Preprints of the 2012 20th Mediterranean Conference on Control & Automation (MED), Barcelona, Spain, July 3-6, 2012

An Artificial Intelligence Approach to Forward Kinematics of Stewart Platforms

Antonio Morell, Leopoldo Acosta and Jonay Toledo

Abstract— **The Stewart Platform, one of the most successful and popular parallel robots, has attracted the attention of many researchers in recent decades. The solution of the forward kinematics problem in real–time is one of the key aspects that continues to garner interest. In this paper we propose a new approach for solving this particular case using Support Vector Machines, a popular Machine Learning method for classification and regression. The algorithm involves a data generation and preprocessing off–line phase, and a fast on–line evaluation. The experiments show that this method is very accurate and suitable for use in real–time.**

Index Terms— **Parallel Robots, Stewart Platform, Forward Kinematics, real–time, Support Vector Machines, Support Vector Regression.**

I. INTRODUCTION

EXTENSIVELY used as manipulators in a wide variety of fields, e.g. medicine, optics, astronomy and many industries like aerospace, automotive and aviation, parallel robots have attracted the attention of many researchers in recent decades. They can be classified as closed–loop mechanisms, in contrast with serial robots, which are usually open–loop kinematics chains. Instead, they consist of two frames connected by means of active links, such as prismatic actuators, sometimes called *legs*. The advantages of this type of structure include a greater dexterity, rigidity and positioning capability, good dynamic performance and high load carrying capacity, which makes them very attractive in many applications and fields.

This paper considers a new approach for solving the forward kinematics of Parallel Robots, one of the relevant topics related to parallel manipulators in general, and to the Stewart Platform in particular [1]. Whereas the inverse kinematics problem has a closed–form mathematical solution, the forward case lacks one. It is an under–defined problem with a high–degree nonlinear formulation in which the solution in most cases is not unique. Although the kinematics is one of the most studied aspects of parallel robots, the forward problem continues to gain interest, especially in terms of those methods that can solve it in real–time, as it is essential for a platform characterization and its closed control loop to know the position and orientation (*pose*) by means of the length of the linear actuators attached to the joints.

A review of the literature shows different methods and strategies. One of the main groups are those that describe an analytical solution [2]. Some authors propose a simplification of the model [3], which is solved with numerical methods like Newton–Raphson, or use an interval analysis [4] to work the solution out. However, these kinds of methods lack generalization, since they are proposed for a particular type of platform. Other methods develop a solution to the forward kinematics problem by directly applying a numerical method [5] like Newton–Raphson. In general these kinds of techniques achieve accurate solutions with enough iterations of the algorithms, but may have convergence problems and high computational requirements. Furthermore, there are other solutions that add rotary type sensors or extra links [6] – [7] to obtain additional information, in order to aid and simplify the algorithms. However, these approaches may be difficult to generalize and in some cases will not be applicable due to structural or mechanical constrains. A second group of techniques are those which obtain approximate solutions using a wide variety of methods, from neural networks [8] to polynomial fitting [9]. These types of strategies are more suited to real–time applications and yield good enough approximations for a wide variety of fields.

In this context, we present a different approach for obtaining approximate but accurate solutions in real–time for the forward kinematics of Generalized Stewart Platforms using *Support Vector Machines* (SVMs). Whereas many of the main concepts and features of the SVMs have been present in Artificial Intelligence and Machine Learning since the 60's, they were first formally introduced in the 1992 paper published by Vapnik and his co–workers [10]. SVMs were developed initially as a classification tool, and a few years later were extended to solve regression problems [11]. More specific terms for both cases are *Support Vector Classification* (SVC) and *Support Vector Regression* (SVR). Whereas the former was developed as a pattern recognition tool [12], the latter was proposed for estimating regressions, constructing multidimensional splines and solving linear operator equations [13]. When solving pattern recognition problems, a SVM is able to find a decision rule with good generalization by selecting a small subset of the training data, called the Support Vectors (SVs). It can be shown that optimal separation of the SVs is equivalent to the optimal

Manuscript received January 30, 2012. The authors gratefully acknowledge the contribution of the Spanish Ministry of Science and Technology under Project SAGENIA DPI2010-18349.

A. Morell is with the *Departamento de Ingeniería de Sistemas y Automática y Arquitectura y Tecnología de Computadores (ISAATC), Universidad de La Laguna, La Laguna 38203, Spain* (corresponding author to provide phone: +34–922–318287, e–mail: amorell@isaatc.ull.es

L. Acosta is with the *Departamento de Ingeniería de Sistemas y Automática y Arquitectura y Tecnología de Computadores (ISAATC), Universidad de La Laguna, La Laguna 38203, Spain* (e–mail: leo@isaatc.ull.es

J. Toledo is with the *Departamento de Ingeniería de Sistemas y Automática y Arquitectura y Tecnología de Computadores (ISAATC), Universidad de La Laguna, La Laguna 38203, Spain* (e–mail: jonay@isaatc.ull.es

978-1-4673-2529-5/12/$31.00 ©2012 IEEE

separation of the entire data [11]. Hence, SVMs can be used to summarize the information contained in a data set.

We have chosen the paper presented by M. Tarokh [9] as a reference to compare our results against the exact analytical solution for a Generalized Stewart Platform presented by Liu et al [2]. This paper is organized as follows. Support Vector Methods and Support Vector Regression are described in Section II. Then, after a detailed description of the proposed procedure presented in Section III, the results are discussed in Section IV. Finally, the main conclusions are summarized in Section V.

II. SUPPORT VECTOR METHODS

A. Support Vector Classification

The training algorithm requires a set of p pairs $(\mathbf{x}_i, y_i)$, with a sample vector $\mathbf{x}_i \in \Re^n$ and a class label $y_i \in \{-1, 1\}$. The algorithm finds the parameters of a decision function $D(\mathbf{x})$ using these training samples. An unknown pattern will belong to the first class if $D(\mathbf{x}) > 0$ and to the second otherwise. The direct space decision function is:

$$D(\mathbf{x}_i) = \langle w, \mathbf{x}_i \rangle + b \quad . \tag{1}$$

The goal of the algorithm is to find the parameter w and the bias b that represents the optimal separating hyperplane.

The decision function has a dual space representation, which reduces the number of computations required to train the classifier [10]:

$$D(\mathbf{x}) = \sum_{i=1}^{p} \alpha_i K(\mathbf{x}_i, \mathbf{x}) + b \quad , \tag{2}$$

where the coefficients α_i are the parameters found by the algorithm and $\mathbf{x}_i$ are the training samples. $K(\mathbf{x}_i, \mathbf{x})$ is the kernel function, which allows operations to be performed in the input space rather than in a high dimensional feature space. The kernel performs the non–linear mapping into the feature space, and can be chosen from among the most commonly employed functions, such as polynomials, radial basis functions and certain sigmoid functions.

The samples are sparse over the input space and we can assume that they are linearly separable. Classes separated with a generalized optimal hyperplane are subject to:

$$y_i[\langle w, \mathbf{x}_i \rangle + b] \geq 1 - \xi_i \quad , \quad i = 1 \ldots, p \quad , \tag{3}$$

where w is normal to the hyperplane and ξ_i is a measure of the misclassification error. The perpendicular distance from the hyperplane to the origin is $\frac{b}{\|w\|}$, and the optimal hyperplane is the one that minimizes,

$$\Phi(w, \xi) = \frac{1}{2} \|w\|^2 + C \sum_i \xi_i \quad , \tag{4}$$

which can be solved with Quadratic Programming (QP) optimization. This optimization problem can be transformed by means of the Lagrangian, and the solution is given by the saddle point:

$$\Phi(w, b, \alpha, \xi, \beta) = \frac{1}{2} \|w\|^2 + C \sum_i \xi_i - \sum_{j=1}^{p} \beta_j \xi_j - \\ - \sum_{i=1}^{p} \alpha_i (y_i [\langle w, \mathbf{x}_i \rangle + b] - 1 + \xi_i) \quad , \tag{5}$$

where α, β are the Lagrange multipliers. Thanks to the duality of the classical Lagrangian, (5) is minimized with respect to $w, b, \mathbf{x}$ and maximized with respect to α, β. The dual problem is given by:

$$\max_{\alpha} W(\alpha, \beta) = \max_{\alpha, \beta} (\min_{w, b, \xi} \Phi(w, b, \alpha, \xi, \beta)) \quad . \tag{6}$$

Differentiating (5) and setting the derivatives to zero leads to the dual problem formulation,

$$\max_{\alpha} W(\alpha) = \max_{\alpha} -\frac{1}{2} \sum_{i=1}^{p} \sum_{j=1}^{p} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle - \sum_{k=1}^{p} \alpha_k \quad , \tag{7}$$

subject to the constraints,

$$0 \leq \alpha_i \leq C \quad i = 1, \ldots, p \tag{8a}$$

$$\sum_{j=1}^{p} \alpha_j y_j = 0 \quad . \tag{8b}$$

The given value C is a regularization parameter, which must be chosen to reflect the noise in the data [14].

Maximizing the margin between classes and training samples is an alternative to other optimizing cost function methods, e.g., mean squared error. This provides some important features to SVMs, like automatic capacity tuning of the classification function, the representation of the data relevant for the classification by a small subset (data compression), and uniqueness of the solution.

B. Support Vector Regression

As a regression tool, a SVR employs an alternative loss function, which includes a distance measure. Vapnik proposed the *ε–insensitive loss function* [13], a more sophisticated penalty tool, which defines a region between y_i, the predicted value, and t_i, the actual value. Only when $|t_i - y_i| \geq \varepsilon$ does the function add one of two slack variable penalties, depending on whether they lie *above* (ξ^+) or *below* (ξ^-) the region called the *ε–insensitive tube*. The data to be modeled is a vector $\mathbf{x}_i \in \Re^n$ and a value $y \in \Re$, with a linear function:

$$f(x) = \langle w, \mathbf{x} \rangle + b \quad . \tag{9}$$

The optimal regression function is given by the minimum of the functional,

$$\Phi(w, \xi) = \frac{1}{2} \|w\|^2 + C \sum_i (\xi_i^- + \xi_i^+) \quad , \tag{10}$$

with a given value C.

The primal form can be obtained by introducing the Lagrange multipliers $\alpha_i^+ \geq 0, \alpha_i^- \geq 0, \mu_i^+ \geq 0, \mu_i^- \geq 0 \quad \forall i$:

$$L_P = C\sum_{i=1}^{L}(\xi_i^+ + \xi_i^-) + \frac{1}{2}\|w\|^2 - \sum_{i=1}^{L}(\mu_i^+\xi_i^+ + \mu_i^-\xi_i^-) - \\ - \sum_{i=1}^{L}\alpha_i^+(\varepsilon + \xi_i^+ + y_i - t_i) - \sum_{i=1}^{L}\alpha_i^-(\varepsilon + \xi_i^- - y_i + t_i) \tag{11}$$

Differentiating (11), setting the derivatives to zero and substituting leads to the optimization problem:

$$\max_{\alpha^+,\alpha^-}\left[\sum_{i=1}^{L}(\alpha_i^+ - \alpha_i^-)t_i - \varepsilon\sum_{i=1}^{L}(\alpha_i^+ - \alpha_i^-) - \right. \\ \left. -\frac{1}{2}\sum_{i,j}(\alpha_i^+ - \alpha_i^-)(\alpha_j^+ - \alpha_j^-)\mathbf{x}_i \cdot \mathbf{x}_j\right] \quad , \tag{12}$$

subject to,

$$0 \leq \alpha_i^+ \leq C \tag{13a}$$

$$0 \leq \alpha_i^- \leq C \tag{13b}$$

$$\sum_{i=1}^{L}(\alpha_i^+ - \alpha_i^-) = 0 \quad \forall_i \quad . \tag{13c}$$

Predicted values can be found with:

$$y' = \sum_{i=1}^{L}(\alpha_i^+ - \alpha_i^-)\mathbf{x}_i \cdot \mathbf{x}' + b \quad . \tag{14}$$

Those samples with $0 < \alpha_i < C$ and $\xi_i^+ = 0$ (or $\xi_i^- = 0$) will be elements of the set S of Support Vectors $\mathbf{x}_s$.

Finally, the bias can be obtained as the average over all elements in S:

$$b = \frac{1}{N_s}\sum_{s\in S}\left[t_s - \varepsilon - \sum_{m\in S}^{L}(\alpha_m^+ - \alpha_m^-)\mathbf{x}_m \cdot \mathbf{x}_s\right] \quad . \tag{15}$$

C. SVM Implementation

There are several libraries that implement SVMs, and one of the most widely used is LIBSVM [15]. We have chosen it for its ease of use, outstanding performance and because it can interface with MATLAB, which allows working with a rapid prototyping and testing framework. It is a library under active development, and provides a set of tools that quickly yield acceptable results. It implements a set of SVM formulations, multi–class classification and probability estimates.

III. ALGORITHM

The algorithm consists of an *off–line* and *on–line*. In the former, the algorithm generates a SVM regression model from each division (or *cell*) of the link space, that is partitioned by a fixed amount in each dimension. In the latter phase, the model corresponding to the region whose pose is going to be predicted is recovered from a *look-up table*, and then the forward kinematics can be approximated with simple and fast operations, suitable to real–time applications.

A. Model Representation and Data Generation

As shown in Fig. 1, the generalized configuration is composed of two frames and a set of linear actuators, typically six. A platform is modeled with two 4×6 matrices, A and B, which represent the end effector and fixed frame, respectively, by the Cartesian homogeneous coordinates where each joint is connected. Thus, it is computationally easy to solve the inverse kinematics by simple mathematical operations (16) where needed.

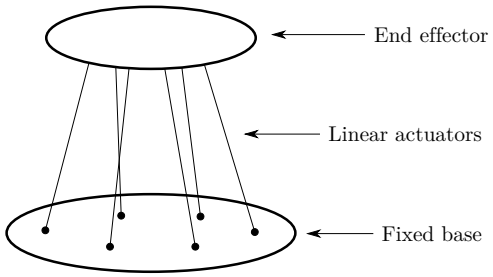

Fig. 1. A class of Parallel Robot.

Consider a fixed base B, with a coordinate frame XYZ attached to it, and another coordinate frame xyz fixed to the end effector A as well. Let us define the *link space* as the 6–D space consisting of the position (as length) of each linear actuator. A *link vector* represents a position in the link space, as a 6–coordinate vector $\mathbf{l} = (l_1 l_2 \cdots l_6)^\top$. Similarly, the *pose space* is the 6–D space that represents the position of the mobile plane (or end effector), as a combination in 3–D Cartesian coordinates of position (x, y, z) and orientation (α, β, γ). A *pose* is defined by the vector $\mathbf{p} = (xyz\alpha\beta\gamma)^\top$.

Given D and R as the position vector and rotation matrix of the end effector, the inverse kinematics can be solved as the Euclidean distance between the positions where each joint is attached:

$$l_i = \sqrt{\|Ra_i + D - b_i\|} \quad i = 1, 2, \cdots, 6 \\ = f_i(\mathbf{p}) \quad , \tag{16}$$

where a_i and b_i are the *i*–th row of the matrix that represents the end effector and fixed frame respectively.

In order to simplify the complexity of the approximation method, the link space needs to be partitioned. The method in the reference paper [9] involves fitting a set of polynomial equations to obtain a parametric model to approximate the forward kinematics solutions in each partition. In this paper we propose a different approach by using SVR machine models to represent the behavior of the platform in a given *region* or partition of the pose and link space. As we stated before, one of the advantages of SVMs is their ability to easily fit complex non–linear functions, but there is a tradeoff with the size of the dataset and the parameters used in terms of training time and accuracy: large training sets will not significantly improve the accuracy of the solutions and will add a time penalty. At the same time, more aggressive parameters will improve the solutions, but will worsen the training time as well. Therefore, the size of the divisions

made to the link space balances the tradeoff, because it affects the complexity of the machines to be tuned.

Once the size of the partitions is established, each region needs to be populated with a meaningful number of poses so that the SVR models can be successfully trained. A generated pose is associated with its link space cell, thanks to the inverse kinematics definition (16). The number of divisions along an axis can be described by the equation:

$$N_i = \frac{l_{i,max} - l_{i,min}}{l_c} \quad , \quad i = 1, 2, \cdots, 6 \quad , \qquad (17)$$

where l_c defines the length of each cell. Each generated pose must be tested to discard those that are invalid. A pose is valid if its corresponding link lengths are within the feasible range, and if they do not violate any platform–specific mechanical constraints, e.g., joint limitations or link collisions.

Randomly generated poses are a better choice for yielding a valid dataset than regularly spaced intervals, which will not produce uniform link space coverage due to the nonlinear mapping of the forward kinematics. We have chosen a pseudorandom number generator that belongs to the TGFSR (*Twisted Generalized Feedback Shift Register*) family, introduced in 2006 by L'Ecuyer and Panneton [16], called WELL (*Well Equidistributed Long-period Linear*), a class of robust generators with very good equidistribution, periods, speed and other interesting properties.

A cell of the link space is mapped into a region of the pose space, that can be identified by a unique index, which resembles a change of numerical base. For a given link configuration $L_k = (l_{k1}, l_{k2}, l_{k3}, l_{k4}, l_{k5}, l_{k6})^T$, the index can be obtained by:

$$\text{cell}_k = \sum_{i=1}^{6} \left(\left\lfloor \frac{l_{ki} - l_{i,min}}{l_c} \right\rfloor (N_i)^{i-1} \right) + 1 \quad , \qquad (18)$$

where $l_{i,min}$ is the minimal length or position of the *i*–th link and N_i is the number of cells in each dimension, given by (17). Then, the valid generated poses can be associated by their region index and stored based on this index so they can be processed in the next step.

B. Classification

It should be noted that some regions will represent more than one forward kinematics solution, so before starting the SVR training, they need to be identified and classified. As shown in Fig. 2 (divided into two 3D representations for ease of visualization), the stored data are organized by clusters that need to be identified. In this case they appear overlapped because of the 2D representation, but in other regions they will look like a homogeneous cloud.

The number of clusters to identify, K_{max} partially depends on the structure of the parallel robot: more general configurations of the platform will have more solutions in a given region, i.e. 6–6 (base–end effector link connections) compared to more constrained versions like 6–3 or 3–3. For the considered platform we have set $K_{max} = 8$ as in [9].

The method chosen to estimate the number of clusters for each region starts by fitting a Gaussian Mixture model

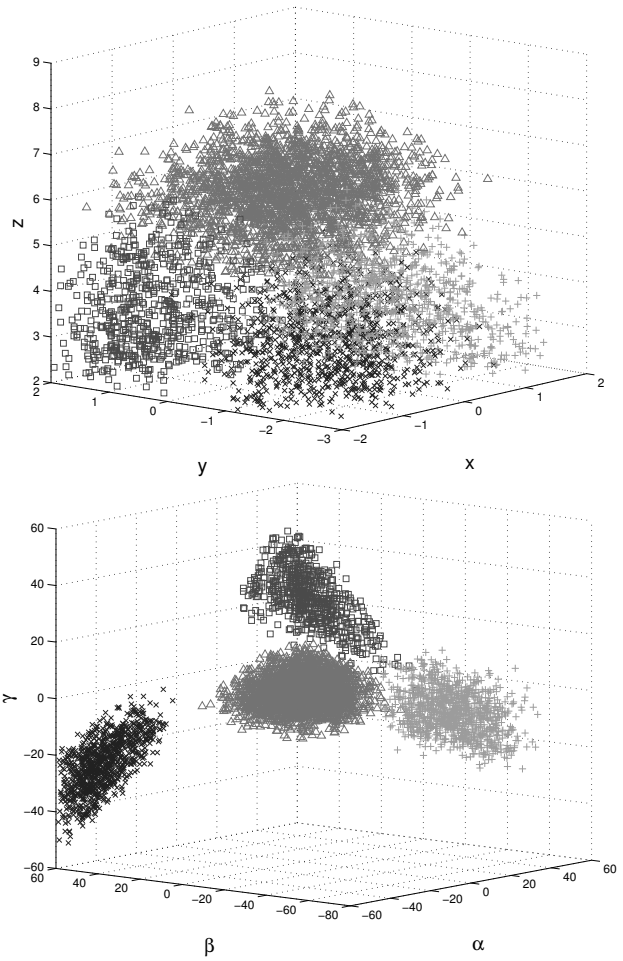

Fig. 2. 3D representation of region 1: top – position subspace; bottom – orientation subspace.

using an Expectation Maximization (EM) algorithm [17] with maximum likelihood estimates, and then applying a measure of the relative goodness of fit of a statistical model called *Akaike Information Criterion* (AIC). Developed by Akaike [18], it is widely used in model selection because it is an estimate of the expected Kullback–Leibler information of a fitted model. A simple way to apply this criterion is by an iterative process, where the Gaussian Mixture model is fitted with *k* components, and where *k* iterates from 1 to the previously set K_{max} value. In each iteration the AIC, a dimensionless quantity, is calculated by the following expression:

$$AIC = 2P - 2\ln(L) \quad , \qquad (19)$$

where P is the number of parameters in the statistical model and L is the maximized value of the likelihood function for the estimated model, obtained with the EM algorithm.

By means of a simple heuristic, it is possible to select a value for K_{max} using the AIC. A procedure like the *Elbow method* does not guarantee the most appropriate guess of the number of clusters: when the data cloud is homogeneous and the points are spread across the region, the presence of an *elbow* point between two iterations might not be a proper

estimation of K_{max}. One possible approach to improve the estimation consists of setting a threshold to find a cutoff iteration from which increasing the number of clusters does not improve the fit significantly. If that iteration is greater than the elbow point we have found that setting K_{max} as the midpoint gives a good guess regarding the number of clusters.

The main advantage of this procedure for estimating K_{max} is that we can calculate the posterior probability of each component from the fitted Gaussian Mixture model. Hence, in order to classify each pose we only need to determine the cluster in which its posterior probability is a maximum.

C. Model Training and Lookup Table

After the clustering process, the data is ready to be used as training samples. An SVR is a MISO system, which supports a very large number of inputs as data features. The output or target value is predicted with a decision function, applied to a data instance. Since a trained machine will output one predicted value, then for every cluster we need to train one SVR model for each dimension $[\, xyz\alpha\beta\gamma \,]$ using the same data instances. Therefore, a given cluster will need a total of six trained SVR machines to represent an estimated pose. The training parameters will be described in Section IV.

Link data must be scaled to avoid numerical difficulties and the negative effect of greater numeric ranges against those which happen to be smaller. Both training and testing data must be scaled with the same method and range, e.g., $[\,-1, 1\,]$ or $[\,0, 1\,]$. In the experiments described in the next section we have used the former to adjust the input data.

Since the method produces a large number of models, we need to populate an indexed data structure or *lookup table* in the *off–line* phase to store them. The morphology of this lookup table should allow for fast indexation. Given that we can apply (18) to obtain the address of the *k*–th element of the input vector, the structure can be a sequence of elements that contains the trained model parameters. As the input links in the *on–line* phase must be scaled according to the training data to obtain a valid prediction, the scaling information used for training is also stored. This ends the *off–line* phase, which only needs to be executed once for a single parallel robot.

D. On–line pose parameter estimation

Given a link vector $\mathbf{l}$, the address of the lookup table where the model parameters are stored can be obtained by applying (18). Then, after scaling the link coordinates, each pose space parameter is predicted using $\mathbf{l}$ with the LIBSVM function `svmpredict`. The time to retrieve the six model parameters and generate the estimation is negligible.

The mean error obtained after the experiments is small enough to satisfy a wide variety of applications, but if needed, the accuracy of the approximation can be improved by a correction step. Let $\hat{\mathbf{p}}$ be the pose parameters estimated for the $\mathbf{l}$ input vector after the *on–line* phase. We then apply the inverse kinematics expression (16) to obtain the link lengths $\hat{\mathbf{l}}$ associated with the pose estimation $\hat{\mathbf{p}}$. The difference between the target and the estimated link lengths can be calculated as $\Delta\mathbf{l} = (\mathbf{l} - \hat{\mathbf{l}})$, and can be added to $\mathbf{l}$ to obtain a new target link configuration $\lambda = \mathbf{l} + \Delta\mathbf{l}$. Finally, solving the forward kinematics of the corrected link lengths λ will give an improved approximation of the initial link vector $\mathbf{l}$.

IV. RESULTS AND DISCUSSION

The experiments were made on a workstation with an Intel Xeon CPU E5440@2.83 GHz, 16 GB of RAM, with Linux kernel version 2.6.24–24–generic and MATLAB R2011B, both 64–bit versions. The most time–consuming steps of this method are the classification and the fitting with SVR machines. The data generation took 80 seconds, and the total training time was 11 255 seconds (about 3 hours). Since regions are independent, some steps like classification and SVR training processes can benefit from parallelization, but for the purpose of this paper we have implemented a non-parallel version of the method. Whereas the implementation was mainly written in MATLAB, the random pose generator was coded in C++ and compiled as a MATLAB EXecutable (*MEX*) binary file to reduce the total computation time. The LIBSVM library also provides a MATLAB implementation of its functions with MATLAB *MEX*–files. The resulting lookup table after the end of the *off–line* phase occupied about 57 MB of memory.

The method was tested with the experimental model proposed in [2], where the authors solved the forward kinematics of a parallel manipulator analytically. The link space was decomposed into cells of length $lc = 3.5\ m$, which gives, applying (17), $N_i = 2$ divisions along each dimension. Therefore, the number of cells and regions is $2^6 = 64$. The pose generation was carried out as described in Section III-A, with random data generated within the ranges

$$\begin{array}{llllll} x = [\,-2 & 2\,] & (m) & \alpha = [\,-60 & 60\,] & (^\circ) \\ y = [\,-2.5 & 2\,] & (m) & \beta = [\,-80 & 60\,] & (^\circ) \\ z = [\;\; 2 & 13\,] & (m) & \gamma = [\,-70 & 70\,] & (^\circ) \end{array} \quad , \tag{20}$$

according to the extreme link configurations reported in [2].

When exploring the best training settings for SVRs, we found that the Radial Basis Function (RBF) kernel gave us the best accuracy/training time ratio, especially as compared to the sigmoid kernel. Given the cost of adjusting each one individually, all the SVRs were trained with the same parameters. In the model training function, `svmtrain`, we choose the ε–SVM method, with the parameter $\gamma = 0.1$ (the width of the Gaussian basis function) and the cost equal to 1000 (C parameter in loss function). Also, we set the tolerance of the termination criterion equal to 5×10^{-3} and the epsilon parameter of the loss function equal to 10^{-3}.

There are six configurations of the linear actuators that represent extreme positions of the end effector, relevant when designing a parallel manipulator, since they provide an insight into its workspace: the *lowest and highest position* of the mobile plane, where all of the actuators are at their minimum/maximum lengths; the *most tilted position* (with two asymmetric cases) where one pair of the linear actuators is at its maximum/minimum length, while the other two pairs

TABLE I

COMPARISON OF BOTH METHODS AND THE ANALYTICAL SOLUTION

	$x\ (m)$	$y\ (m)$	$z\ (m)$	$\alpha\ (^\circ)$	$\beta\ (^\circ)$	$\gamma\ (^\circ)$
			Lowest position			
A	0.0000	0.0000	2.6460	0.0000	0.0000	0.0000
P	0.0000	0.0000	2.6460	0.0000	0.0000	0.0000
S	0.0000	0.0000	2.6458	0.0006	0.0001	0.0000
			Highest position			
A	0.0000	0.0000	12.9600	0.0000	0.0000	0.0000
P	0.0000	0.0000	12.9610	0.0000	0.0000	0.0000
S	0.0005	−0.0008	12.9610	0.0005	0.0009	−0.0016
			Most Tilted position #1			
A	−1.2360	−2.1420	5.5030	58.9900	−73.8400	−46.0600
P	−1.2390	−2.1390	5.5060	58.9900	−74.0000	−46.0700
S	−1.2369	−2.1421	5.5036	58.9890	−73.8400	−46.0520
			Most Tilted position #2			
A	−0.6930	−1.2000	10.4950	−47.7200	39.4200	−18.0100
S	−0.6939	−1.2017	10.4940	−47.7280	39.4280	−18.0160
			Most Twisted position #1			
A	0.0000	0.0000	7.1920	0.0000	0.0000	68.3620
P	0.0000	0.0000	7.1940	0.0000	0.0000	68.3400
S	0.0001	0.0000	7.1924	0.0012	0.0000	68.3590
			Most Twisted position #2			
A	0.0000	0.0000	7.1920	0.0000	0.0000	−68.3620
S	0.0000	0.0001	7.1923	−0.0001	−0.0010	−68.3600

A : Analytical solution [2]
P : Polynomial method [9]
S : SVR machine method proposed

are at their minimum/maximum lengths; and the *most twisted position*, where the three odd actuators are at their minimum length with the even ones extended to their maximum (with a symmetric opposite case).

Table I shows a comparison of the proposed approach and the polynomial method described in [9] against the exact analytical solution presented in [2]. There were no data available to compare with the polynomial method for the most twisted and most tilted position symmetric and asymmetric cases. It can be seen that, while our method has in some cases a slight decimal error, it actually yields a very good approximation of the exact solution. All results are presented with the correction described in Section III-D. Finally, the experiments yielded an average *on–line* evaluation time of 4.87 milliseconds, enough for real–time operation.

V. CONCLUSION

In this paper we have proposed the use of Support Vector Regression Machines for solving a relevant topic within the parallel robotics field. Unlike other methods, the proposed algorithm is not dependent on the geometry of the platform. Also, the results have shown that it can be applied to problems that require high accuracy. Improvements can be made, however. Smaller cell sizes would provide a rich variety of linear actuators configurations, but the time needed to populate the regions with valid poses would increase as well. The *on–line* phase is fast and suitable for real–time applications, and like the accuracy, the computation and evaluation time can be improved with an implementation in a compiled language like C++. Furthermore, it does not have high memory requirements as it only needs to accommodate the lookup table. Finally, the accuracy of the solutions can be easily improved adjusting the parameters of the SVRs, at the cost of more *off–line* training time.

REFERENCES

[1] B. Dasgupta and T. S. Mruthyunjaya, "The Stewart platform manipulator: a review," *Mechanism and Machine Theory*, vol. 35, no. 1, pp. 15–40, 2000.

[2] K. Liu, J. M. Fitzgerald, and F. L. Lewis, "Kinematic analysis of a Stewart platform manipulator," *IEEE Transactions on Industrial Electronics*, vol. 40, no. 2, pp. 282–293, 1993.

[3] P. Nanua, K. Waldron, and V. Murthy, "Direct Kinematic solution of a Stewart Platform," *Robotics and Automation, IEEE Transactions on*, vol. 6, no. 4, pp. 438–444, August 1990.

[4] J.-P. Merlet, "Solving the Forward Kinematics of a Gough–Type Parallel Manipulator with Interval Analysis," *International Journal of Robotic Research*, vol. 23 (3), pp. 221–235, March 2004.

[5] J. E. Dieudonne, R. V. Parrish, and R. E. Bardusch, "An actuator extension transformation for a motion simulator and an inverse transformation applying Newton–Raphson's method," NASA Langley Research Center, Hampton, Va., Technical Report NASA-TN-D-7067, 1972.

[6] J.-P. Merlet, "Closed–form resolution of the direct kinematics of parallel manipulators using extra sensors data," in *Robotics and Automation, 1993. Proceedings., IEEE International Conference on*, vol. 1, may 1993, pp. 200–204.

[7] S.-H. Chen and L.-C. Fu, "The Forward Kinematics of the 6–6 Stewart Platform Using Extra Sensors," in *Systems, Man and Cybernetics, 2006. SMC '06. IEEE International Conference on*, vol. 6, oct. 2006, pp. 4671–4676.

[8] P. J. Parikh and S. S. Lam, "Solving the forward kinematics problem in parallel manipulators using an iterative artificial neural network strategy," *The International Journal of Advanced Manufacturing Technology*, vol. 40, pp. 595–606, 2009, 10.1007/s00170-007-1360-x. [Online]. Available: http://dx.doi.org/10.1007/s00170-007-1360-x

[9] M. Tarokh, "Real Time Forward Kinematics Solutions for General Stewart Platforms," in *Robotics and Automation, 2007 IEEE International Conference on*, april 2007, pp. 901–906.

[10] B. E. Boser, I. Guyon, and V. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the 5th annual ACM Workshop on Computational Learning Theory*. ACM Press, 1992, pp. 144–152.

[11] V. Vapnik, S. E. Golowich, and A. Smola, "Support vector method for function approximation, regression estimation, and signal processing," in *Advances in Neural Information Processing Systems 9*. MIT Press, 1996, pp. 281–287.

[12] B. E. Boser, I. Guyon, and V. Vapnik, "Pattern recognition system using support vectors," U.S. Patent 5,649,068, July 15, 1997.

[13] V. Vapnik, *The nature of statistical learning theory*. New York, NY, USA: Springer–Verlag New York, Inc., 1995.

[14] S. R. Gunn, "Support vector machines for classification and regression," School of Electronics and Computer Science, University of Southampton, Southampton, U.K., Tech. Rep., 1998.

[15] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011, Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[16] F. Panneton, P. L'Ecuyer, and M. Matsumoto, "Improved long–period generators based on linear recurrences modulo 2," *ACM Trans. Math. Softw.*, vol. 32, pp. 1–16, March 2006. [Online]. Available: http://doi.acm.org/10.1145/1132973.1132974

[17] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum Likelihood Estimation From Incomplete Data via the EM Algorithm," *Journal of the Royal Statistical Society, Series B*, vol. 39, no. 1, pp. 1–38, 1977.

[18] H. Akaike, "A New Look at the Statistical Model Identification," *IEEE Transactions on Automatic Control*, vol. 19, no. 6, pp. 716–723, 1974.