Calculating the Pose of a 3-DOF Parallel Manipulator: Forward and Inverse Kinematics using Artificial Neural Networks

W. Matt Peterson
Montana State University, Mechanical and Industrial Engineering Dept.
Bozeman, MT 59718
May 1, 2014

In this report we describe the construction and training of Artificial Neural Networks (ANNs) capable of rapidly solving the forward and inverse kinematic problem for the pose of a planar 3-degree of freedom parallel manipulator. Firstly, the analytical expressions for the inverse kinematic problem are developed. Secondly, these expressions are implemented in Matlab in order to obtain exact solutions for a number of discrete positions in the reachable workspace for a specified parallel manipulator. Thirdly, an ANN is constructed and trained using a subset of the exact solutions and is shown to be capable of finding highly accurate solutions to the inverse kinematic problem over the entire reachable workspace. Finally, and most importantly, an ANN is then constructed and trained to rapidly solve the forward kinematic problem with high accuracy, while entirely avoiding the costly iterative numerical solution that is normally required.

I. Introduction

Parallel manipulators are multi-degree of freedom robots with multiple articulating limbs connected to a single platform, or end-effector, thus forming multiple closed-loop kinematic chains between the limbs, platform, and base. Parallel manipulators are often used for "pick-and-place" material handling tasks (for example, the delta robot in Fig. 1a) or as flight/automobile simulator platforms (for example, the Gough-Stewart platform in Fig. 1b).

(a) (b)

Figure 1: A delta robot (a); a Gough-Stewart platform (b).

For this report we take as inspiration the Gough-Stewart (GS) platform, also known as a hexapod or six-axis platform. The GS platform normally consists of 3 pairs of prismatic actuators connecting the base to the end-effector, all of which act in combination to produce 6 degrees of freedom at the working platform: 3 translational DOF in the Cartesian x, y, and z directions, and 3 rotational DOF about each of the Cartesian axes (also known as roll, pitch, and yaw).

A simpler variation of the standard 6-DOF GS platform is the 2-legged planar parallel manipulator illustrated in Fig. 2. Each prismatic joint is connected to a fixed base and the moving platform by passive revolute joints. For planar motion, the number of DOF (i.e., the mobility) of the mechanism can be verified using the equation:

$$\begin{aligned} M &= 3(n-g) + \sum_{i=1}^{g} f_i \\ &= 3(5-6) + 6 \\ &= 3 \end{aligned} \qquad \text{Eq. (1)}$$

where n is the number of moving links, g is the number of joints, and f_i is the number of unconstrained DOF of the i^{th} joint. Because this configuration has 3 DOF (translation in the x- and y-directions, and rotation about the z-axis) and each leg consists of two revolute joints and a prismatic actuator, we will denote this configuration as a planar 3-RPR parallel manipulator, or simply as a 3-RPR. Notice that in this configuration the dual prismatic joints must work synergistically to produce translation in the x and y directions, as well as a rotation about the z axis.

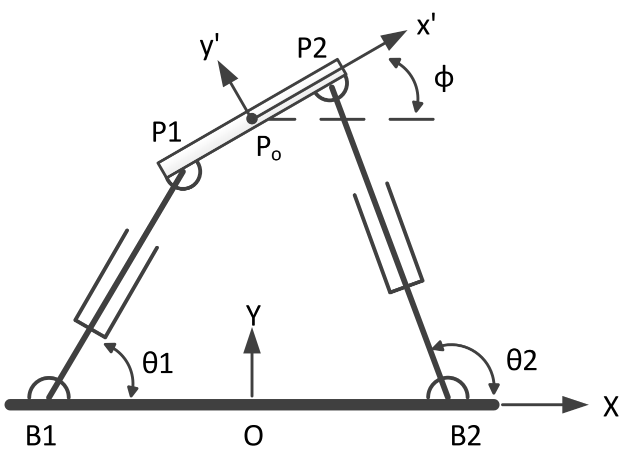


Figure 2: A Planar 3-RPR Parallel Manipulator

One might note a similarity between a 4-bar linkage and the 3-RPR. The main difference is that in the 3-RPR the length of each leg is permitted to change, and the kinematic problem becomes a function of the prismatic actuator length.

II. Mechanical Configuration, Posture, and Platform Pose

Broadly speaking, kinematics refer to the study of motion – this includes position, velocity, acceleration, as well as trajectory. Each of these topics are important in the design of a parallel robot. In this report, we are interested only in the fundamental task of relating joint parameters to the position and orientation of the working platform (also known as the "pose" of the platform) – in other words, we will determine the "posture" of the mechanical system for any desired platform pose.

We must define some useful geometric parameters. First, let the Cartesian pose of the platform be denoted by the set:

$$\chi = (T_x\ , T_y, \varphi) \qquad \text{Eq. (2)}$$

where (T_x, Ty) are the components of a 2D translation vector, $\boldsymbol{T}$, from the origin of the base reference frame, O, to the platform coordinate system, P_o, and φ is the platform rotation measured from the positive base frame X-axis. The active prismatic joint variables are the leg lengths:

$$\begin{aligned} L &= (l_1, l_2) \\ &= (a_1 + d_1, a_2 + d_2) \end{aligned} \qquad \text{Eq. (3)}$$

where a_i is the minimum length of the i^{th} leg. Note that the prismatic joint extension of the i^{th} leg is actively controlled within the range $0 \le d_i \le d_{max}$. Therefore, the total leg lengths may vary between $a_i \le L_i \le a_i + d_{max}$ The passive revolute joint angles, measured CCW from the base X-axis, are:

$$\theta = (\theta_1, \theta_2) \qquad \text{Eq. (4)}$$

As shown in Fig. 3a, each link in the 3-RPR can be represented by the vectors:

Vector	Reference Frame	Description
$\boldsymbol{T}$	Base, O	From $O \rightarrow P_o$
$\boldsymbol{b_1}$	Base, O	From $O \rightarrow B1$
$\boldsymbol{b_2}$	Base, O	From $O \rightarrow B2$
$\boldsymbol{l_1}$	Base, O	From $B1 \rightarrow P2$
$\boldsymbol{l_2}$	Base, O	From $B2 \rightarrow P2$
$\boldsymbol{p_1}$	Platform, P_o	From $P_o \rightarrow P1$
$\boldsymbol{p_2}$	Platform, P_o	From $P_o \rightarrow P2$

Notice that vectors $\boldsymbol{p_1}$ and $\boldsymbol{p_2}$ are described in the platform reference frame (x', y').

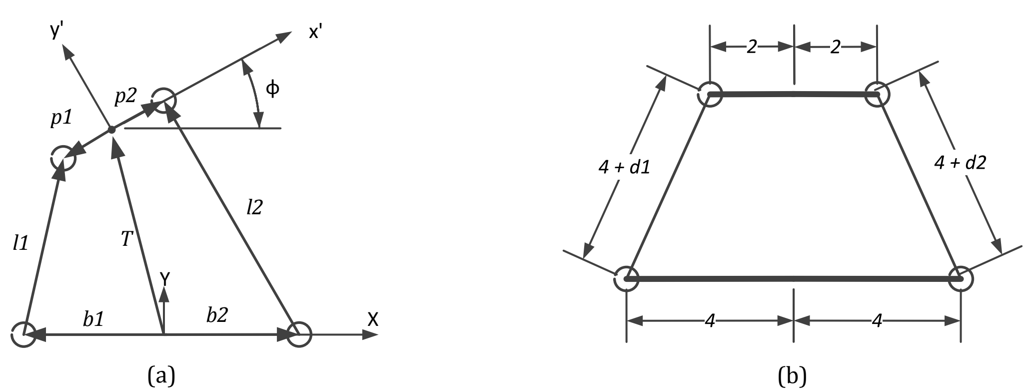


Figure 3: Directed line segment description (a), Specific geometric values (b).

We will analyze a specific 3-RPR configuration by assuming the static geometric values (see Fig 3b):

$\boldsymbol{b_1}$ = (-4, 0) $\qquad$ $\boldsymbol{p_1}$ = (-2, 0) $\qquad$ a_1 = 4

$$\boldsymbol{b_2} = (4, 0) \qquad \boldsymbol{p_2} = (2, 0) \qquad a_2 = 4$$

Finally, we also specify a default "home" position for the working platform, chosen here as:

$$\chi = (0, 6, 0°)$$

III. Kinematic Equations

The kinematic equations relating the platform pose to the leg and joint configuration (i.e., the posture of the system) can be approached in two complimentary ways:

I. Forward Kinematics – The forward kinematics problem for the 3-RPR is stated as: "Given the current length and angle of each leg, calculate the platform pose."

II. Inverse Kinematics – The inverse kinematic problem for the 3-RPR is stated as: "Given the desired platform pose, calculate the required leg lengths."

The robotics literature appears to favor rather tedious compositions of transformation matrices in order to calculate the IKP. While this is understandably needed for serial manipulators, we have found this to be unnecessary for a parallel manipulator and will take a more direct approach, as follows. For the inverse kinematic problem (IKP), we may specify the desired platform pose and calculate the leg vectors using the equation:

$$\boldsymbol{l}_i = \boldsymbol{T} + \boldsymbol{R} \cdot \boldsymbol{p}_i - \boldsymbol{b}_i \qquad for\ i = 1,2 \qquad \text{Eq. (5)}$$

where $\boldsymbol{R}$ is the rotation (transformation) matrix for the desired platform angle. Notice that the IKP of Eq. (5) can be calculated independently for each leg for any given pose, χ, with the same result. (However, we will later show that the equations to solve the FKP cannot be calculated independently, which complicates matters somewhat). For example, the IKP using leg 1 we obtain:

$$\boldsymbol{l}_1 = \begin{Bmatrix} l_{1x} \\ l_{1y} \end{Bmatrix} = \begin{Bmatrix} T_x \\ T_y \end{Bmatrix} + \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} \cdot \begin{Bmatrix} p_{1x} \\ p_{1y} \end{Bmatrix} - \begin{Bmatrix} b_{1x} \\ b_{1y} \end{Bmatrix} \qquad \text{Eq. (6)}$$

where the components $l_{1x}\ and\ l_{1y}$ are the only unknowns. We may then solve for the prismatic joint extension:

$$d_i = |l_i| - a_i \qquad \text{Eq. (7)}$$

We can also find the leg angles $\theta_1$and θ_2 (shown in Fig. 2) using the vector dot product:

$$\theta_i = \frac{\operatorname{acos}(\boldsymbol{b}_i \cdot \boldsymbol{l}_i)}{|\boldsymbol{b}_i||\boldsymbol{l}_i|} \qquad \text{Eq. (8)}$$

Notice that for θ_1 the negative of $\boldsymbol{b}_i$ should be used, which gets us the angle we are interested in. Thus, the IKP is straight-forward for the 3-RPR and most other parallel manipulators, and is relatively computationally inexpensive.

For the forward kinematic problem (FKP), we specify values for the vectors describing the legs connection points on the base and working platform ($\boldsymbol{b}_i$ and $\boldsymbol{p}_i$, respectively), as well as the leg vectors ($\boldsymbol{l}_i$), and solve for the platform pose, $\chi = (T_x, T_y, \varphi)$.

Rearranging the vector loop equations of Eq. (2), we obtain a system of 4 simultaneous nonlinear trigonometric equations (2 for each leg):

$$\begin{Bmatrix} T_x \\ T_y \end{Bmatrix} + \begin{bmatrix} cos\,\phi & -sin\,\phi \\ sin\,\phi & cos\,\phi \end{bmatrix} \cdot \begin{Bmatrix} p_{1x} \\ p_{1y} \end{Bmatrix} = \begin{Bmatrix} l_{1x} \cos\theta_1 \\ l_{1y} \sin\theta_1 \end{Bmatrix} + \begin{Bmatrix} b_{1x} \\ b_{1y} \end{Bmatrix} \qquad \text{Eq. (9)}$$

$$\begin{Bmatrix} T_x \\ T_y \end{Bmatrix} + \begin{bmatrix} cos\,\phi & -sin\,\phi \\ sin\,\phi & cos\,\phi \end{bmatrix} \cdot \begin{Bmatrix} p_{2x} \\ p_{2y} \end{Bmatrix} = \begin{Bmatrix} l_{2x} \cos\theta_2 \\ l_{2y} \sin\theta_2 \end{Bmatrix} + \begin{Bmatrix} b_{2x} \\ b_{2y} \end{Bmatrix}$$

The solution to these equations may be approximated using an iterative numerical approach, for example, with the Newton-Raphson method. This becomes a potential bottleneck for parallel manipulators with larger DOF, such as in the case of the standard 6-DOF Gough-Stewart platform (with 18 simultaneous nonlinear iterative equations), and in the case where rapid movement is required.

IV. The Neural Network Approach

Artificial Neural Networks (ANNs) are functional mapping systems that may be advantageous when modeling complex nonlinear systems, and in cases where closed-form solutions are unknown or difficult to obtain. Loosely based on biological principles observed in the nervous systems of animals, ANNs consist of a (sufficiently) large number of simple processors linked by weighted connections. Depending on the connections and network architecture, many complex functions can be approximated with arbitrary accuracy. Like the biological systems they are based upon, ANNs "learn" from example – using input patterns and desired outputs as training data, the connection weights between processing units (the "neurons") can be adjusted so that the network produces the correct output; i.e. the network is adaptive. This can be useful, for example, to account for nonlinear effects which might otherwise be ignored in simplified analytical models. In addition, multiple inputs and outputs may be processed simultaneously, resulting in very fast computation.

All of these properties indicate that an artificial neural network could be a suitable tool for both forward and inverse kinematic problems for a parallel manipulator. In the following section of this report we focus upon using the Matlab Neural Network Toolbox which provides many built-in functions and algorithms for creating and training ANNs, and analyzing the network output.

V. Analytical IKP Solution

The IKP of Eqs. (5-8) were solved, subject to the geometric values (a_i, b_i, p_i) given above. We have also constrained the mechanical system so that physically realistic "admissible" solutions fall within an acceptable range of values:

$$0 \le d_i \le 4$$
$$\theta_1 \le 90°$$
$$\theta_2 \ge 90°$$

Solutions outside of these accepted ranges are flagged as "inadmissible".

The complete analysis was completed using the process described by the flowchart in Fig. 4, which illustrates the relationship between different programs, inputs, and output results.

Note that we have provided an interactive solution process:

1. From the Matlab command line interface (CLI), call the script 'IKPP.m':

   ```
   >> IKPP()
   ```

   - Calling this script with no arguments calculates the inverse kinematic problem for a default "home" position, writes the solution to the command window, and plots the resulting posture (see Fig. 5a).
   - A warning is issued for "inadmissible solutions", for which the leg lengths or leg angles are out of the acceptable range of values.

2. You may call IKPP.m from the CLI with input arguments defining the desired platform pose. For example:

   ```
   >> IKPP(-1.8,6,12.5)
   ```

   - This will calculate the inverse kinematic problem for the platform pose input using the format of Eq. (2): $\chi = \left(T_x\ , T_y, \varphi\right) = (-1.8, 6, 12.5°)$

   - The solution (leg lengths and angles) will be written to the command window, and the plot will update (see Fig. 5b).

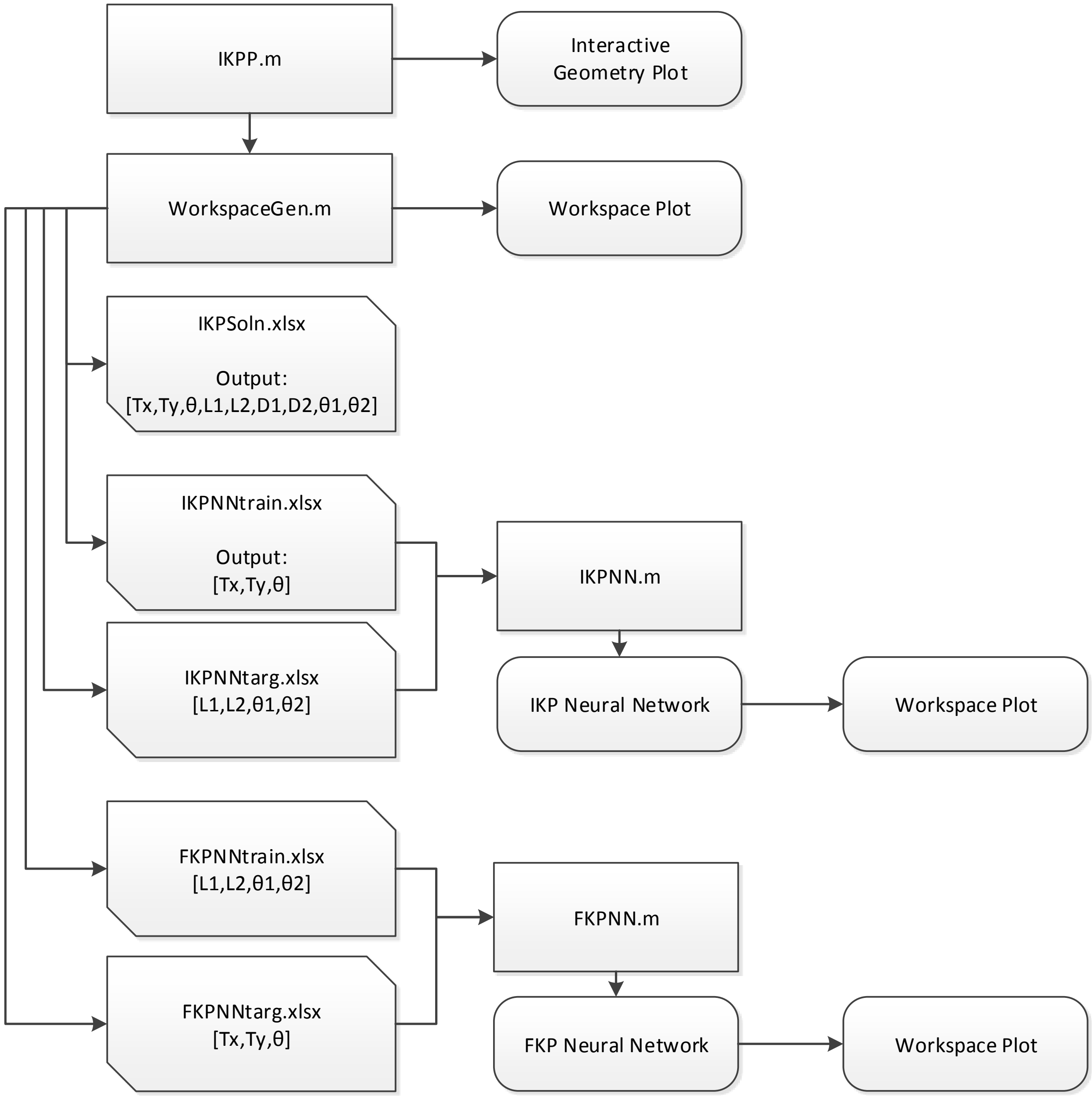
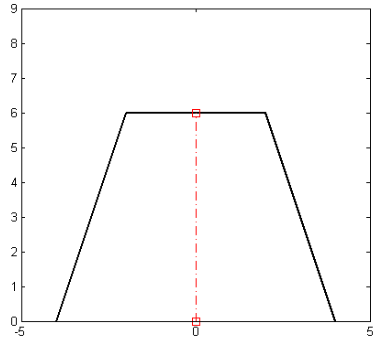
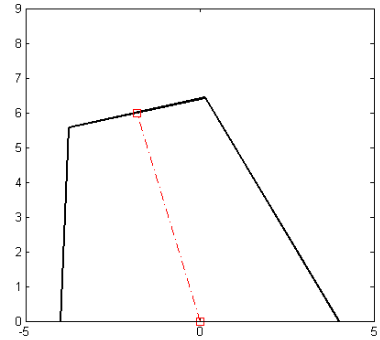


Figure 4: Analysis Flowchart



(a)



(b)

Figure 5: Interactive IKP Solutions. (Generated using 'IKPP.m')

Subject to the mechanical geometry and joint constraints given above, there exists a finite set of points that the origin of the manipulator platform, P_o, can reach. We call this region the robot "workspace". We can distinguish between the "reachable" workspace and the "dexterous" workspace, defined as:

- Reachable Workspace: the region that can be reached with *at least one* platform orientation.
- Dexterous Workspace: the region that can be reached with *any* platform orientation. Thus, the dexterous workspace is a subset of the reachable workspace.

To generate a set of admissible solutions, we provide the Matlab script 'WorkspaceGen.m'. As described in the flowchart of Fig. 4, 'WorkspaceGen.m' automatically generates several Microsoft Excel spreadsheets containing analytical IKP solutions and useful ANN training data. In addition, this script generates a plot illustrating the 3-RPR workspace for the specified mechanical geometry, using red squares to describe the reachable workspace and blue squares to describe the dexterous workspace. The script can be used to produce results for any desired number of discrete positions. For example, the plots in Fig. 6 show increasingly coarse discretizations of the workspace, with correspondingly smaller numbers of evaluation points. Of course, not all of the evaluation points will result in admissible solutions in either the reachable or dexterous workspaces. Inadmissible solutions are intuitively indicated by blank space.

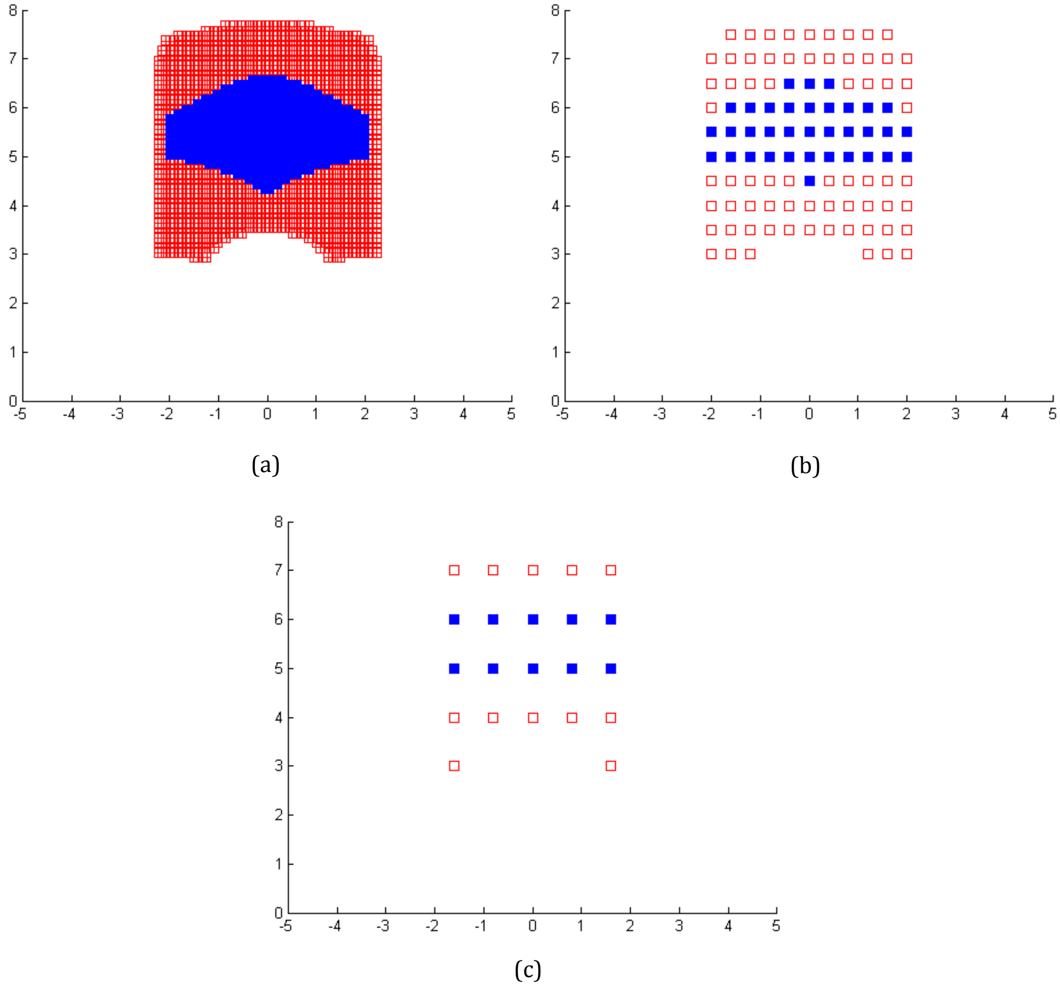


Figure 6: Workspace plots generated by evaluating 310,000 different poses (a), 9,261 different poses (b); and 1,331 different poses (c). (Plots generated using 'WorkspaceGen.m' provided with this report)

VI. Neural Network Solutions

To train and validate the ANNs, we use the solutions obtained from the workspace generation process described above. That is, we obtain the following data sets (see also the flowchart in Fig. 4):

Inverse Kinematic Problem Neural Network (IKPNN):

- Training Data: "IKPNNTrain.xlsx"
  - Consisting of [T_x, T_y, φ] values for each admissible IKP solution.
- Target Data: "IKPNNTarg.xlsx"
  - Consisting of [$L1$, $L2$, $\theta 1$, $\theta 2$] values for each admissible IKP solution.

The FKP Neural Network (FKPNN) uses the same data sets, but in the opposite capacity:

- Training Data: "IKPNNTarg.xlsx"

- Consisting of [$L1$, $L2$, $\theta 1$, $\theta 2$] values for each admissible IKP solution.

- Target Data: "IKPNNTrain.xlsx"
  - Consisting of [T_x, T_y, φ] values for each admissible IKP solution.

Using only the admissible solutions from the generation of the workspace, it was quite easy to find very good (or perfect) correlation between the ANN outputs and analytical solutions. For example, in Table 1, we list some network configurations that worked well. Notice that the single-layer net performed admirably, even for only 10 neurons.

Table 1: Network Architectures Tested and resulting R-values

Net	**Type**	**Neurons per Layer**	**Data Division (divideFcn)**	**Output Processing (processFcns)**	**Epochs**	**R-value (All)**
IKPNN1	fitnet	[10, 10]	dividerand	Layer 1: mapminmax Layer 2: mapminmax	1000	1
IKPNN2	patternnet	[10, 10]	dividerand	Layer 1: mapminmax Layer 2: mapminmax	421	0.99999
IKPNN3	patternnet	[10]	dividerand	Layer 1: mapminmax	498	0.99948
IKPNN4	fitnnet	[10, 10]	dividerand	Layer 1: mapminmax Layer 2: mapminmax	1000	1
FKPNN1	fitnnet	[10, 10]	dividerand	Layer 1: mapminmax Layer 2: mapminmax	1000	1

In the figures below, we visualize the fitness of a few of these networks. Notice that every network tested converged to a very high correlation value. The results in Figs. 7 and 10, with a perfect correlation of R=1, were not difficult to achieve. In Figs. 8 and 11 we see the error histogram, which indicates that once training had completed, nearly all validation checks for the IKPNN had a negligible error. The error band was just slightly larger for the FKPNN, but still well within acceptable ranges.

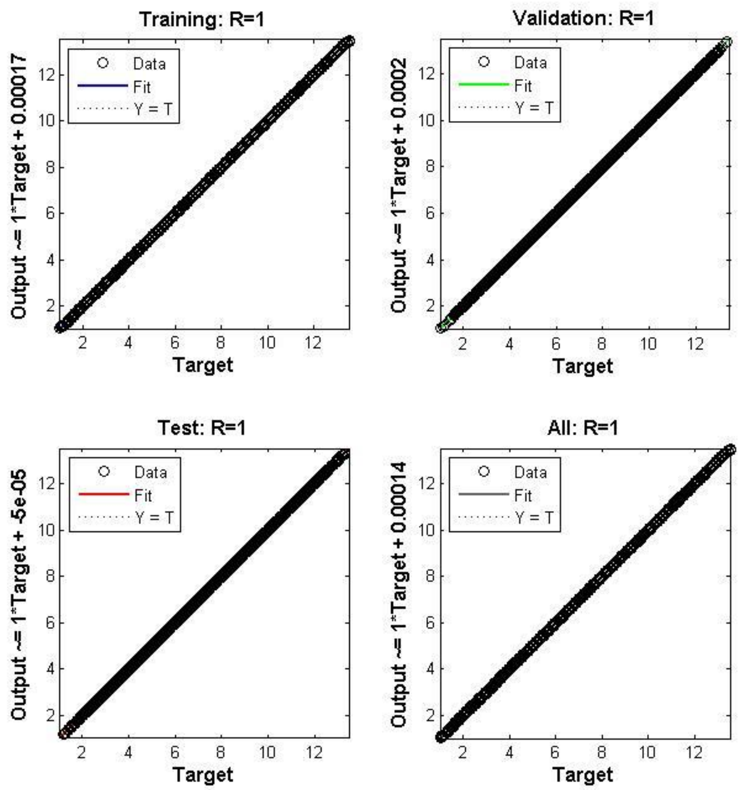


Figure 7. IKPNN

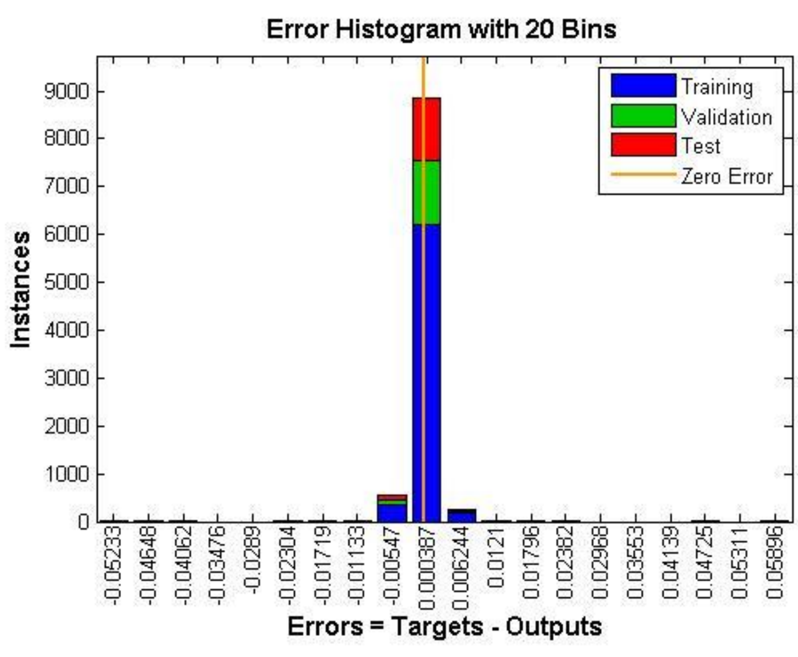


Figure 8. IKPNN

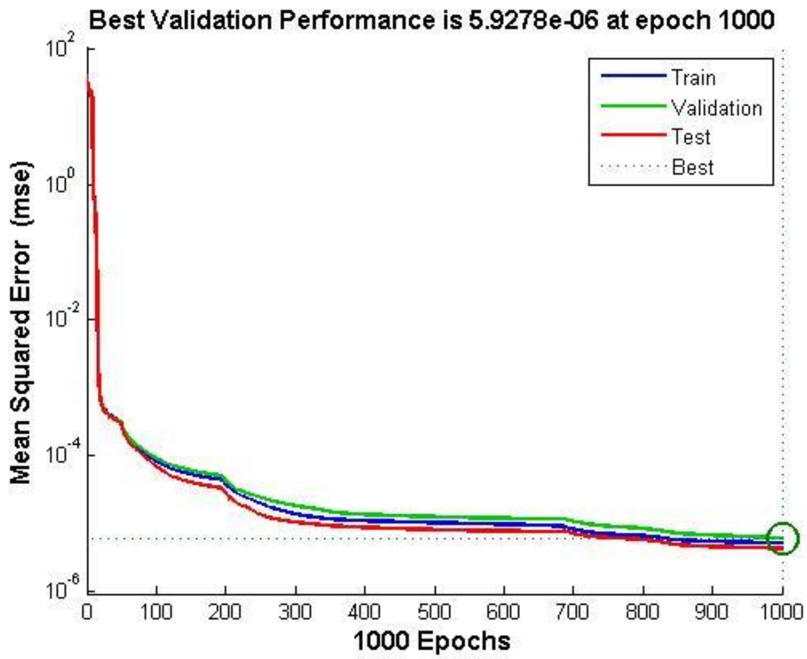


Figure 9. IKPNN

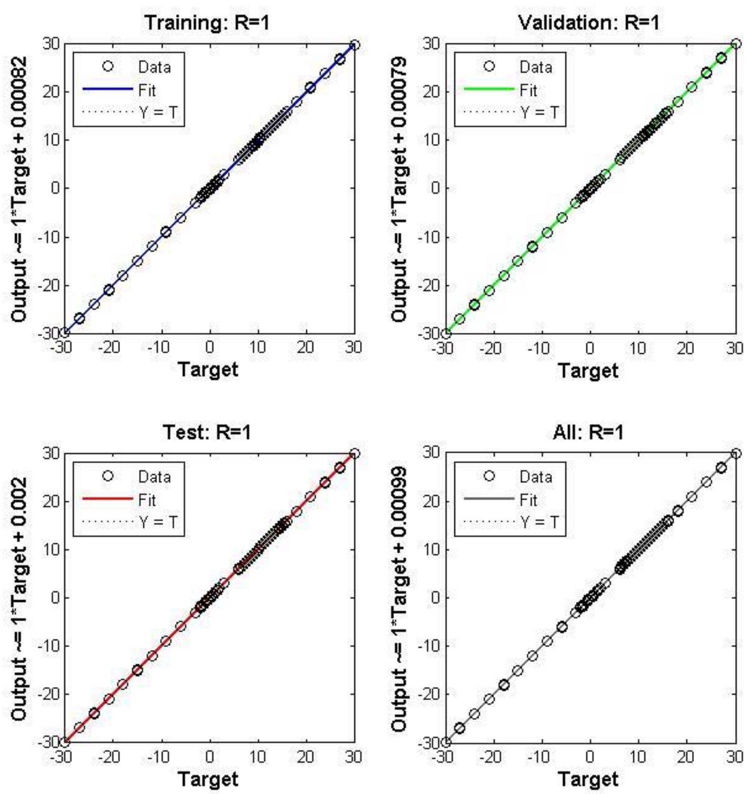


Figure 10. FKPNN

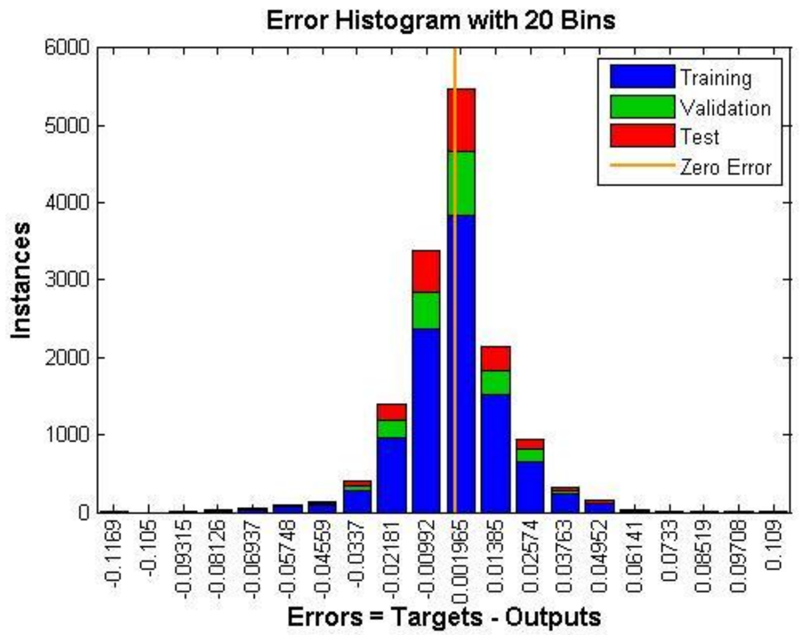


Figure 11. FKPN

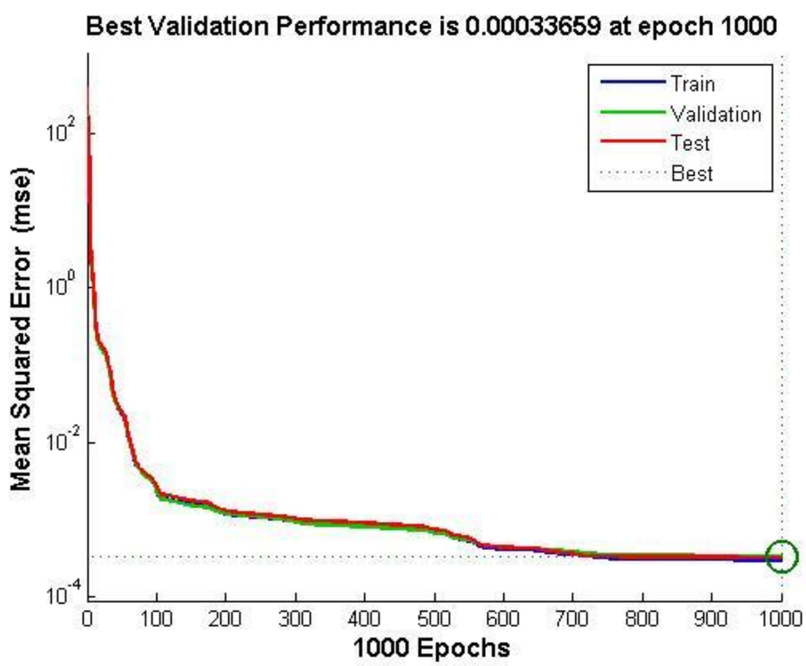


Figure 12. FKPNN

On the other hand, using all the solutions available from the workspace evaluation process (not only the admissible solutions) introduces some singular points and ambiguity in the NN solution. For example, we have theorized that at certain points in the solution field (again, including inadmissible solutions) very little change in the prismatic elements results in large changes in the platform pose. The networks that achieved good results were actually rather large, on the order of 4 layers of 10-30 neurons each. In this manner the ambiguous results seem to be minimized rather effectively, leading to correlation values of about R=0.99999. At the cost of added complexity the results are quite good, although not exactly "perfect." Note that this is unlike the simpler ANNs above that use only the *admissible* solution field.

Finally, regardless of the number of network layers, all of the ANN architectures tested in this report are able to compute and produce highly accurate output nearly instantaneously.

VII. Conclusions

In this report, we have demonstrated the accuracy for which both forward kinematic and inverse kinematic problems can be computed with an Artificial Neural Network. In addition, several Matlab programs were written and are available for interactive usage for future students and researchers.