




A Virtual Reality Interface for an Autonomous Spray Painting UAV

Anurag Sai Vempati , Harshit Khurana, Vojtech Kabelka, Simon Flueckiger, Roland Siegwart ,
and Paul Beardsley 

Abstract—PaintCopter is an autonomous unmanned aerial vehicle (UAV) capable of spray painting on complex three-dimensional (3D) surfaces. This letter aims to make PaintCopter more user-friendly and to enable more intuitive human-robot interaction. We propose a virtual reality interface that allows the user to immerse in a virtual environment, navigate around the target surface, and paint at desired locations using a virtual spray gun. A realistic paint simulator provides a real-time previsualization of the painting activity that can either be processed right away or stored to a disk for later execution. An efficient optimization based planner uses this information to plan the painting task and execute it. The proposed planner maximizes the paint quality while respecting the spray nozzle constraints and platform dynamics. Our experiments show that the interface allows the user to make precise modifications to the target surface. Finally, we demonstrate the use of virtual reality interface to define a painting mission, and then the PaintCopter carrying out the mission to paint a desired multicolored pattern on a 3D surface.

Index Terms—Virtual reality and interfaces, aerial systems: applications, motion and path planning.

I. INTRODUCTION

PAINTING is a key element of maintaining an artificial structure's visual appeal. It needs to be precise and repeatable to maintain the original artistic value. Robotic painting offers a low-cost solution for painting high-rise buildings and structures by eliminating the need for scaffolding. Furthermore, by automating the entire painting process, humans can avoid working in hazardous environments.

PaintCopter [1] is an autonomous UAV that was designed to be able to autonomously spray paint on complex 3D surfaces.

Manuscript received February 23, 2019; accepted June 8, 2019. Date of publication June 12, 2019; date of current version June 21, 2019. This letter was recommended for publication by Associate Editor M. Ferre and Editor A. M. Okamura upon evaluation of the reviewers' comments. (*Corresponding author: Anurag Sai Vempati.*)

A. S. Vempati is with the Autonomous Systems Lab, ETH Zurich, 8092 Zurich, Switzerland, and also with Disney Research Zurich, 8006 Zurich, Switzerland (e-mail: avempati@ethz.ch).

H. Khurana, V. Kabelka, S. Flueckiger, and P. Beardsley are with Disney Research Zurich, 8006 Zurich, Switzerland (e-mail: hkurana@student.ethz.ch; vojta.kabelka@gmail.com; simoflu@student.ethz.ch; paulbeardsley5@gmail.com).

R. Siegwart is with the Autonomous Systems Lab, ETH Zurich, 8092 Zurich, Switzerland (e-mail: rsiegwart@ethz.ch).

This letter has supplementary downloadable material available at <http://ieeexplore.ieee.org>, provided by the author. This video, playable with VLC, Windows Media Player, and QuickTime, presents a unique virtual reality interface for an autonomous spray painting UAV. The total size of the file is 47.1 MB. Contact avempati@ethz.ch for further questions about this work.

Digital Object Identifier 10.1109/LRA.2019.2922588

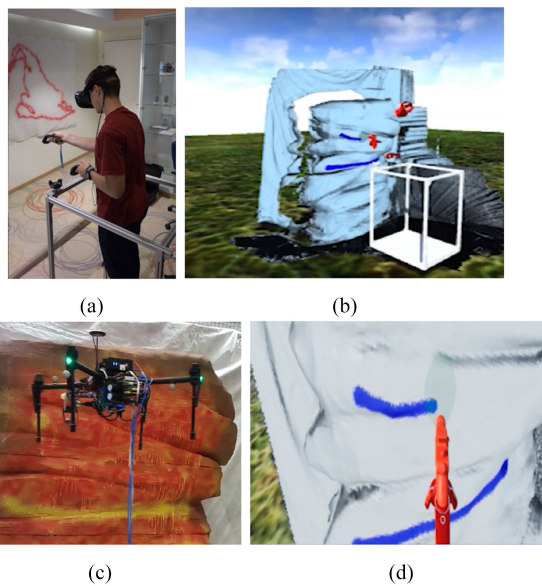


Fig. 1. A user (a) in the virtual environment (b) can intuitively operate the PaintCopter (c) while getting instant feedback of the painting activity (d).

In that work, a painting mission is defined by starting from a 2D line drawing that is projected onto a target surface from a desired viewpoint. Though this is a useful interface, it doesn't provide complete flexibility to the user in being able to accurately modify the target surface. In the real-life scenario, a painter is used to painting on the surface by freely waving a spray gun in order to achieve the desired effect. So, the purpose of the proposed interface is to enable the user to have similar capabilities in controlling the UAV. Such an interface has the benefit of being most intuitive to use, even for an inexperienced user.

For a multi-color texture on a 3D surface, the automatic generation of the painting mission is a complex task [2]. This letter proposes an alternative and manually-guided approach in which a human painter works naturally to paint a surface in a virtual environment, and the human activity is converted to painting commands.

Ideally, the chosen interface should meet the following criteria: (a) Provide a close to real-life painting experience. (b) Be able to capture valuable information such as hand motion, colors used and the produced end result. (c) Provide valuable visual feedback to the user regarding the painting activity.

Virtual Reality (VR) technology has made great progress in recent years. Though traditionally the technology was mainly used

for gaming applications, it is being increasingly used for various robotics applications. VR devices come with a head mounted display (HMD) which allows the user to be fully immersed in a virtual environment. In addition, a pair of hand controllers allow the user to perform actions and manipulation tasks that can be used to command robots in the real world. Moreover, most of the commercial VR devices take advantage of powerful gaming engines such as Unreal and Unity which allow users to develop a wide range of realistic scenarios, intelligent characters and objects with realistic dynamics and kinematics [3]. Hence, a VR interface is chosen as the interface to command PaintCopter.

This work describes the design choices and details regarding various features of the proposed VR interface. The main contributions of this work are:

- Design of an intuitive VR interface to enable high-level control of an autonomous spray painting UAV.
- Development of a paint simulator that can generate realistic renderings of the painting session.
- Formulation of an efficient optimization based planning strategy to generate commands for the UAV.

In the remainder, Section II highlights some related work, Section III describes design of the VR interface and how the data from the VR environment is processed, Section IV describes the paint simulator which generates realistic pre-visualization of the painting activity that serves as a valuable feedback to the user and Section V describes the formulation of the optimization based task planner that provides commands to the UAV. Section VI provides experimental results and finally we conclude with some remarks in Section VII.

II. RELATED WORK

Human-robot interaction is a vast field with several advances made in the design of unique interfaces for many robots. Depending on the robot and the application at hand, the interfaces and the level of human involvement can widely range. In the context of this work, we are mainly interested in interfaces that only provide high-level inputs to autonomous robots. Interfaces for human interaction with UAVs include [4] and [5] where face detection along with hand gesture recognition is used to command a UAV. Such methods, however, have a limited library of possible UAV actions and lack precise control of the trajectory that is needed for an application such as painting 3D surfaces.

Virtual and Augmented reality headsets allow for more immersive interaction with robots. A study on user interfaces for multi-robot scenario [6] shows that virtual reality improves situational awareness without increasing the workload of operators. The authors in [7] use an augmented reality headset that allows the user to provide paths to the UAV by drawing them on a 3D model of the terrain. The live camera feed from the UAV is then streamed back to the user as feedback. VR technology is used to address several UAV applications, such as area coverage and target following in [8], highlighting the endless possibilities for such a technology in robotics.

Painting robots have traditionally been used in industrial setting for surface coating [9], [10]. Robot manipulators are

commonly used for such applications and are predominantly used within the automotive industry. However, there is also a line of research focusing on artistic painting. A visual feedback guided mechanism to paint using acrylic colors is mentioned in [11]. An optimization routine solves for appropriate color and stroke placement on the canvas at each time instant. The authors in [12] use a 6DOF manipulator to produce watercolor paintings while giving the user control to occasionally adjust the parameters such as brush type, dilution etc. during the process. A 7DOF manipulator equipped with a pen is used in [13] to produce line drawings on arbitrary surfaces using impedance control.

Using UAVs for painting is relatively new. UAVs have larger workspace compared to industrial manipulators and can be a lot more dynamic. A tethered platform can fly for longer duration and cover larger areas, making them an ideal choice for painting related applications. In [14], an ink-soaked sponge attached to a tethered UAV is used to paint stipples on a canvas to generate dotted paintings. Our previous work [1] describes a fully autonomous UAV that is capable of painting a reference line drawing on arbitrary 3D surfaces. This work aims to design an immersive and interactive interface for a user to operate such a UAV.

III. VR INTERFACE

In this section, the design of the VR interface and processing of the data from the VR environment are described.

A. Interface Design

The device of choice is HTC Vive. It has an HMD, two base stations and two hand controllers, namely navigation controller and painting controller. The user is physically present in a replica of a painter's cradle. The physical cradle is stationary at all time and it defines the operation space for the user. The cradle design is inspired from what one typically finds on cherry-pickers for lifting painters to high and inaccessible areas.

The virtual environment is developed within Unreal game engine. Even though the software was designed to work with HTC Vive, the proposed system is not limited to work with a particular VR headset. Switching to an alternative VR system would only require modifying the interface to Unreal game engine. The virtual environment consists of a 3D scan model of the target surface and a virtual cradle on which the user stands. The 3D scan model can be obtained either by the UAV [15] or by a Leica Multistation. The virtual cradle is an exact model of the physical one and can be operated with the navigation controller. The cradle has an additional tracker that is used to initialize the pose of the virtual cradle at the start, to make sure that the user always starts by facing the 3D model. A view of the virtual environment and the user in a cradle can be seen in Fig. 1.

The functionality of the virtual cradle is similar to the real-life scenario where the user has to first navigate and reach the target location to paint. Using the trackpad on the navigation controller, the user can navigate anywhere in space and stand in a desired orientation, thus having 4DOF control in positioning the virtual cradle. The painting controller can be used in two different modes, **painting mode** or **viewing mode**. Painting mode

is active when the trackpad is pressed. In this mode, the user can paint on the virtual 3D surface using a virtual spray-gun. Viewing mode is active when the user presses the trigger on the controller. In this mode, the UAV can be commanded to inspect the target surface with its onboard camera. This mode is useful for the user to get feedback of the painting mission before any further modifications to the target surface.

The VR data captures the user activity and it consists of VR state messages (s) which are published at 90 Hz. Each state message stores 6DOF virtual spray-gun pose (nozzle_pose), 4DOF virtual cradle pose (cradle_pose) and paint-hit (paint_hit) locations. Each state message belongs to one of the three different modes: *idle*, *painting* or *viewing*. The state messages are by default in *idle* mode. The state messages are in *painting* or *viewing* mode if the painting controller is in painting or viewing mode respectively. In *idle* mode the navigation controller's pose is published as cradle_pose at all times. In both *viewing* and *painting* mode, the painting controller's pose is used as nozzle_pose. Additionally, in *painting* mode, paint_hit comprises of 3D location along with a normal of the surface where the virtual spray-gun is currently painting.

Visual feedback to the user is vital for improved situational awareness. We modify the appearance of the rock surface in real-time depending on the use-case below. This is accomplished by projecting a color image onto the target surface in Unreal engine. **Visual feedback using a paint simulator** - the paint simulator is used to render the appearance of the rock model from a fixed viewpoint. In addition, the user can see a ray emerging from the virtual spray-gun's nozzle to get an intuition for the paint hit location. **Visual feedback from a live painting mission** - the images arrive from the onboard camera on the UAV and the estimated camera pose is used for obtaining the projection matrix.

The VR interface can be used either in an **offline session** or a **live session**. During the offline session, the VR data can be stored to disk and processed for execution by the UAV at a later point of time. In the live session, the user can command the UAV instantly, as shown in Fig. 1c. Based on the visual feedback, the user has the ability to make the needed adjustments.

B. Processing VR Data

In this section, we describe how the data from VR interface is processed in order to plan a mission for the PaintCopter. VR data is broken down into "segments" (S) and each segment is processed sequentially to plan a "task". Task planning ($PlanTask(S, mode)$) involves processing a segment to generate waypoints for the UAV and the Pan-Tilt Unit (PTU) that controls the spray-gun. If the resulting path is not in collision with the obstacles, the task is executed. More details regarding this can be found in [1]. Depending on whether the VR data is from an offline session or from a live session, it's segmented and processed in different ways. In processing pre-recorded data, a segment consists of all the states from the moment the trigger/trackpad is pressed till the moment it's released. In live session, VR state messages are pushed to a queue (Q) as and when they arrive and are processed 'oldest-first' to avoid skipping any messages. The queue is broken into segments based on a threshold Δ_{seg} .

During the live processing, it's important that the user feels like they are painting instantly without any latencies. However, latencies usually occur due to time consumed in processing each segment and also from the fact that the UAV is usually incapable of painting at the speeds as that of a free waving hand. Having large latencies can deteriorate the user experience. So, the live processing is designed in a way such that the latencies are minimized and the user experience is least affected.

The latency is reduced by making the following choices: (a) As the user is navigating around the target surface, whenever the virtual cradle is farther than a threshold (Δ_{seek}) from the UAV's current position (r), the UAV is commanded to seek the virtual cradle to avoid large latency in getting into start position at the beginning of a viewing/painting activity. (b) When the user is painting on the surface from a very large distance ($\geq \Delta_{far}$), large sections of the surface can be painted in a short time. Painting such a large pattern can take much longer time than what is needed to paint it in the virtual environment. Also, considering the fact that painting from a very large distance to the target surface is unrealistic, we restrict the user from painting on far away target locations. In addition, visual text feedback is sent to the user to move closer to the rock ($SendFeedback()$). (c) Since messages are processed in segments, the UAV doesn't start moving until the hand moves a certain distance. To avoid this, an interrupt is generated when the trigger/trackpad is pressed and the first VR state message is used to move the UAV from its current position to this state. (d) If the user is painting at faster speeds, a virtual UAV pops up in the user's field-of-view, indicating the current UAV position in the real-world. This acts as a feedback that tells the user to paint slower if the UAV is not able to catch up to the speed of the hand. (e) Once the trigger/trackpad is released an interrupt is generated that stops any more incoming VR state messages from entering the queue and the user is notified to wait before painting again. Any new states are accumulated only once the entire queue is empty. This ensures that there is no buffer over-flow as time progresses. Though having the user to wait is undesirable, usually this wait time is less than a few tens of seconds.

Data processing during a live session can be summarized in Algorithm 1, where, back(), pop_front() and push_back() are vector operations for (a) accessing last entry, (b) extracting first entry and (c) adding a new entry. In our experiments, we typically use $\Delta_{seek} = 1m$ and $\Delta_{far} = 1m$.

IV. PAINT SIMULATOR

To get a pre-visualization of the appearance of paint on the target surface, we developed a paint simulator. The need for such a simulator is two-fold: (a) To provide instant feedback to the user in VR environment during the live session and (b) to evaluate the accuracy of the painting when the offline session data is processed and the mission is executed inside the RotorS simulator [16]. Paint simulator uses the nozzle dynamics, the nozzle properties (cone angle and flow rate) and the paint properties (scatter and reflectance) to figure out how the target surface would look like after spray painting. This is achieved, at real-time, by accurately modeling the appearance of the surface (with a certain background color) being sprayed by multiple layers of different

Algorithm 1: Process VR states queue \mathbf{Q} .

```

if  $\mathbf{Q}$ .isEmpty() then PlanTask( $\mathbf{S}$ ,  $\mathbf{S}$ .back().mode); return;
 $s \leftarrow \mathbf{Q}$ .pop_front();
if  $s$ .mode == painting then
  if  $\|s$ .paint_hit -  $\mathbf{r}\| \geq \Delta_{far}$  then
    SendFeedback("Move closer to the rock");
     $\mathbf{S}$ .clear();  $d_{seg} = 0$ ; return;
   $d_{seg} += \|s$ .paint_hit -  $\mathbf{S}$ .back().paint_hit||;
   $\mathbf{S}$ .push_back( $s$ );
  if  $d_{seg} \geq \Delta_{seg}$  then PlanTask( $\mathbf{S}$ , painting);
else if  $s$ .mode == viewing then
   $d_{seg} += \|s$ .nozzle_pose -  $\mathbf{S}$ .back().nozzle_pose||;
   $\mathbf{S}$ .push_back( $s$ );
  if  $d_{seg} \geq \Delta_{seg}$  then PlanTask( $\mathbf{S}$ , viewing);
else
  if  $\|s$ .cradle_pose -  $\mathbf{r}\| \geq \Delta_{seek}$  then
    PlanTask( $s$ , viewing);
if  $s$ .isFirstState() then PlanTask( $s$ ,  $s$ .mode);

```

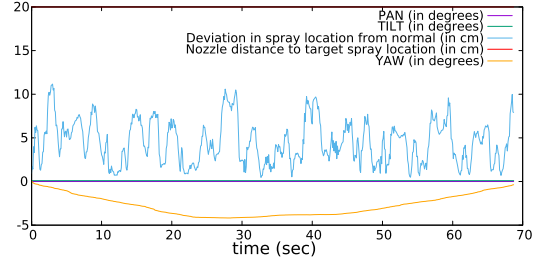
colors, with variable thickness of paint layers. Hence, in essence, the paint simulator provides a substitute for real visual feedback. Some features are absent from the current simulator, in particular, surface properties (absorption), paint runs and ability to vary ambient lighting, but are envisaged for future work.

Nozzle dynamics dictate the position and orientation of the nozzle at each time instant. In the live session the user's hand orientation in the VR environment, whereas, in offline session, the nozzle on the virtual UAV in the RotorS simulator is used as the nozzle orientation. While the visualization for the first mode is instantly available and helps the user in getting a quick preview of the painting commands, the visualization in second mode is useful for evaluating the end-result of a painting mission after the UAV executes the commands in the RotorS simulator, thus providing an intuition for what to expect if the mission is executed on the PaintCopter.

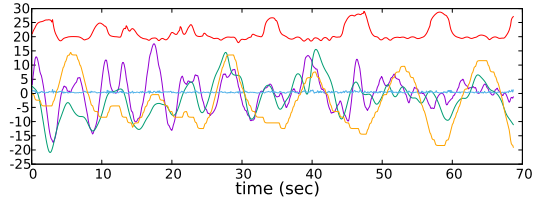
The paint flow from the nozzle has been modeled according to the findings of [17]. The atomized flow through the nozzle is found to be spraying out in a conical manner with the amount of paint distributed in a Gaussian manner. The parameters for this model, such as nozzle cone angle and the flow rate are found empirically from experiments. The orientation of the nozzle and its distance to the rock surface is used to keep track of the amount of paint deposited on each triangle on the mesh representation of the rock surface. A very well-established Kubelka-Munk (KM) theory [18] is then used to model the appearance of rock surface with layered painting involving multiple colors. Finally, an image rendering of the rock surface is transferred to the VR environment so that the user can visualize the painted rockwork and get a preview.

V. TASK PLANNING

Task Planning involves generating feasible waypoints for the UAV and the reference joint angles for PTU. In our previous work, we proposed a smoothing planner for generating smooth trajectories for the UAV. While the UAV executes this trajectory,



(a) Smoothing planner



(b) Optimal planner

Fig. 2. For a chosen scenario, PTU reference angles, distance of nozzle from the target spray location and deviation of nozzle's location from the surface normal, vs. time.

the PTU is controlled to spray at the reference points on the target surface. In such an approach, the PTU is used mainly to correct for the UAV's disturbances from the planned trajectory and hence the PTU is not used to its full potential. Moreover, the smoothing results in the nozzle spraying further away from the surface's normal direction and it has been shown in the previous work [1] that spraying away from the surface normal results in lower quality in the end result.

In this work, we propose an optimization based planner, that can produce better painting quality while respecting the spray nozzle constraints and platform dynamics. To illustrate the improvement over the smoothing planner, we choose an ideal scenario where the UAV is perfectly following the planned trajectory. For this chosen scenario, Fig. 2a shows reference PTU angles, UAV's heading and the nozzle's deviation from the surface normal direction over time. In the case of the smoothing planner, since the UAV is always perfectly aligned towards the target spray location, the reference joint angles for the PTU are zero. Moreover, the deviation of the nozzle from the surface normal direction is large (4.5 cm on average and up to 10 cm at times). Alternatively, as seen in Fig. 2b, in the case of the proposed planner, the PTU is utilized to a greater extent, which results in minimal deviation of the nozzle from the surface normal direction (0.4 cm on average and less than 2 cm at all times) and thus producing better quality painting. Furthermore, the reference UAV heading and PTU joint angles are smooth, respecting the physical constraints of the UAV and the PTU.

A. Optimal Planner

Notation: UAV states are defined as $\mathbf{s}_{uav} = [x, y, z, \psi, \theta_1, \theta_2]$, where $\mathbf{r} = [x, y, z]$ is the UAV's position, ψ is its yaw and θ_1, θ_2 are the pan and tilt values of the PTU. Nozzle states are defined as $\mathbf{s}_{nozzle} = [p_x, p_y, p_z, \psi_1, \psi_2, \psi]$, where $\mathbf{p} = [p_x, p_y, p_z]$ is the nozzle's position, \mathbf{h} is the nozzle's

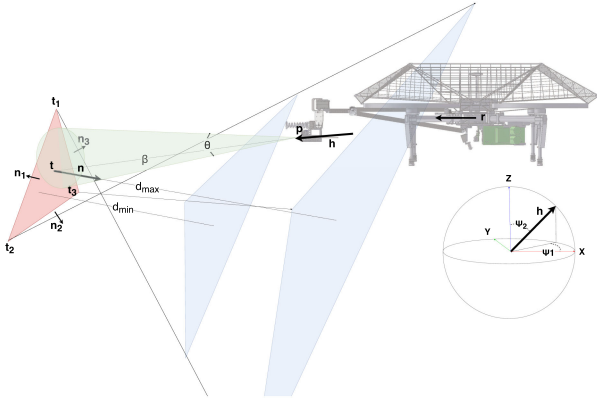


Fig. 3. Optimal planner notation.

heading with $\psi_1 \in [0, 2\pi]$ as its azimuthal angle and $\psi_2 \in [0, \pi]$ as its polar angle (see Fig. 3).

To get an optimal path for painting, we start by determining suitable states for the nozzle at each time instant. Each triangle (with vertices $\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3$, center \mathbf{t} and normal \mathbf{n}) in the mesh, that is affected by the painting action, defines an initialization for a nozzle state along with a set of constraints limiting the space within which the nozzle states are sampled (see Fig. 3). Then, a two-step optimization approach is used to sample better states in successive iterations. In the first stage of optimization, the states are defined as $\mathbf{s} = [p_x, p_y, p_z, \psi_1, \psi_2]$, and in this stage, new states are sampled while avoiding big changes in nozzle's position and orientation. Then a second stage evaluates the desired UAV orientation ψ by minimizing changes in the UAV's position and heading. Optimization is terminated when the maximum iteration limit is reached. Finally, UAV states are obtained from the nozzle states using inverse kinematics equations.

Initialization for each nozzle state is chosen as:

$$\mathbf{s}_{nozzle}^0 = [\mathbf{t} + d^* \mathbf{n}, \psi_1([-n_y, -n_x]), \cos^{-1}(-n_z), \tan^{-1}(-n_y, -n_x)] \quad (1)$$

where, d^* is the desired distance for a nozzle to be spraying from. $\psi_1(\cdot)$ is defined such that it is in the range $[0, 2\pi]$.

A set of constraints are imposed on the nozzle states to ensure that the nozzle is always spraying at the desired target location on the surface. Constraints on the nozzle position $\mathbf{p} = [p_x, p_y, p_z]$ ensure that the nozzle stays within a range $[d_{min}, d_{max}]$ from the target location \mathbf{t} and furthermore, the nozzle is spraying at an incidence angle no greater than θ_{inc} . These constraints are similar to ones proposed in [19], and are formulated as follows:

$$\begin{bmatrix} (\mathbf{p} - \mathbf{t})^T \mathbf{n} \\ -(\mathbf{p} - \mathbf{t})^T \mathbf{n} \\ (\mathbf{p} - \mathbf{t}_i)^T \mathbf{n}_i \end{bmatrix} \geq \begin{bmatrix} d_{min} \\ -d_{max} \\ 0 \end{bmatrix}, i \in \{1, 2, 3\} \quad (2)$$

where, \mathbf{n}_i are normals of separating hyperplanes that define the state space respecting the maximum incidence angle constraint, as shown in Fig. 3. Constraints on the nozzle heading $[\psi_1, \psi_2]$ ensure that the nozzle is always spraying towards the target location $\mathbf{t} = [t_x, t_y, t_z]$. Assuming a nozzle's output is typically

characterized by a cone with an aperture angle θ , these constraints can be formulated as follows:

$$\psi_1 \in \begin{cases} \left[\cos^{-1} \left(\frac{t_x - p_x}{\beta} \right) \pm \theta/2 \right], & \text{if } t_y \geq p_y \\ \left[2\pi - \cos^{-1} \left(\frac{t_x - p_x}{\beta} \right) \pm \theta/2 \right], & \text{otherwise} \end{cases} \quad (3)$$

$$\psi_2 \in \left[\cos^{-1} \left(\frac{t_z - p_z}{\beta} \right) \pm \theta/2 \right]$$

where, $\beta = \|\mathbf{t} - \mathbf{p}\|$.

The constraints can be further simplified as:

$$\cos(\psi_i) \in \left[\frac{t_i - p_i}{\beta} \cos(\theta/2) \pm \alpha_i \right] \quad (4)$$

where, $i \in \{1, 2\}$, $t_1 = t_x, t_2 = t_z, p_1 = p_x, p_2 = p_z$ and $\alpha_i = \sin(\cos^{-1}(\frac{t_i - p_i}{\beta})) \sin(\theta/2)$.

By using small angle assumption on changes in heading between successive iterations $\psi_i - \psi_i^{k-1}$, $\cos(\psi_i)$ can be written as $\cos(\psi_i^{k-1}) - \sin(\psi_i^{k-1})(\psi_i - \psi_i^{k-1})$. As a result, at each iteration k , the constraints in Eq. (4) can be linearized to give:

$$\begin{aligned} \frac{t_i \cos(\theta/2)}{\beta} - \gamma_i - \alpha_i &\leq \frac{\cos(\theta/2)}{\beta} p_i - \sin(\psi_i^{k-1}) \psi_i \\ &\leq \frac{t_i \cos(\theta/2)}{\beta} - \gamma_i + \alpha_i \end{aligned} \quad (5)$$

where, $i \in \{1, 2\}$, $\gamma_i = \cos(\psi_i^{k-1}) + \psi_i^{k-1} \sin(\psi_i^{k-1})$ and, α_i and β are evaluated using \mathbf{p}^{k-1} .

The optimization objective for evaluating the state \mathbf{s}^k at iteration k is to minimize the squared distance to preceding and succeeding states $\mathbf{s}_p^{k-1}, \mathbf{s}_s^{k-1}$ along with the estimate of the state from previous iteration, \mathbf{s}^{k-1} . As highlighted in [19], the intuition behind this is to shorten the tour by minimizing the jumps in position and orientation of the nozzle and also to limit the improvement from the previous iteration. This small improvement is essential to justify our small angle assumption made in obtaining Eq. (5). In addition to these, we also minimize the squared distance of the nozzle from the normal direction $\|\mathbf{n} \times (\mathbf{p} - \mathbf{t})\|^2$. This cost term pushes the nozzle to spray from a position closer to the normal direction of the target triangle.

The resulting convex optimization problem takes the shape of a Quadratic Program (QP) with linear constraints and can be formulated as follows:

$$\begin{aligned} \min_{\mathbf{s}} \quad & (\mathbf{s} - \mathbf{s}_p^{k-1})^T A (\mathbf{s} - \mathbf{s}_p^{k-1}) + (\mathbf{s} - \mathbf{s}_s^{k-1})^T A (\mathbf{s} - \mathbf{s}_s^{k-1}) \\ & + (\mathbf{s} - \mathbf{s}^{k-1})^T B (\mathbf{s} - \mathbf{s}^{k-1}) + \mathbf{q}^T C \mathbf{q} \\ \text{s.t.} \quad & \mathbf{q} = \begin{bmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{bmatrix} (\mathbf{p} - \mathbf{t}), \\ & \text{constraints (2) \& (5)} \end{aligned} \quad (6)$$

where, $A = \text{diag}([c_a, c_a, c_a, c_b, c_b]), B = \text{diag}([c_c, c_c, c_c, c_c, c_c]), C = \text{diag}([c_d, c_d, c_d])$ and $\text{diag}(\mathbf{u})$ is a diagonal matrix with elements in \mathbf{u} along the diagonal. c_a, c_b, c_c, c_d are tuning parameters. We use qpOASES [20] to solve the above QP.

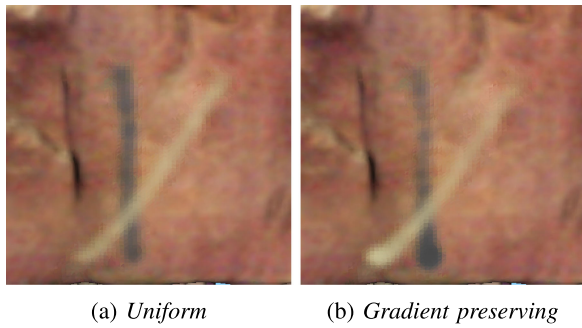


Fig. 4. Different time-allocation policies for path planning result in different appearances.

The second stage optimization solves for the heading of the UAV ψ by minimizing its difference to previous and successive state's heading ψ_p, ψ_s along with the UAV's squared distance to previous and successive position $\mathbf{r}_p, \mathbf{r}_s$. This is achieved by minimizing the cost function:

$$\begin{aligned} \min_{\psi} & (\psi - \psi_p^{k-1})^2/d_p + (\psi - \psi_s^{k-1})^2/d_s + c_e d_p^2 + c_e d_s^2 \\ \text{s.t. } & d_p = \|\mathbf{r} - \mathbf{r}_p^{k-1}\|^2, d_s = \|\mathbf{r} - \mathbf{r}_s^{k-1}\|^2, \\ & \text{isNotInCollision}(\mathbf{r}, \psi) \end{aligned} \quad (7)$$

where, $\text{isNotInCollision}(\mathbf{r}, \psi)$ checks if the UAV with position \mathbf{r} and heading ψ is in collision with the target surface. c_e is a tuning weight parameter. A brute-force search is used to solve for the UAV heading ψ by searching at fixed increments in the range $[0, 2\pi)$. Given the optimal states \mathbf{s} from the first stage optimization, the position of the UAV \mathbf{r} is evaluated for each ψ using inverse kinematics equations. In practice, more than one previous and successive states are used to evaluate the cost function described in Eq. (7). This is found to generate much smoother trajectories.

The output of the above optimization routine is a set of waypoints defined by s_{uav} . These waypoints are connected with minimum-snap polynomial trajectories (segments). These trajectory segments are of fixed time and are defined by independent polynomials for each of the flat-output variables $[x, y, z, \psi]$. The polynomials are evaluated using the unconstrained QP solution provided in [21]. We use two different time-allocation policies for evaluating segment times: *Uniform* and *Gradient preserving*. *Uniform*: The user defines a fixed painting speed for each individual segment, to get uniform paint density along the pattern. *Gradient preserving*: The segment times are evaluated as per the user's painting speed in the VR environment, thus preserving the gradients in the end result. Fig. 4 illustrates the end result when employing the two different time allocation policies for a single given dataset where the user paints multi-color strokes at an increasing speed to obtain a gradient. In the rare cases where the user paints at speeds not feasible for the UAV, the time per each segment is scaled up and the nozzle flow rate is scaled down by the same factor so that the desired gradient

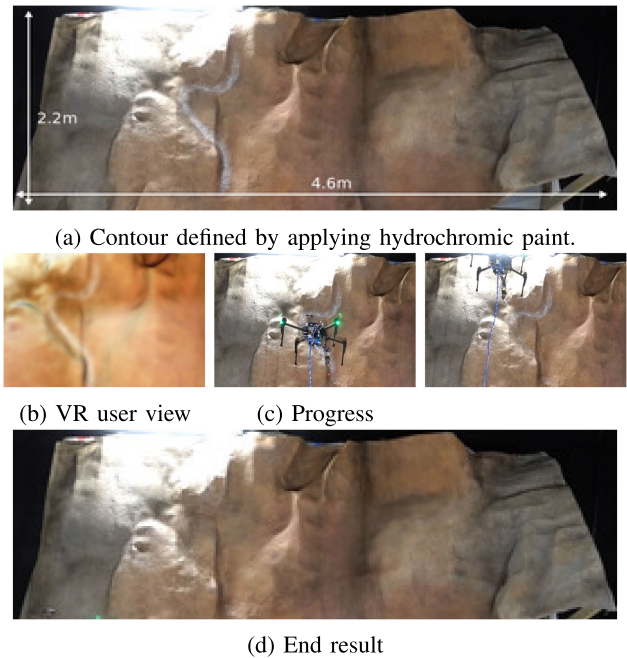


Fig. 5. Accuracy test: spraying water on a specific contour.

effect is obtained while keeping the trajectory feasible. However, this has only been tested in simulation since our nozzle doesn't support adaptive control of the flow-rate.

VI. EXPERIMENTAL RESULTS

In this section, experimental results are provided for paintings of various colors on a rock surface of approximately $4.6 \text{ m} \times 2.2 \text{ m}$ at a height of 2.6 m above ground (Fig. 5a). Experiments to evaluate the interface and validate the paint simulator are provided below. Finally, some qualitative results of a long painting activity are shown at various stages, both in simulation and in real-world.

A. Interface Accuracy

To test the accuracy of the VR interface, we simulated a maintenance task of spraying mildewcide to treat mold growth on a rock surface. The mold is simulated by applying hydrochromic paint along a contour on the rock surface, as seen in Fig. 5a. On contact with water, hydrochromic paint turns transparent, revealing the texture beneath. So, by spraying water on the contour, we simulated the scenario of spraying mildewcide. In such a setup, the accuracy of the system can be visually evaluated.

Within the virtual environment, the user is able to identify and spray along the chosen contour, as seen in Fig. 5b. A painting mission is then planned and executed by the UAV to spray water along the chosen contour, as seen in Fig. 5c. This experiment demonstrates that a user in the virtual environment can clearly identify visual features on the target surface and paint accurately along the desired contour. The end result (Fig. 5d) shows no trace

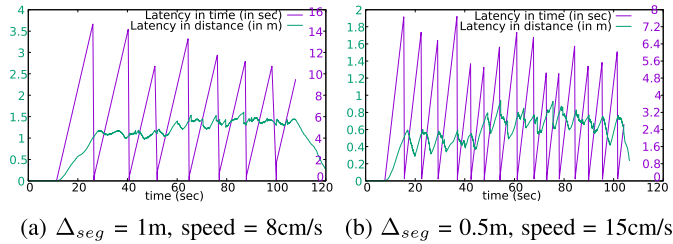


Fig. 6. Latency metrics.

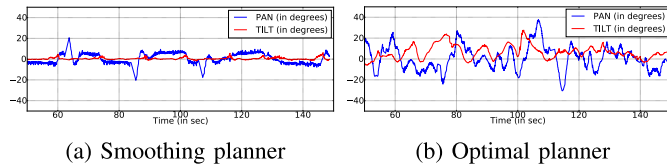


Fig. 7. Optimal planner uses the PTU to a greater extent.

of the hydrochromic paint, which emphasizes the accuracy of the painting.

B. Interface Latency

During a live VR session, the VR states are processed in segments, as mentioned in Section III-B. In such a scenario, the user observes latency between the paint-hit location of the virtual spray gun and the UAV's nozzle. For a chosen scenario, where the user was spraying a 7.75 m long area-fill pattern at an average speed of 8 cm/s, we show in Fig. 6 the latency metrics as a function of time. The latency in time (LT) is defined as the difference between the timestamps of the current incoming VR state and the first VR state of the segment currently being processed. This latency depends on the speed at which the user is painting and also the threshold Δ_{seg} . In addition to the above factors, the time needed for task planning plus the time needed to execute the segment results in the UAV falling behind the user's hand. The latency in distance (LD) is defined as the distance by which UAV's nozzle is lagging behind the virtual nozzle along the pattern.

In Fig. 6a, Δ_{seg} is chosen as 1 m, resulting in 8 segments in total. The average processing time for planning a task is 356 ms per segment and the UAV is commanded to paint at the speed of 8 cm/s. In this configuration, LT is between 10–15 sec and LD is between 1–1.5 m. Alternatively, in Fig. 6b, Δ_{seg} is chosen as 0.5 m, resulting in 16 segments in total. The average processing time for planning a task is 189 ms per segment and the UAV is commanded to paint at the speed of 15 cm/s. In this configuration, LT is between 5–8 sec and LD is between 0.3–0.9 m.

C. Task Planner Comparison

In Section V, with an example scenario, we showed that the optimal planner exploits the full potential of the PTU to generate better quality end result. Fig. 7 shows the PTU angles from a real experiment and it can be clearly noted that in the case of

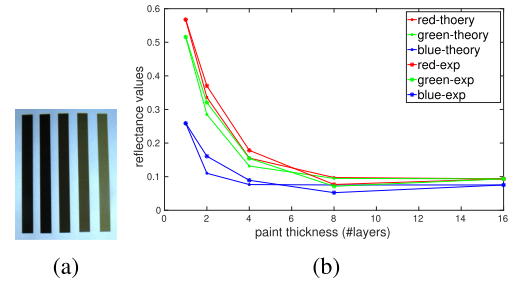


Fig. 8. Paint simulator validation experiment. (a) Non-fluorescent paper with layered paint of increasing thickness (right-to-left) (b) Measurements follow theoretical model.

the smoothing planner, the PTU is merely used for correcting for deviations from the planned path whereas, in the case of the optimal planner it's used to a greater extent.

D. Paint Simulator Validation

This section describes the experiments performed to validate the models and assumptions made in the design of the paint simulator (Section IV). For this, we use a laser printer to print layers of colors on a non-fluorescent sheet of paper. Non-fluorescence is required as we do not want the reflected light to affect the color measurements. Fig. 8a shows the printed specimen for a sample yellow color. From right to left, there are 1, 2, 4, 8 and 16 layers of the same color. The increase in the number of layers printed increases the thickness of the color and tends to saturate as the thickness tends to increase to infinity. A picture of the non-fluorescent paper is captured so that we can eliminate the uneven illumination mathematically in the calculation of reflectance. Fig. 8b shows theoretical plots as compared with experimental data, and we see that the trend of the reflectance values obtained from the KM model is similar to those obtained by measuring through a camera.

E. Qualitative Results

In this section, we provide some qualitative results of the user's painting activities involving area-fill and line drawings with multiple colors. The data from the VR environment is processed by the task planner to generate painting missions for each individual activity. Renderings of the surface as generated by the paint simulator and the images captured from the real-world are compared at various stages in Fig. 9, showing the close resemblance between the simulation and reality. But, despite using human operator's commands to execute a painting, there are several differences in the painting process and the end result achieved by the UAV in comparison to the human: (a) We found that the UAV is significantly faster than a human in painting large areas. This arises from the fact that the user has to navigate using the cradle for accessing areas beyond the hand's reach, whereas, the UAV can perform the painting in one continuous motion. (b) An expert human is very good at choosing the painting parameters that minimize undesired effects such as paint run. Our UAV at the moment doesn't model these while

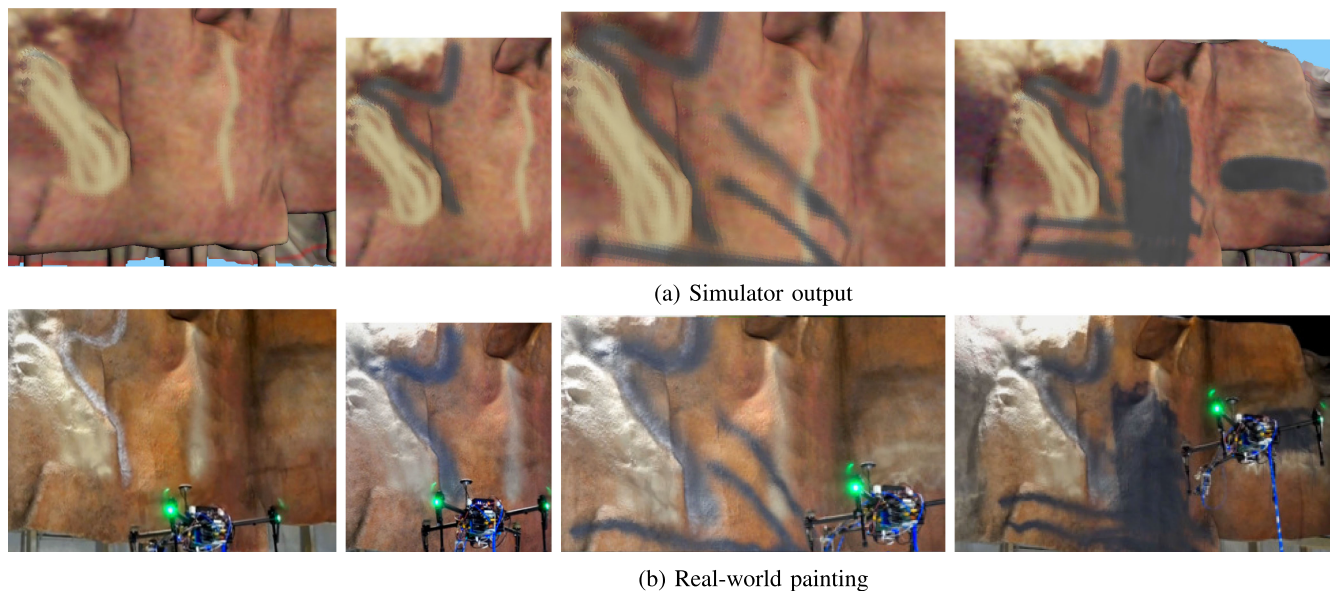


Fig. 9. Qualitative results comparing the multi-color painting activity, (a) as rendered by the Simulator, and (b) as captured in the real-world. The activity involves drawings made in the VR and projection of pre-defined patterns onto the surface.

planning its path and hence is incapable of tuning the necessary parameters in the flight.

VII. CONCLUSION

In this letter, we presented a virtual reality interface for a spray painting UAV. The proposed interface is immersive and allows even an inexperienced user to command the autonomous UAV. As a part of this work, we developed a paint simulator that allows for quick pre-visualization of the user's painting activity. Furthermore, we proposed an optimization based planner for generating higher quality paintings compared to our earlier work. Our experimental results show that the interface is accurate enough to have precise control over the painting activity and has low latencies. However, we occasionally notice undesired effects such as paint run. In the future, we envision being able to tune painting parameters in the flight so as to avoid such artifacts.

REFERENCES

- [1] A. S. Vempati *et al.*, "Paintcopter: An autonomous UAV for spray painting on three-dimensional surfaces," *IEEE Robot. Autom. Lett.*, vol. 3, no. 4, pp. 2862–2869, Oct. 2018.
- [2] M. El Helou, S. Mandt, A. Krause, and P. Beardsley, "Mobile robotic painting of texture," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2019, pp. 640–647.
- [3] J. J. Roldán *et al.*, "Multi-robot systems, virtual reality and Ros: Developing a new generation of operator interfaces," in *Proc. Robot Operating Syst.*, 2019, pp. 29–64.
- [4] J. Nagi, A. Giusti, G. A. Di Caro, and L. M. Gambardella, "Human control of UAVs using face pose estimates and hand gestures," in *Proc. ACM/IEEE Int. Conf. Human-Robot Interaction*, 2014, pp. 252–253.
- [5] S. MohaimenianPour and R. Vaughan, "Hands and faces, fast: Monocamera user detection robust enough to directly control a UAV in flight," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 5224–5231.
- [6] J. J. Roldán *et al.*, "Multi-robot interfaces and operator situational awareness: Study of the impact of immersion and prediction," *Sensors*, vol. 17, no. 8, 2017, Art. no. E1720.
- [7] N. Li, S. Cartwright, A. Shekhar Nittala, E. Sharlin, and M. Costa Sousa, "Flying frustum: A spatial interface for enhancing human-uav awareness," in *Proc. 3rd Int. Conf. Human-Agent Interaction*, 2015, pp. 27–31.
- [8] W. Hnig, C. Milanese, L. Scaria, T. Phan, M. Bolas, and N. Ayanian, "Mixed reality for robotics," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Sep. 2015, pp. 5382–5387.
- [9] W. Chen, Y. Chen, B. Li, W. Zhang, and K. Chen, "Design of redundant robot painting system for long non-regular duct," *Ind. Robot: An Int. J.*, vol. 43, no. 1, pp. 58–64, 2016.
- [10] E. Asadi, B. Li, and I.-M. Chen, "Pictobot," *IEEE Robot. Autom. Mag.*, vol. 25, no. 2, pp. 82–94, Jun. 2018.
- [11] T. Lindemeier, J. Metzner, L. Pollak, and O. Deussen, "Hardware-based non-photorealistic rendering using a painting robot," *Comput. Graphics Forum*, vol. 34, no. 2, pp. 311–323, 2015.
- [12] L. Scalera, S. Seriani, A. Gasparetto, and P. Gallina, "Watercolour robotic painting: A novel automatic system for artistic rendering," *J. Intell. Robot. Syst.*, pp. 1–16, Sep. 2018, doi: 10.1007/s10846-018-0937-y.
- [13] D. Song, T. Lee, and Y. J. Kim, "Artistic pen drawing on an arbitrary surface using an impedance-controlled robot," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 4085–4090.
- [14] B. Galea and P. G. Kry, "Tethered flight control of a small quadrotor robot for stippling," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 1713–1718.
- [15] A. S. Vempati, I. Gilitschenski, J. Nieto, P. Beardsley, and R. Siegwart, "Onboard real-time dense reconstruction of large-scale environments for UAV," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 3479–3486.
- [16] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, "RotorS—A modular Gazebo MAV simulator framework," in *Robot Operating System (ROS): The Complete Reference (Volume 1)*. Cham, Switzerland: Springer, 2016, pp. 595–625. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-26054-9_23
- [17] N. Berinpanathan, M. Kamel, R. Siegwart, and P. Beardsley, "Characterizing the spray coverage of nozzles," 2018.
- [18] P. Kubelka, "New contributions to the optics of intensely light-scattering materials. Part i," *Josa*, vol. 38, no. 5, pp. 448–457, 1948.
- [19] A. Bircher *et al.*, "Structural inspection path planning via iterative viewpoint resampling with application to aerial robotics," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2015, pp. 6423–6430.
- [20] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, "qpOases: A parametric active-set algorithm for quadratic programming," *Math. Program. Computation*, vol. 6, no. 4, pp. 327–363, 2014.
- [21] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Robotics Research*. New York, NY, USA: Springer, 2016, pp. 649–666.