMED Lab, University of Basel
// November 2016
// main.cpp
#include<stdio.h>
#include <windows.h>
#include"falcon.h"
int main()
{
double cursorPos[3];
Falcon run;
while(1)
{
run.getPosition(cursorPos);
printf("X: %f, Y: %f, Z: %f \n", 100*cursorPos[0], 100*cursorPos[1], 100*cursorPos[2]);
Sleep(100);
}
return 0;
}
```

```
// Gregorio Aiello, BIROMED Lab, University of Basel
// November 2016
// Falcon.h
#pragma once
#include<stdio.h>
#include <windows.h>
#include "hdl/hdl.h"
#include "hdlu/hdlu.h"
const bool bNonBlocking = false;
const bool bBlocking = true;
class Falcon
{
friend HDLServoOpExitCode ContactCB(void *data);
public:
Falcon(void);
~Falcon(void);
// Get position
void getPosition(double pos[3]);
private:
void init();
// Clean up
    void uninit();
// Nothing happens until initialization is done
    bool m_inited;
// Variables used only by servo thread
    double m_positionServo[3];
    bool   m_buttonServo;
```

```
    double m_forceServo[3];
    bool   m_buttonApp;
// Handle to device
    HDLDeviceHandle m_deviceHandle;
    // Handle to Contact Callback
    HDLServoOpExitCode m_servoOp;
};
```

```
// Gregorio Aiello, BIROMED Lab, University of Basel
// November 2016
// Falcon.cpp
#include "Falcon.h"
Falcon::Falcon(void)
{
init();
}
Falcon::~Falcon(void)
{
uninit();
}
void Falcon::init()
{
m_forceServo[0] = 0;
    m_forceServo[1] = 0;
    m_forceServo[2] = 0;
HDLError err = HDL_NO_ERROR;
    // Passing "DEFAULT" or 0 initializes the default device based on the
    // [DEFAULT] section of HDAL.INI.   The names of other sections of HDAL.INI
    // could be passed instead, allowing run-time control of different devices
    // or the same device with different parameters.  See HDAL.INI for details.
    m_deviceHandle = hdlInitNamedDevice("DEFAULT");
    // Now that the device is fully initialized, start the servo thread.
    // Failing to do this will result in a non-funtional haptics application.
    hdlStart();
m_servoOp = hdlCreateServoOp(ContactCB, this, bNonBlocking);
```

```
}
void Falcon::getPosition(double pos[3])
{
    pos[0] = m_positionServo[0];
    pos[1] = m_positionServo[1];
    pos[2] = m_positionServo[2];
}
HDLServoOpExitCode ContactCB(void* pUserData)
{
    // Get pointer to haptics object
    Falcon* haptics = static_cast< Falcon* >( pUserData );
    // Get current state of haptic device
    hdlToolPosition(haptics->m_positionServo);
    hdlToolButton(&(haptics->m_buttonServo));
    // Send forces to device
    hdlSetToolForce(haptics->m_forceServo);
    // Make sure to continue processing
    return HDL_SERVOOP_CONTINUE;
}
void Falcon::uninit()
{
    if (m_servoOp != HDL_INVALID_HANDLE)
    {
        hdlDestroyServoOp(m_servoOp);
        m_servoOp = HDL_INVALID_HANDLE;
    }
    hdlStop();
```

```
    if (m_deviceHandle != HDL_INVALID_HANDLE)
    {
        hdlUninitDevice(m_deviceHandle);
        m_deviceHandle = HDL_INVALID_HANDLE;
    }
    m_inited = false;
}
```

References

(1) Stamper, R. E.; w. Tsai, A. L. A Three Degree Of Freedom Parallel Manipulator With Only Translational Degrees Of Freedom. 1997.

(2) Martin, S.; Hillier, N. Characterisation of the Novint Falcon Haptic Device for Application as a Robot Manipulator. Australasian Conference on Robotics and Automation (ACRA). 2009.

(3) Simaan, N. Analysis and Synthesis of Parallel Robots for Medical Applications. 1999.

(4) Ponleitner, B. Fundamental Mathematical Concepts for Problems Arising in Robotics. 2013.

(5) Merlet, J. Parallel Robots, Second Edition, Springer. 2006.

(6) Gosselin, A. Singularity Analysis of Closed-Loop Kinematic Chains. 1990.

(7) Lopez., M. Delta robot: inverse, direct, and intermediate Jacobians. 2005.

(8) Novint Technologies incorporated Albuquerque, N. U. Haptic Device Abstraction Layer (HDAL): Programmer's Guide Version 2.1.3. 2008.

(9) Novint Technologies incorporated Albuquerque, N. U. Haptic Device Abstraction Layer (HDAL): API Reference Version 2.1.3. 2008.