2023 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)
June 28-30, 2023. Seattle, Washington, USA

Point-Based 3D Virtual Fixture Generating for Image-Guided and Robot-Assisted Surgery in Orthopedics*

Teng Li[1], Armin Badre[2], Hamid D. Taghirad[1,3], and Mahdi Tavakoli[1], *Senior Member, IEEE*

***Abstract*— Virtual fixture (VF) has been playing a vital role in robot-assisted surgeries, such as guiding surgical tools' movement and protecting a beating heart. In orthopedic surgery, preplanned images are often used in the operating room, on which planning curves might be drawn, for instance, to mark out the boundaries for osteophytes to be removed. These curves can be used to generate VF to assist in removing osteophytes during the operation. A challenge is that the hand-drawn curves usually have a random shape and cannot be mathematically represented by equations, thus most of the existing algorithms will not work in this scenario. In this paper, an algorithm of VF generating based on point clouds is presented, with which VF can be generated directly from cloud points, for example, point clouds of hand-drawn curves extracted from an image. The effectiveness of the VF algorithm is evaluated by a series of simulations and experiments. The VF algorithm is also tested in an image-based scenario and its effectiveness is demonstrated. The presented point-based VF algorithm is promising to be used in various applications in image-guided surgery to generate VF for objects with various shapes.**

I. Introduction

Virtual fixture (VF), also known as active constraint and first proposed in [1], is usually categorized into two types according to its purpose, that is guidance virtual fixture (GVF) and forbidden-region virtual fixture (FRVF) [2], [3]. Intuitively, the GVF serves like a ruler to assist in drawing a straight line, while the FRVF serves like an armor to prevent tool tip from entering a protected area. Both types play a vital role during various surgical procedures in robot-assisted surgery, such as suturing [4], knot tying [5], dissection [6], either assisting in moving the surgical tool along a trajectory or preventing it from entering a specific area for protecting the objects (*e.g.*, beating heart or nerve) inside [7], [8].

VF is usually generated based on geometric elements, such as lines, planes, surfaces, and volumetric primitives [3]. The vector field approach is the most common one to be used for VF generating, which works for any shape that can be expressed as mathematical equations [9]. The advantage of the vector field approach is that it is simple, straightforward, and stable, while the disadvantage is that it requires an explicit/implicit mathematical representation for the object to be modeled as VF. For objects with regular shapes like cubes and spheres, their mathematical representations can be easily established then the VF can be constructed relatively easily. However, for objects with irregular shapes like a humerus bone or skull, they may not be able to be expressed by equations, then the vector field approach may not work for these objects anymore.

The vector field approach cannot correctly handle situations of being in contact with multiple objects simultaneously and situations of thin objects. To solve this problem, Zilles and Salisbury developed a constraint-based god-object algorithm [9]. In their work, the god-object is a proxy of the haptic interface point (HIP) which is attached to the HIP when it is in free motion. Once the HIP encounters VF (*e.g.*, a virtual wall), the proxy will always remain on the top of the virtual wall and never penetrate into it. This is ensured in their algorithm by applying the Lagrange multiplier technique on a set of active constraints to find the position of the proxy in each servo loop. Meanwhile, the virtual spring and/or damper linkage between the HIP and the proxy will render a haptic force that tries to pull the HIP back out of the virtual wall.

Kapoor *et al.* developed a constrained optimization method for generating VF [10], in which a suitable objective function is required in order to do the optimization. In the method, five basic geometric constraints are established as VF task primitives which can be used for assembling customized VF. With similar techniques, Marinho *et al.* employed a method of vector-field inequalities to generate VF for collision avoidance [11] and guidance in a looping task during suturing [4]. Xia *et al.* developed a constrained optimization framework of VF generating for multi-robot collaborative teleoperation tasks, *e.g.*, knot positioning [5].

There are also some other methods for VF generating for different purposes, such as potential field method for collision avoidance or guidance [12], [13] and nonenergy storing method for a more stable robot behavior [14]. Readers are directed to [3] for a comprehensive review of VF.

In the field of robot-assisted surgery, VF has been widely used due to plenty of advantages, such as reducing surgeons' cognitive load [5], improving surgical performance [15], making the surgical outcome more accurate and safe. Park *et*

*This research is supported in part by the Canada Foundation for Innovation (CFI) under grants LOF 28241 and JELF 35916, in part by the Government of Alberta under grants IAE RCP-12-021 and EDT RCP-17-019-SEG, in part by the Government of Alberta's grant to Centre for Autonomous Systems in Strengthening Future Communities (RCP-19-001-MIF), in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada under grants RTI-2018-00681, RGPIN-2019-04662, and RGPAS-2019-00106.

[1]T. Li, H. D. Taghirad, and M. Tavakoli are with the Department of Electrical and Computer Engineering, Faculty of Engineering, University of Alberta, Edmonton T6G 1H9, Alberta, Canada. E-mail: {teng4, h.taghirad, mahdi.tavakoli}@ualberta.ca

[2]A. Badre is with the Western Upper Limb Facility, Sturgeon Hospital, St. Albert, Alberta, Canada, and the Division of Orthopaedic Surgery, Department of Surgery, Faculty of Medicine & Dentistry, University of Alberta, Edmonton, Alberta, Canada. E-mail: badre@ualberta.ca

[3]H. D. Taghirad is with the Advanced Robotics and Automated Systems (ARAS), Faculty of Electrical Engineering, K. N. Toosi University of Technology, Tehran, Iran. E-mail: taghirad@kntu.ac.ir

al. [16] conducted a preliminary test before applying VF in coronary artery bypass surgery. In the test, VF is generated for a blunt dissection task at a position determined from a preoperative CT scan image, and the VF is a regular plane thus the VF generating is relatively easy in their work.

Ryden *et al.* [7] developed a method to generate VF directly from point cloud to protect the beating heart during surgery. They improved their method in [17], [18], which established a solid foundation for point-based VF algorithm.

In orthopedics, a series of preoperative images of a patient are first acquired before the surgery. Then, some surgical preplans will be made on the images, for example, drawing some curves to mark out areas to remove osteophytes, or to protect nerves inside. During the surgery, these hand-drawn curves can be used to generate VF which can assist in removing osteophytes or protecting the nerves by providing haptic feedback. The main challenge here is that the hand-drawn curves are usually in irregular shapes which may not be able to be presented mathematically by equations that are often required by most of existing VF-generating algorithms.

Inspired by Ryden's work and motivated to solve the challenge mentioned above, in this paper, an algorithm for VF generating from cloud points is developed and presented. The effectiveness of the VF algorithm is evaluated by a series of simulations and experiments on geometric entities with regular or irregular shapes. Lastly, the algorithm is tested in a specific preplanned image-based scenario which can be further generalized to image-based surgery.

The remainder of this paper is organized as follows: Section II is devoted to presenting and explaining the complete point-based VF algorithm. Section III presents simulations, experiments, and corresponding results in different scenarios where the VFs are with various shapes. Some advantages and limitations of the algorithm are discussed in Section IV. Finally, conclusions are provided in Section V.

II. Methods

In this section, an algorithm for point cloud based virtual fixture (VF) generation method is presented, which includes one main algorithm and three embedded proxy algorithms for the proxy in different states. The major parameters used in the VF algorithm are summarized in Table I. The general idea of the VF algorithm is explained as follows. The robot end-effector (EE) position denotes as $\mathbf{P_{HIP}}$ while its virtual proxy denotes as $\mathbf{P_{proxy}}$. As illustrated in Fig. 1, taking the proxy point $\mathbf{P_{proxy}}$ as the center, three spheres with radii ($r_1 < r_2 < r_3$) are defined as proxy regions while a contacting region r_c is determined by $r_c = (r_1+r_2)/2$. When there is no contact between the robot EE and the point cloud, the $\mathbf{P_{proxy}}$ always coincides with $\mathbf{P_{HIP}}$, whereas they may be detached from each other in order to generate force feedback when a contact or penetration occurred. The state of the proxy ($state_{\mathtt{proxy}}$) will be determined as $-1/0/1/2$ based on the relative position relationship between the proxy regions and the point cloud, *i.e.*, no neighbor (-1), in free motion (0), in contact (1), entrenched (2). The last three states may be combined together as a state of in-neighbor

TABLE I: Major parameters and description.

Parameters	Description
$\mathbf{P_{HIP}}$	haptic interface point (= robot EE point)
$\mathbf{P_{proxy}}$	proxy point, the avatar of HIP
$\mathbf{L_{pcloud}}$	point cloud list, the collection of all cloud points
$\mathbf{p}_i$	a single point in the point cloud
N	the number of points in the point cloud
$state_{\mathtt{proxy}}$	proxy state, from {-1,0,1,2}
$r_{\{1,2,3,c\}}$	proxy regions radii, $r_1 < r_c < r_2 < r_3$, $r_c = \frac{(r_1+r_2)}{2}$
r_v	radius of each cloud point (default as 0)
$\hat{\mathbf{s}}$	step vector for proxy movement in each servo loop
$\hat{\mathbf{n}}$	normal vector of the point cloud
$\vec{u}$	a vector pointing from $\mathbf{P}_{\mathtt{proxy}}$ to $\mathbf{P}_{\mathtt{HIP}}$

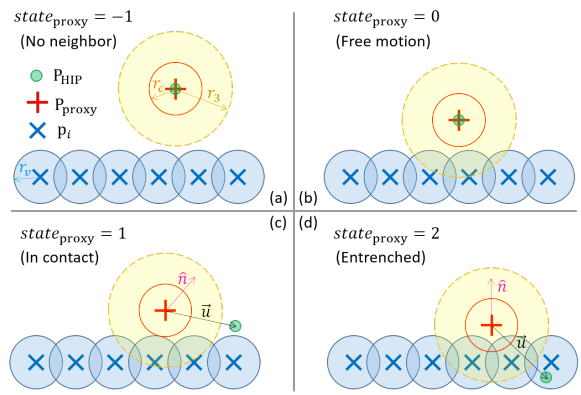


Fig. 1: Illustration of the proxy in different states.

($0/1/2$) for some explanations. For each proxy state, a step vector $\hat{\mathbf{s}}$ will be determined by the proxy algorithms which will be introduced later in this section. Finally, the determined $\hat{\mathbf{s}}$ will be used to move the proxy point at the end of each servo loop while ensuring that it is always on the surface of the point cloud and never penetrates into it.

The general idea described above will be explained in detail in the remaining part of this section based on one main algorithm (algorithm 1) and three proxy algorithms (algorithm 2, algorithm 3, algorithm 4). An illustration of the proxy in different states is shown in Fig. 1. Note that in this paper point cloud refers to a set of points representing a target object, and virtual fixture refers to all areas defined by the cloud points and their radius r_v.

A. Main algorithm

The main algorithm for generating VF from point cloud is presented in algorithm 1 including parameterization realized in this paper. A set of point cloud is acquired in advance and their 3D coordinates are expressed in the robot base frame.

First of all, all parameters are initialized including the initial position of the HIP and the proxy. Note that $\mathbf{P}_{\mathtt{HIP}}$ is initialized as $[-1, 0, 0]$ for the simulations while $[0, 0, 0]$ for the experiments. The proxy point position is initialized as the same as the HIP point position ($\mathbf{P}_{\mathtt{proxy}} = \mathbf{P}_{\mathtt{HIP}}$).

In each servo loop of the VF algorithm, the HIP position ($\mathbf{P}_{\mathtt{HIP}}$) will first be updated as the real-time position of the robot EE. Then, the distance de between the proxy position ($\mathbf{P}_{\mathtt{proxy}}$) and each point in the cloud will be calculated. Based

on the distance de, each point in the cloud will be categorized into one of four lists, *i.e.*, list of entrenched ($\mathbf{L}_{\texttt{Entrenched}}$), in contact ($\mathbf{L}_{\texttt{InContact}}$), in free motion ($\mathbf{L}_{\texttt{FreeMotion}}$), and out neighbor ($\mathbf{L}_{\texttt{OutNeighbor}}$), respectively. Note that the first three lists together composed a new list of in-neighbor ($\mathbf{L}_{\texttt{InNeighbor}}$), while all the four lists together composed the whole point cloud. Then, based on the number of points in each of the four lists, the proxy state ($state_{\texttt{proxy}}$) can be determined as one of the four states, *i.e.*, state of entrenched (2), state of in contact (1), state of in free motion (0), state of no neighbor (-1). Note that for high computing efficiency, the point cloud needs to be treated as a whole matrix when calculating the distance de and doing the categorization.

A normal vector ($\hat{\mathbf{n}}$) needs to be determined when the proxy is in state of in contact or entrenched ($state_{\texttt{proxy}} = 1/2$). The normal vector ($\hat{\mathbf{n}}$) is originated from the proxy point, normal to the local surface formed by the point cloud, and pointing outwards. The normal vector ($\hat{\mathbf{n}}$) is determined by all the cloud points that fell in the proxy neighbor region ($\mathbf{L}_{\texttt{InNeighbor}}$). Let $\mathbf{p}_i$, $i = 1, 2, ..., M$ be the points fell in $\mathbf{L}_{\texttt{InNeighbor}}$, then the normal vector $\hat{\mathbf{n}}$ can be determined by

$$\begin{cases} \vec{nk} = \sum_{i=1}^{M} \frac{\mathbf{P}_{\texttt{proxy}} - \mathbf{p}_i}{\|\mathbf{P}_{\texttt{proxy}} - \mathbf{p}_i\|_2} \phi(r) \\ \hat{\mathbf{n}} = \frac{\vec{nk}}{\|\vec{nk}\|_2} \quad \text{(normalization)} \end{cases} \tag{1}$$

where $\phi(r)$ is a modified version of the Wendland function [17], [19] given by (2), which can provide a smoothly and monotonically decreasing between r_1 and r_3.

$$\phi(r) = \begin{cases} 1 & \text{for} \quad r \in [0, r_1] \\ [1 + \frac{4(r-r_1)}{r_3-r_1}](1 - \frac{r-r_1}{r_3-r_1})^4 & \text{for} \quad r \in (r_1, r_3) \\ 0 & \text{for} \quad r \in [r_3, +inf) \end{cases} \tag{2}$$

where $r = \|\mathbf{P}_{\texttt{proxy}} - \mathbf{p}_i\|_2$ is the distance between a cloud point and the proxy point.

Finally, in each of the four states ($state_{\texttt{proxy}} = -1/0/1/2$), a step vector ($\hat{\mathbf{s}}$) of the proxy movements will be determined based on proxy algorithms (algorithm 2, algorithm 3, algorithm 4) that will be introduced subsequently. Once the step vector ($\hat{\mathbf{s}}$) is determined, the proxy point position ($\mathbf{P}_{\texttt{proxy}}$) can be updated correspondingly.

B. Proxy algorithms

In this subsection, three proxy algorithms are presented and explained in detail in order to determine the step vector ($\hat{\mathbf{s}}$) in different proxy states, *i.e.*, algorithm 2 for state of in free motion ($state_{\texttt{proxy}} = 0$), algorithm 3 for state of in contact ($state_{\texttt{proxy}} = 1$), algorithm 4 for state of entrenched and no neighbor ($state_{\texttt{proxy}} = -1/2$). Then, the determined step vector ($\hat{\mathbf{s}}$) will be used to move the proxy a step in each servo loop while ensuring the proxy point does not penetrate into the point cloud.

(a) State of in free motion ($state_{\texttt{proxy}} = 0$)

The proxy movement algorithm for this state is presented in algorithm 2. When the proxy is in free motion state ($state_{\texttt{proxy}} = 0$) and assuming the HIP is going to penetrate

Algorithm 1: Main algorithm

Data: A set of point cloud ($\mathbf{L}_{\texttt{pcloud}}$) is predefined.
Result: A step vector ($\hat{\mathbf{s}}$) is calculated in each servo loop according to different states of the proxy ($state_{\texttt{proxy}}$), which can ensure that the proxy point ($\mathbf{P}_{\texttt{proxy}}$) always remains outside of the point cloud and never penetrates into it.

```
%(Initialization)
P_HIP ← [−1, 0, 0] (for simulations) ;
P_proxy ← P_HIP ;
r_c ← 5 * 10^−3 ;
r_1 ← r_c − 0.01 * 10^−3 ;
r_2 ← r_c + 0.01 * 10^−3 ;
r_3 ← 2 * r_c ;
r_v ← 0 ;
while in a servo loop do
    %(Update HIP position)
    P_HIP ← real-time robot EE position ;
    %(To categorize point cloud)
    for each point p_i in cloud L_pcloud do
        de = ‖p_i − P_proxy‖_2;
        L_Entrenched ← who has de < r_1 + r_v ;
        L_InContact ← who has r_1 + r_v ≤ de ≤ r_2 + r_v ;
        L_FreeMotion ← who has r_2 + r_v < de < r_3 + r_v ;
        L_OutNeighbor ← who has de ≥ r_3 + r_v ;
    end
    Note: Here the for-loop is only for illustration.
    For efficiency, the point cloud needs to be treated
    as a whole matrix when doing the categorization.
    %(To determine proxy state state_proxy)
    if L_Entrenched ≠ null then
        state_proxy = 2 (entrenched) ;
    else
        if L_InContact ≠ null then
            state_proxy = 1 (in contact) ;
        else
            if L_FreeMotion ≠ null then
                state_proxy = 0 (free motion) ;
            else
                state_proxy = −1 (no neighbor) ;
            end
        end
    end
    %(To determine normal vector n̂)
    if state_proxy = 0/1/2 then
        find the normal vector n̂ via (1) and (2) ;
    end
    %(To determine proxy movement step ŝ)
    if state_proxy = −1/0/1/2 then
        To determine ŝ from proxy algorithms
         (algorithm 2, algorithm 3, algorithm 4) ;
        (ŝ determined)
    end
    %(Update proxy position)
    P_proxy = P_proxy + ŝ ;
end
```

into the point cloud, the proxy needs to move towards the HIP in aiming to be in contact with the point cloud. In this scenario, the step vector ($\hat{\mathbf{s}}$) for the proxy movement can be determined based on the cloud points fell in $\mathbf{L}_{\texttt{InNeighbor}}$ (note that now $\mathbf{L}_{\texttt{InNeighbor}} = \mathbf{L}_{\texttt{FreeMotion}}$) by solving for $d0$ in (3).

$$r_c + r_v = \|\mathbf{p}_i - \mathbf{P}_{\texttt{proxy}} - d0_i \frac{\vec{u}}{\|\vec{u}\|_2}\|_2 \tag{3}$$

where $\vec{u} = \mathbf{P}_{\texttt{HIP}} - \mathbf{P}_{\texttt{proxy}}$ is a vector pointing from $\mathbf{P}_{\texttt{proxy}}$ to $\mathbf{P}_{\texttt{HIP}}$, and $\mathbf{p}_i$, $i = 1, 2, ..., M$ is the cloud point fell in $\mathbf{L}_{\texttt{InNeighbor}}$, and $d0_i$ is a scalar step size needs to be solved for the ith point in $\mathbf{L}_{\texttt{InNeighbor}}$. Therefore, this procedure has to be done M times. After that, step size d will be determined by the minimum value of $d0_i$, *i.e.*, $d = min(d0_i)$. Then, the step vector ($\hat{\mathbf{s}}$) will be determined by $\hat{\mathbf{s}} = d * \frac{\vec{u}}{\|\vec{u}\|}$ which means to bring the proxy point a step towards the HIP point.

Before updating the proxy point position, three special scenarios may need to be considered:

- If $d = 0$ and the HIP is inside of the point cloud, then a projection vector $\vec{u}_p$, which is obtained by projecting $\vec{u}$ onto the normal plane determined by the normal vector $\hat{\mathbf{n}}$, will be used to determine d (5). In this scenario, the step vector will be determined by $\hat{\mathbf{s}} = d * \frac{\vec{u}_p}{\|\vec{u}_p\|}$.
- If the HIP is outside of the point cloud and $\frac{1}{M}\sum_{i=1}^{M}\|\mathbf{p}_i - \mathbf{P}_{\texttt{HIP}}\|_2 > \frac{1}{M}\sum_{i=1}^{M}\|\mathbf{p}_i - \mathbf{P}_{\texttt{proxy}}\|_2$ ($\mathbf{p}_i \in \mathbf{L}_{\texttt{InNeighbor}}$), *i.e.*, the HIP is moving away from the point cloud, then set $\hat{\mathbf{s}} = \mathbf{P}_{\texttt{HIP}} - \mathbf{P}_{\texttt{proxy}}$, which means to detach cloud-proxy and attach proxy-HIP by setting $\mathbf{P}_{\texttt{proxy}} = \mathbf{P}_{\texttt{HIP}}$.

The normal vector $\hat{\mathbf{n}}$ can solely determine a normal plane that is going through the proxy point and normal to $\hat{\mathbf{n}}$. The vector $\vec{u} = \mathbf{P}_{\texttt{HIP}} - \mathbf{P}_{\texttt{proxy}}$ pointing from $\mathbf{P}_{\texttt{proxy}}$ to $\mathbf{P}_{\texttt{HIP}}$ can be projected onto the normal plane, and the projection $\vec{u}_p$ can be obtained by

$$\vec{u}_p = \vec{u} - (\hat{\mathbf{n}} \cdot \vec{u})\hat{\mathbf{n}} \tag{4}$$

Based on the projection $\vec{u}_p$, the step size d can be determined by

$$d = \begin{cases} \xi\|\vec{u}_p\|_2 & \text{for} \quad \|\vec{u}_p\|_2 \le r_1 \\ \xi r_1 & \text{for} \quad \|\vec{u}_p\|_2 > r_1 \end{cases} \tag{5}$$

where $0 < \xi \le 1$ is a constant gain used to ensure that one step size is not greater than the smallest proxy region r_1.

(b) State of in contact ($state_{\texttt{proxy}} = 1$)

The proxy movement algorithm for this state is presented in algorithm 3. When the proxy and the point cloud are in contact ($state_{\texttt{proxy}} = 1$), the step size d will be determined by the projection $\vec{u}_p$ via (5). Then, the proxy step vector $\hat{\mathbf{s}}$ will be determined by

$$\hat{\mathbf{s}} = \begin{cases} d * \frac{\vec{u}_p}{\|\vec{u}_p\|_2} & \text{for} \quad \text{HIP is inside of VF} \\ d * \frac{\vec{u}}{\|\vec{u}\|_2} & \text{for} \quad \text{HIP is outside of VF} \end{cases} \tag{6}$$

where d is the step size, $\vec{u}_p$ indicates a direction tangential to the VF, $\vec{u}$ indicates a direction pointing from the proxy to the HIP, while whether the HIP is inside or outside of VF is determined by the angle between the vector $\hat{\mathbf{n}}$ and $\vec{u}$.

Algorithm 2: Proxy movement step ($state_{\texttt{proxy}} = 0$)

Result: To determine step vector $\hat{\mathbf{s}}$ for proxy state of in free motion ($state_{\texttt{proxy}} = 0$).

```
%(When proxy state is in free motion)
if state_proxy = 0 then
    if P_proxy = P_HIP then
        ŝ = [0, 0, 0] ;
    else
        if P_proxy ≠ P_HIP then
            d = min(d0_i), if min(d0_i) < ||u|| ;
            d = ||u||, if min(d0_i) ≥ ||u|| ;
            ŝ = d * u/||u|| ;
        end
        %(To do a special case-1 check on ŝ)
        if d = 0 & HIP inside VF then
            To determine d via projection u_p (5) ;
            ŝ = d * u_p/||u_p|| ;
        end
        %(To do a special case-2 check on ŝ)
        if HIP is moving away from point cloud then
            ŝ = P_HIP − P_proxy ;
        end
        (ŝ determined)
    end
end
```

Algorithm 3: Proxy movement step ($state_{\texttt{proxy}} = 1$)

Result: To determine step vector $\hat{\mathbf{s}}$ for proxy state of in contact ($state_{\texttt{proxy}} = 1$).

```
%(When proxy state is in contact)
if state_proxy = 1 then
    HIP inside/outside VF ← angle between n̂ & u ;
    projection vector u_p ← from n̂ & u ;
    if ||u_p|| ≤ r_1 then
        d = ξ||u_p|| ;
    else
        if ||u_p|| > r_1 then
            d = ξr_1 ;
        end
    end
    (d determined)
    if HIP inside VF then
        ŝ = d * u_p/||u_p|| ;
    else
        if HIP outside VF then
            ŝ = d * u/||u|| ;
        end
    end
    (ŝ determined)
end
```

(c) State of entrenched ($state_{\texttt{proxy}} = 2$)

The proxy algorithm for this state is presented in algorithm 4. When the proxy is entrenched into the point cloud ($state_{\texttt{proxy}} = 2$) occasionally, the proxy needs to be moved onto the top of the point cloud surface with a single step. The step size d can be determined by solving for $d2_i$ in (7).

$$r_c + r_v = \|\mathbf{p}_i - \mathbf{P}_{\texttt{proxy}} - d2_i * \hat{\mathbf{n}}\|_2 \tag{7}$$

where $\mathbf{p}_i \in \mathbf{L}_{\texttt{Entrenched}}$, $i = 1, 2, ..., M$, and $d2_i$ is a scalar step size corresponding to the ith cloud point. Therefore, this procedure has to be done M times. Then, the step size d will be determined by the maximum value of $d2_i$, *i.e.*, $d = max(d2_i)$, which means that a max step size will be used to bring the proxy point out of the point cloud along the direction of the normal vector $\hat{\mathbf{n}}$.

(d) State of no neighbor ($state_{\texttt{proxy}} = -1$)

When there is no cloud point in the neighbor region of the proxy ($state_{\texttt{proxy}} = -1$), simply set $\hat{\mathbf{s}} = \mathbf{P}_{\texttt{HIP}} - \mathbf{P}_{\texttt{proxy}}$. This means that the proxy point ($\mathbf{P}_{\texttt{proxy}}$) always coincides with the HIP point ($\mathbf{P}_{\texttt{HIP}}$).

Algorithm 4: Proxy movement step ($state_{\texttt{proxy}} = -1/2$)

Result: To determine step vector $\hat{\mathbf{s}}$ for proxy state of entrenched ($state_{\texttt{proxy}} = 2$) and no neighbor ($state_{\texttt{proxy}} = -1$).

%(When proxy state is entrenched)
if $state_{\texttt{proxy}} = 2$ **then**
  $d = max(d2_i)$ from (7) ;
  $\hat{\mathbf{s}} = d * \hat{\mathbf{n}}$;
  ($\hat{\mathbf{s}}$ determined)
end
%(When proxy state is no neighbor)
if $state_{\texttt{proxy}} = -1$ **then**
  $\hat{\mathbf{s}} = \mathbf{P}_{\texttt{HIP}} - \mathbf{P}_{\texttt{proxy}}$;
  ($\hat{\mathbf{s}}$ determined)
end

C. Virtual force rendering

Once the proxy point $\mathbf{P}_{\texttt{proxy}}$ is determined in each servo loop, the virtual force can be rendered based on the coordinates of the HIP point ($\mathbf{P}_{\texttt{HIP}}$) and the proxy point ($\mathbf{P}_{\texttt{proxy}}$). The force rendering algorithm can be expressed as

$$\mathbf{F_v} = \mathbf{K}(\mathbf{P_{proxy}} - \mathbf{P_{HIP}}) \tag{8}$$

where $\mathbf{K} \in \mathbb{R}^{3\times3}$ is a diagonal matrix indicating the stiffness of the VF along each axis and can be tuned as necessary, and $\mathbf{F_v}$ is the rendered VF force which will be delivered to the human user via the robot. Note that friction is not rendered for the VF in this work.

For implementing the VF algorithm onto a robot, in this paper, the rendered VF force is directly added into the impedance control law as an independent term since our robot employs an impedance controller. Note that the VF algorithm is independent of the controller design and controller implementation, thus they work independently and do not affect each other in terms of functioning. For robot running with other controllers, *e.g.*, admittance controller, velocity controller, the rendered VF force may need first to be converted to a displacement or velocity by differentiation/integration or an appropriate gain.

III. Simulations, Experiments, and Results

In this section, a series of simulations and experiments are conducted to evaluate the effectiveness of the point-based VF algorithm. The employed point clouds are with various shapes and dimensionality (ranging from 0D to 3D). The corresponding results are presented and analyzed. The last experiment is designed to assess the VF algorithm in a preplanned image scenario which can be generalized to image-guided surgery in orthopedics.

A. Apparatus

The simulations are conducted in MATLAB/Simulink (version R2020a, MathWorks Inc., Natick, MA, USA). The MATLAB/Simulink runs on a computer with a 3.70 GHz Intel(R) Core(TM) i5-9600K CPU and a Windows 10 Education 64-bit operating system. The HIP point (*i.e.*, the robot EE) in the simulations is represented by the mouse pointer controlled by a human user, and the 2D position of the pointer is retrieved at a frequency of about 50 Hz when it moves in a MATLAB figure window.

The experiments are performed on a 7-DOF Franka Emika Panda robot (Franka Emika GmbH, Munich, Germany). The proposed VF algorithm is implemented on the Panda robot via an impedance control interface and C++ code. The Panda robot is controlled on a workstation computer of Intel(R) Core(TM) i5-8400 CPU @ 2.80 GHz $\times$ 6 with the Ubuntu 16.04 LTS (Xenial Xerus) 64-bit operating system. The control rate of the Panda robot is 1000 Hz. MATLAB/Simulink (version R2019a) with Quarc real-time control software (Quanser Inc., Markham, ON, Canada) is used for visualizing the real-time position of the robot EE, the point cloud, and the rendered VF force. The communications between the Robot Operating System (ROS) nodes of the workstation computer and the MATLAB/Simulink (version R2019a) are realized via User Datagram Protocol (UDP) at a frequency of 100 Hz. The demonstrations of simulations and experiments can be found in the supplementary video[1].

B. Simulations

In this section, simulations in four scenarios are conducted separately in aiming to evaluate the point-based VF algorithm. The four scenarios are regarding to point clouds with various shapes and dimensions as follows,

1) a 0D single point, (see Fig. 2a);
2) a 1D line segment, (see Fig. 2b);
3) a 2D teapot, (see Fig. 2c);
4) a 2D hand-drawn Ω symbol from image, (see Fig. 2d).

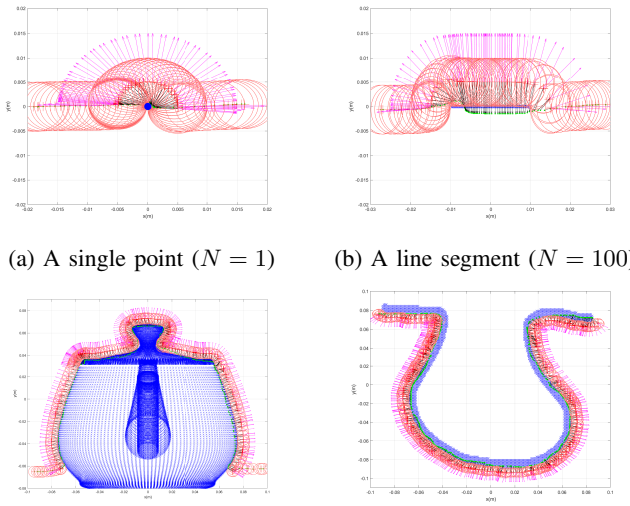
[1]online video link: https://drive.google.com/file/d/1dIR8yllubIRubRXq9nhGsfazgDtSizF6/view?usp=share_link

Note that there is no scenario of a 3D point cloud is considered in the simulations due to the fact that the mouse pointer representing the HIP point cannot be controlled to accurately move in a virtual 3D space. The ability of the VF algorithm in 3D space will be evaluated by experiments.

In each of the four simulated scenarios, as shown in Fig. 2, the HIP point represented by the mouse pointer, moves from left to right in a 2D MATLAB figure window, and during the movement, it will encounter the point cloud. The VF algorithm will calculate the position of the proxy point in real time while the proxy point is expected to remain on the surface of the point cloud and never penetrate into it. The HIP point can penetrate into the point cloud and the relative position between the HIP point and the proxy point will determine the rendered VF force. Note that the VF force is not considered in the simulations but will be rendered in the physical experiments.

The simulation results of the four scenarios are presented in Fig. 2. As can be seen in the figure, as the HIP point (represented as green point) moves along the contour of the point cloud from left to right, the proxy point (represented as the center of the red circle) always stay outside of the point cloud (represented as the blue cross markers) which is expected. The proxy point with its contacting region r_c are represented by a circle in 2D space whose radius is predefined and can be tuned as necessary (in this paper all $r_c = 5$ mm). The normal vector $\hat{\mathbf{n}}$ (represented as arrows in magenta color) should be always orthogonal to the local contour of the point cloud and be pointing outwards.

The simulation results indicated that the proposed VF algorithm works well for point cloud not only with regular shapes (*e.g.*, single point, line) but also with free-style irregular shapes (*e.g.*, teapot, hand-drawn Ω symbol). The effectiveness of the VF algorithm on geometric entities from 0D to 2D is well demonstrated by the simulations.



(a) A single point ($N = 1$) (b) A line segment ($N = 100$)

(c) A 2D teapot ($N = 41472$) (d) An inverted Ω ($N = 2062$)

Fig. 2: Simulation results of implementing the VF algorithm in four different scenarios. The HIP point is represented by green point, the proxy contacting region is represented by red circle while the center represents the proxy point, the normal vector $\hat{\mathbf{n}}$ is represented by magenta arrow. Black arrow represents the vector $\vec{u}$ pointing from the proxy to the HIP. The point cloud of a 2D teapot is obtained from MATLAB via command `pcread('teapot.ply')`, then scaled in this work. Note that the movement of the green point is controlled by a human user via mouse, thus the trajectory is irregular and the speed is not constant.

C. Experiments

Two physical experiments on a 7-DOF robot are designed to evaluate the point-based VF algorithm. In Experiment 1, a set of point cloud with a regular 2D square shape in 3D space (Fig. 3b) is employed, while in Experiment 2 of an image-based scenario, a set of point cloud with a shape of hand-drawn curve (Fig. 3d) from a preplanned 2D image (Fig. 5) is employed. The point clouds are registered in the robot base frame as illustrated in Fig. 3a, 3c.

Experiment 1: 2D square

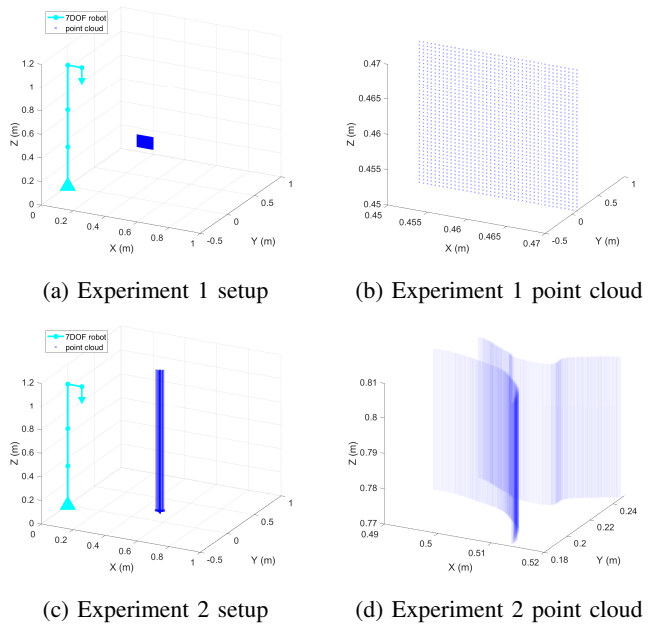
In Experiment 1, a set of point cloud with a 2D square shape is generated in the area of $x = [0.4, 0.5], y = 0.1, z = [0.4, 0.5]$ m in robot base frame, and the step size is 0.5 mm (meaning a density of 2.01 points/mm) for x-axis and z-axis. Therefore, a total of $40401 (= 201 \times 1 \times 201)$ points are generated for the 2D square point cloud.

During the experiment, the user moves the robot EE to probe both sides of the VF (*i.e.*, the 2D square point cloud), and the rendered VF force, as shown in Fig. 4a, is delivered to the user via the robot. In the figure, the green-colored area represents a specific trial. The trajectory of the specific trial is shown in Fig. 4b, with the proxy and its contacting region (red circle and its center), and the normal vector $\hat{\mathbf{n}}$ (magenta arrow) are visualized in a frequency of 10 Hz.

Experiment 1 demonstrates that the VF force can be appropriately rendered, and the robot EE may penetrate into the point cloud but the proxy point should never. The VF force can be rendered on both sides of the 2D square point cloud (*i.e.*, $y < 0.1$, and $y > 0.1$), which indicates that the VF algorithm is valid in 3D space. The results in Experiment 1 show that the VF algorithm in the physical experiment behaves the same as that in the simulations.

Experiment 2: Image-based scenario

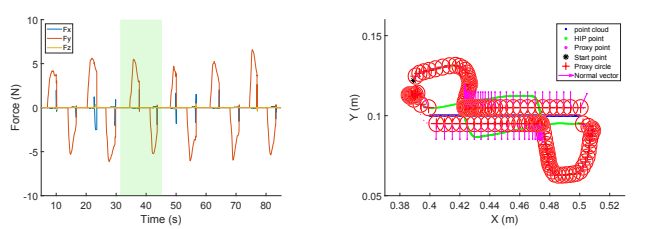
In Experiment 2, the point-based VF algorithm is implemented and evaluated in a preplanned image scenario. Before the experiment, some preparation work is needed. First, a series of 2D CT images are acquired from a patient who has been diagnosed with osteoarthritis and an elbow arthroscopic debridement surgery is required. Then, preplans are conducted on one image as shown in Fig. 5a. In the preplanned image, a hand-drawn curve in red color is shown for planning to remove osteophytes. Nine markers (x1-x9) are marked in the image which will be used later for image-robot registration. Once the preplans are done, the hand-drawn curve is extracted and the corresponding binary image is shown in Fig. 5b. Lastly, a set of point cloud representing the preplans is extracted from the binary image.



(a) Experiment 1 setup (b) Experiment 1 point cloud

(c) Experiment 2 setup (d) Experiment 2 point cloud

Fig. 3: Setup and point cloud patterns for Experiment 1 and Experiment 2. Experiment 1 employs a set of point cloud with a 2D square shape ($N = 40401 = 201 \times 1 \times 201$), while Experiment 2 employs a set of point cloud with a hand-drawn curve shape ($N = 2204$). Note that the VF in Experiment 2 along z-axis is set as continuous and infinite by ignoring the z-axis coordinate during VF force rendering.

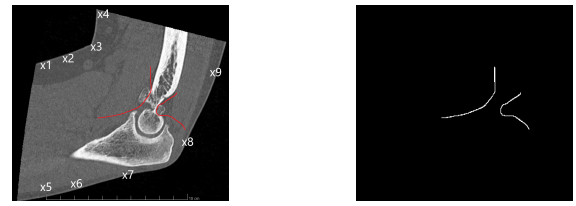


(a) VF force (b) trajectory for colored area

Fig. 4: Experiment 1 of implementing VF on a 2D square point cloud with a size of $N = 40401 = 201 \times 1 \times 201$. The blue cross, green point, and magenta point represent the point cloud, the robot EE (the HIP), and the proxy, respectively. The proxy contacting region and the normal vector $\hat{\mathbf{n}}$ are represented by red circle and magenta arrow, respectively.

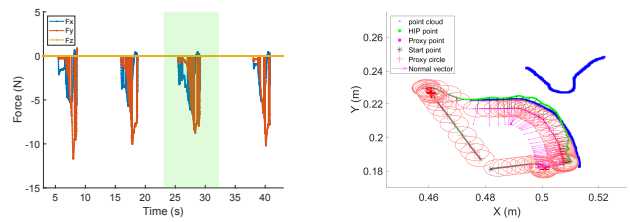
Now we start to do image-robot registration. Considering that the main purpose of this experiment is to evaluate the effectiveness of the VF algorithm, and for simplicity, a paper-printed 2D bone instead of a 3D physical bone is used in the registration. The paper-printed 2D bone is fixed on a horizontal desktop in the workspace of the Panda robot (see Fig. 3c for illustration). The registration is done by using the ordinary least-squares (OLS) method [20] based on the nine markers (x1-x9) on both the paper and the digital image.

The point cloud of the hand-drawn curve extracted from the image consists of 2204 points with a density of 5.3



(a) Preplanned image (b) Curve extracting

Fig. 5: Preplanned 2D image of a patient with osteoarthritis diagnosed and elbow arthroscopic debridement surgery required. Preplanned image size 871×786 (width $\times$ height) pixels. The red curve is hand-drawn for planning to remove osteophytes. The markers x1 to x9 will be used to do an image-robot registration.



(a) VF force (b) Trajectory for colored area

Fig. 6: Experiment 2 of implementing VF on point cloud of a hand-drawn curve with a size of $N = 2204$. The blue cross, green point, and magenta point represent the point cloud, the robot EE (the HIP), and the proxy, respectively. The proxy contacting region and the normal vector $\hat{\mathbf{n}}$ are represented by red circle and magenta arrow, respectively.

points/mm. During the experiment, the user moves the robot EE to probe the point cloud several rounds, and the corresponding VF force is rendered and delivered to the human user as shown in Fig. 6a, in which the green-colored area indicates one specific trial. The robot EE trajectory in the specific trial is represented by green dots as shown in Fig. 6b. In the figure, the proxy and its contacting region (red circle and its center), and the normal vector $\hat{\mathbf{n}}$ (magenta arrow) are visualized in a frequency of 10 Hz. Note that the z-axis is ignored when rendering the VF force, which means that the generated VF along z-axis is continuous and infinitely long.

Experiment 2 demonstrates that the proposed VF algorithm is valid in the image-base scenario. The VF can be generated based on a hand-drawn curve from an image while the VF force can be appropriately rendered in 3D space.

IV. Discussions

In this paper, an algorithm for generating virtual fixture (VF) directly from a set of point cloud data is presented. The effectiveness of the VF algorithm is evaluated by a series of simulations and experiments.

The simulations demonstrated that the VF algorithm works well on point clouds of various geometric entities, *i.e.*, a single point, a line segment, a 2D teapot, and a 2D hand-drawn curve. In Experiment 1, the results demonstrated that

the VF algorithm works well in 3D space by employing a set of point cloud in the shape of a 2D square, while the user can feel the resistant force generated by the VF when touching either side of the square via robot EE. In Experiment 2, 3D VF is generated based on a hand-drawn curve extracted from a preplanned image, and the VF force is appropriately rendered. Both the simulations and the experiments verified the effectiveness of the VF algorithm. Particularly, the results of Experiment 2 showed the possibility to implement the VF algorithm in image-guided surgery.

The VF algorithm is used for static VF in this paper. It should be noted that the algorithm is capable of serving for dynamic VF, *i.e.*, generating and updating VF in an online manner. This can be realized by online updating the point cloud dataset. This feature could be very useful for some surgical scenarios, such as bone burring during arthroscopic surgery, in which case the VF can online update itself based on the real-time shape of the target bone.

One advantage of the point-based VF algorithm is that the VF is generated directly from points, the simplest geometric entity. Therefore, there is no need for 2D/3D surface/volume reconstruction before generating VF which can save a substantial amount of procedures and computations. More importantly, the point cloud can be in any shape.

A limitation of this work is visualization. In our experiments, only the point cloud and the robot EE (*i.e.*, the HIP) is visualized on a monitor during the task. In future work, all key VF features (*e.g.*, proxy point, contacting region, normal vector) in 3D space will be visualized by using the augmented reality (AR) technique, which can better help surgeons to utilize the VF.

V. Conclusions

Virtual fixture (VF) plays an important role in robot-assisted surgeries. A variety of algorithms for generating VF have been developed for various surgical applications. However, generating VF for a free-style curve/surface, *e.g.*, a hand-drawn spline, is still a challenging problem due to the fact that an accurate mathematical function cannot always be found for such types of curves and surfaces. In this paper, a point-based VF algorithm is presented which allows to generate VF directly from the point cloud data. The effectiveness of the algorithm is demonstrated by both simulations and experiments. An experiment in an image-based scenario verified the capability of the algorithm to generate VF based on a hand-drawn curve in an image.

The point-based VF algorithm is promising to be applied in various surgical scenarios in robot-assisted surgery and image-guided surgery, as long as a set of point cloud of the target object can be obtained. In future work, we will implement the VF algorithm into a realistic arthroscopic surgery scenario by using 3D physical bones and develop accurate registration methods for robot-image-bone registration.

References

[1] L. B. Rosenberg, "Virtual fixtures: Perceptual tools for telerobotic manipulation," in *Proceedings of IEEE virtual reality annual international symposium*. IEEE, 1993, pp. 76–82.

[2] J. J. Abbott, P. Marayong, and A. M. Okamura, "Haptic virtual fixtures for robot-assisted manipulation," in *Robotics research*. Springer, 2007, pp. 49–64.

[3] S. A. Bowyer, B. L. Davies, and F. R. y Baena, "Active constraints/virtual fixtures: A survey," *IEEE Transactions on Robotics*, vol. 30, no. 1, pp. 138–157, 2013.

[4] M. M. Marinho, H. Ishida, K. Harada, K. Deie, and M. Mitsuishi, "Virtual fixture assistance for suturing in robot-aided pediatric endoscopic surgery," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 524–531, 2020.

[5] T. Xia, A. Kapoor, P. Kazanzides, and R. Taylor, "A constrained optimization approach to virtual fixtures for multi-robot collaborative teleoperation," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 639–644.

[6] M. Selvaggio, G. A. Fontanelli, F. Ficuciello, L. Villani, and B. Siciliano, "Passive virtual fixtures adaptation in minimally invasive robotic surgery," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3129–3136, 2018.

[7] F. Ryden and H. J. Chizeck, "Forbidden-region virtual fixtures from streaming point clouds: Remotely touching and protecting a beating heart," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 3308–3313.

[8] R. Moccia, C. Iacono, B. Siciliano, and F. Ficuciello, "Vision-based dynamic virtual fixtures for tools collision avoidance in robotic surgery," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1650–1655, 2020.

[9] C. B. Zilles and J. K. Salisbury, "A constraint-based god-object method for haptic display," in *Proceedings 1995 ieee/rsj international conference on intelligent robots and systems. Human robot interaction and cooperative robots*, vol. 3. IEEE, 1995, pp. 146–151.

[10] A. Kapoor, M. Li, and R. H. Taylor, "Constrained control for surgical assistant robots," in *ICRA*, 2006, pp. 231–236.

[11] M. M. Marinho, B. V. Adorno, K. Harada, and M. Mitsuishi, "Dynamic active constraints for surgical robots using vector-field inequalities," *IEEE Transactions on Robotics*, vol. 35, no. 5, pp. 1166–1185, 2019.

[12] N. Y. Chong, T. Kotoku, K. Ohba, and K. Tanie, "Virtual repulsive force field guided coordination for multi-telerobot collaboration," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation*, vol. 1. IEEE, 2001, pp. 1013–1018.

[13] M. Selvaggio, F. Chen, B. Gao, G. Notomista, F. Trapani, and D. Caldwell, "Vision based virtual fixture generation for teleoperated robotic manipulation," in *2016 International Conference on Advanced Robotics and Mechatronics (ICARM)*. IEEE, 2016, pp. 190–195.

[14] R. Kikuuwe, N. Takesue, and H. Fujimoto, "A control framework to generate nonenergy-storing virtual fixtures: Use of simulated plasticity," *IEEE Transactions on Robotics*, vol. 24, no. 4, pp. 781–793, 2008.

[15] T. L. Gibo, L. N. Verner, D. D. Yuh, and A. M. Okamura, "Design considerations and human-machine performance of moving virtual fixtures," in *2009 IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 671–676.

[16] S. Park, R. D. Howe, and D. F. Torchiana, "Virtual fixtures for robotic cardiac surgery," in *International conference on medical image computing and computer-assisted intervention*. Springer, 2001, pp. 1419–1420.

[17] F. Ryden and H. J. Chizeck, "A proxy method for real-time 3-DOF haptic rendering of streaming point cloud data," *IEEE transactions on Haptics*, vol. 6, no. 3, pp. 257–267, 2013.

[18] S. Nia Kosari, F. Rydén, T. S. Lendvay, B. Hannaford, and H. J. Chizeck, "Forbidden region virtual fixtures from streaming point clouds," *Advanced Robotics*, vol. 28, no. 22, pp. 1507–1518, 2014.

[19] H. Wendland, "Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree," *Advances in computational Mathematics*, vol. 4, no. 1, pp. 389–396, 1995.

[20] C. Gaz, M. Cognetti, A. Oliva, P. R. Giordano, and A. De Luca, "Dynamic identification of the Franka Emika Panda robot with retrieval of feasible parameters using penalty-based optimization," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4147–4154, 2019.