Assisted Painting of 3D Structures Using Shared Control with Under-actuated Robots

Joshua Elsdon[1] and Yiannis Demiris[1]

Abstract*— We present a shared control method of painting 3D geometries, using a handheld robot that has a simple actuation scheme. The user scans the robot near to the desired painting location, the robot then uses its single movement axis to satisfy the required paint distribution. A simultaneous simulation of the spraying procedure is performed, giving an open loop approximation of the current state of the painting. An online prediction of the best path is calculated by producing a paint required density map in the 2D space formed from the current nozzle position on the gantry and the time into the future. A directed graph then extracts its edge weights from this density graph and Dijkstra's algorithm is then used to find the candidate for the most effective path. Due to the heavy parallelisation of this approach and the majority of the calculations taking place on a GPU we can run the prediction loop in 32.6ms for a prediction horizon of 1 second, this approach is computationally efficient, outperforming a greedy algorithm. The path chosen by the proposed method on average chooses a path in the top 15% of all paths as calculated by exhaustive testing. This approach enables development of real time path planning for assisted spray painting onto complicated 3D geometries. This method could be applied to applications such as assistive painting for people with disabilities, or accurate placement of liquid when large scale positioning of the head is too expensive.

I. Introduction

Painting a 3D model with an airbrush is a skilled discipline, fine control of nozzle speed, position and timing of the air valve are required. Here we outline an algorithm for path planning and automated airbrush hardware that alleviates some of this skill requirement by allowing the user to share control of the airbrush nozzle with a robotic system. This could prove to be useful in assistive painting, where the user has reduced control of their extremities, or reduced mental acuity [1]. Further it could also be useful in other circumstances where a liquid must be applied accurately, but positioning the head on a large scale is not feasible due to cost or space requirements, such as applying medicine to skin [2].

Our system plans movement of a spray nozzle along a one dimensional linear slide to maximise the amount of paint that can be correctly placed onto the 3D geometry. The system has access to the location and velocity of the robot and can control only the nozzle location on the gantry and whether the nozzle is dispensing paint. The robot is handheld and is manoeuvred by the user. A typical system setup is shown in Figure 1.

[1]Joshua Elsdon, je10@ic.ac.uk, and Yiannis Demiris, y.demiris@ic.ac.uk, are with the Department of Electrical and Electronic Engineering, Imperial College London,

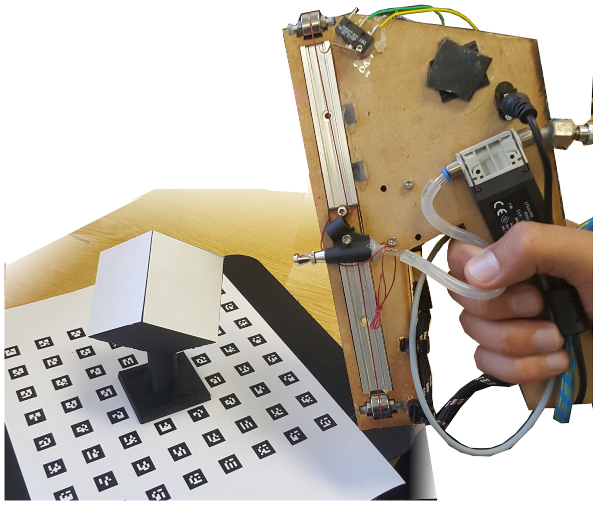


Fig. 1: The experimental setup. On the right is the robot, it has a single axis of movement which can move the airbrush head up and down. On the left is the object to be painted, it is located on top of an array of visual markers, which are observed by a camera mounted on the robot, see Figure 6 for a more detailed view of the hardware.

II. Background

Two research areas provide fruitful background reading for this problem: automated industrial painting, such as painting car bodies [3][4][5][6][7] and assistive painting tools in the artistic domain [8][9]. Car manufacturers, in order to keep costs low, use robots to paint car panels. These robots will usually follow a strict hard coded path, determined by a mixture of automated software tools and expert adjustments. The robots the car industry use are large and expensive, and the fixed path planning makes them most effective when reproducing the same work over and over again. Artistic pursuits however are likely to require a different outcome on each iteration, further, humans being in the creative loop is often a positive [1], unlike industrial painting processes [10][11]. Industrial painting usually is simulated in an offline manner[3][4], though for artistic pursuits real time visualisation and interaction is important[8].

Hegels *et al.* [3] presented a method for modifying existing robotic arm trajectories for painting car panels. The method maximised the evenness of the paint, whilst keeping the accelerations acceptable for the large robotic arms. Their approach is entirely offline, and at run time the coating is performed open loop, with no checking of the final coat evenness. They specify a method of capturing the real coating distribution by spraying for a short time onto a plate, which is then sampled across its area. They use the sampled distribution in evaluating the cost function of the current iteration of the trajectory, but in order to use optimisation methods that utilise gradients, they fitted this sampled distribution to a distribution that could be described analytically. This simplified distribution is used to calculate the next direction to search in the perimeter space. Their choice to use a simplified model for path planning followed by an expensive method for tracking the cost is one that we have emulated to some extent in this work.

In contrast to the industrial painting methods, Prevost *et al.* [8] use a shared control approach, where the robot is assisting the human towards the joint goal of painting a mural. In their design the robot's position is found via external cameras that locate QR codes mounted on the top and bottom of a standard spray paint can. When the robot is judged to be in a good position to add paint to the canvas, a radio controlled servo is actuated to press the valve of the spray paint. The user just has to meander the robot across the canvas, and the paint will be applied such that the state of the painting moves towards that of the target design. The system provides the user with a graphical representation of areas of the painting that need more paint, and a total possible added value using the current colour of paint. Prevost *et al.* [8] did a similar sampling followed by simplification of the paint distribution as Hagels *et al* [3]. The work we present here has a number of similarities, though we extend the robot to have some movement control in the form of a single axis moving the head, and the algorithm presented efficiently finds an actuation plan for this axis to maximise painting efficiency.

Arikan *et al.* [4] present a thorough overview of the calculations necessary to track the thickness of paint given the path, geometry to be painted and distribution pattern of the spray nozzle. They implemented a method to produce a path that produced an even coating over the surface of a car body panel, this was conducted offline and the robot simply follows the instructions. Their planning algorithm is unlikely to translate well to a situation where the velocity of the head is not known until run time as it relies on the path having a set offset from the adjacent pass. This is a common theme for car body painting, as this is the best method for getting an even coat, which is usually the target in car body painting, this is confirmed by Chen *et al.* [6] in their review of path planning for spray painting.

Konieczny *et al.* [9] give an account of their work regarding simulating airbrushes for graphics creation on a computer, this is completed with the design of an electronic airbrush for the user to interact with. Their system tracks the electronic airbrush using a magnetic tracker, and has a dual action trigger (allows control of paint flow and air flow). They also present a mature algorithm for simulating the the paint droplets, including the methods used to blend colours for a realistic looking finish. They implemented all of the computationally expensive operations on a GPU. Our work uses a similar techniques for GPU acceleration of the paint simulation.

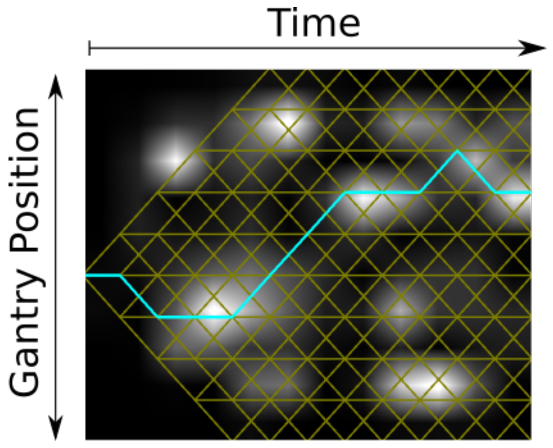


Fig. 2: The path that the nozzle takes is based on a grid of positions with line segments connecting them. The path is chosen such that it maximises the amount of paint that can be dispensed into the target areas. The white sections of this image are in need of paint. The cyan line is the chosen path, yellow represents all possible paths.

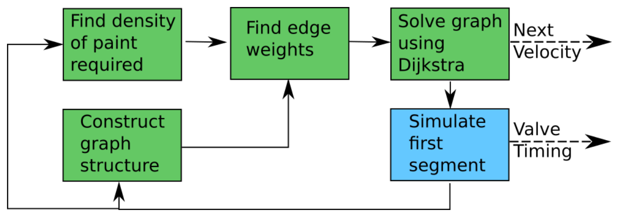
Shared control is an area of research that concerns a wide range of applications other than assistive painting [12][13], Carlson *et al.* [14] presented work outlining their experiments regarding having a robotic motorised wheelchair share control with a user. They used a secondary task, playing a simple game on a monitor, to impair the users to simulate reduced attention or reduced mental acuity. Using a shared control frame work they were able to reduce the number of collisions from 78% of participants to effectively none with the system active, one collision occurred due to a mechanical failure. This experiment shows effectively that shared control can enable users to do activities which would otherwise be too difficult or unsafe to complete.

III. Method

This section will outline the method of generating an actuation plan for the robot. The outputs from the algorithm are nozzle velocities and times to turn the airbrush on and off. A simplified view of the problem is presented in Figure 2, a flow chart of the method is presented in Figure 3.

There are two main stages to the software, candidate path selection and path simulation. The function of the candidate path selection is to suggest a path for the future movement that is likely to provide rich opportunities for the nozzle to dispense paint. The path simulation is then used to both calculate when the nozzle should dispense paint within this path and to update the internal representation of the state of the painting.

The candidate path selection starts by sampling the whole movement space for required paint quantity, generating a 2D representation of required paint density with axis being the position of the spray nozzle on the gantry and time from the beginning of the current movement, this is described in detail in Section IV-A. Using this density map we can then estimate the maximum benefit that any movement can make.



Find density of paint required
Find edge weights
Solve graph using Dijkstra
Next Velocity
Construct graph structure
Simulate first segment
Valve Timing

Fig. 3: Green sections are part of the candidate path selection. The blue process is the path simulation described in Section V. The dashed arrows on the right indicate outputs to the hardware described in Section VI-B.

We assume the nozzle can only dispense at one rate, hence for areas that need sparse covering with paint the nozzle should have a high speed in order not to over paint the region. For areas that require a high density of paint the nozzle will be allowed to move more slowly. A tree structured directed acyclic graph is formed where the edge cost is found from the required paint density, this is described in Section IV-C and Section IV-B. Dijkstra's algorithm [15] is then run to find the route that the nozzle should take to maximise its ability to dispense paint according to the target paint distribution.

The path simulation stage then performs a simulation of the path suggested by the candidate path selection. At each time step it tests whether the nozzle should be dispensing paint, if so it updates the representation of the paint distribution accordingly. Only the first section of the path that is actually going to be implemented on the hardware needs to be addressed by this section of the software. The fact that the candidate path includes a larger time horizon than is actually implemented allows this algorithm to find a solution that is closer to the globally optimum path whilst still constantly correcting for adjustments due to unexpected movements of the robot, and measurement drift. The path simulation is described in detail in Section V.

IV. Candidate Path Selection

Simulating paths fully as demonstrated in Section V is an expensive operation; it cannot be parallelised across time, as future paint deposits will rely on past paint deposits. Therefore using such a simulation to evaluate many paths, then picking the best one is not a tenable solution if the allowed solution space is large. We must make some assumptions to accelerate the search for good candidate paths. Firstly we assume that the operator is moving the robot primarily perpendicularly to the gantry direction and the spray direction, as to sweep the largest possible area. This helps minimise the situation where the head could revisit the same physical location at a later time, thus reducing the dependence of later paint deposits on previous paint deposits. We also assume that the user is trying to keep the robot's velocity smooth at all times.

This software module has 4 stages to its implementation: 1) Calculating the required paint density across the space defined by the spray nozzle's location on the gantry and time. 2) Producing a graph structure that can account for mechanical feasibility. 3) Calculating the maximum benefit of line segments in this graph. 4) Solving for the path most likely to place the maximum amount of paint in the correct place using Dijkstra's algorithm [15]. These stages are highlighted in green in Figure 3.

A. Calculating Required Paint Density

The spray painting process has a number of variables that will affect the rate at which the paint covers a particular area, for example, if the nozzle is at a large distance from the object being painted the paint per second arriving at a given area will be less than if the nozzle were close. Other considerations include the distribution pattern of the spray nozzle, obliqueness to the surface etc. We wish to account for all of these effects and simplify their contribution to one variable: at this location on the gantry and this point in time, how much paint is needed.

We sample this quantity at regular intervals in a grid across time and nozzle position in the manner defined in Equation 1. At each location 1024 rays are cast into the scene and each ray returns the difference between the current paint coverage, c, and the target coverage, p. This is divided by the distance squared, d^2. This operation gives use the required flux of paint at unit distance for this ray. The direction of the rays is decided by the distribution pattern of the airbrush nozzle. This averaged value is now representative of the average paint flux at unit distance at this location on the gantry, g, at a given time, t. Repeating this operation in a grid across many gantry positions and time instants produces a map of required paint density in all positions that the nozzle will have access to over the allotted time horizon. This can be visualised as in Figure 2, where the white sections are requiring a large paint density, and the black areas require little or no paint density. This section does not make any strong assumptions about the geometry of the object to be painted, as long as the model is a good approximation of the real object to be painted and is triangulated reasonably, avoiding very thin triangles and avoiding unwanted gaps between triangles. We use this map of required paint density as the basis for generating the edge weights of a directed graph in Section IV-C.

$$d(g,t) = \frac{1}{\text{rays}} \sum_{i=0}^{i=\text{rays}} \frac{c_i - p_i}{d^2} \qquad (1)$$

B. Building Graph Structure

There are an infinite number of paths that the nozzle could take between the start and the time horizon, so it is necessary to discretise the movements. We define this discretisation by allowing the head to only be in discrete locations along the gantry at given intervals of time. This forms a grid on the space defined in Section IV-A. The nozzle can then move between these locations in line segments. At each grid location there is a predefined number of alternative routes going forward, which we call the number of *divisions*. In order to account for the fact that the nozzle should not be allowed an infeasible change in speed the graph structure needs to account for the incoming speed to each grid location and only allow the nozzle to leave the grid location at an

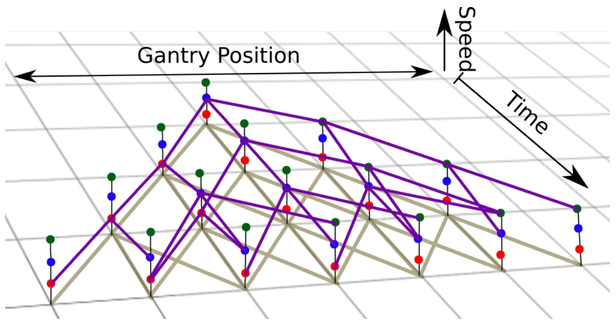


Fig. 4: A visualisation of the graph structure. The stacked nodes encode the speed incoming to the node, blue is 0, red is $+1$, green is -1. The purple lines indicate the edges of the graph, notice they will not allow a transition between $+1$ and -1 speed or the reverse. In a realistically proportioned graph there is a parameter *speed change* which defines the allowable change in speed at each node. The shadowed lines at the bottom of the image indicate the edges of the basic tree structure. The purple edges will inherit their edge costs from this base graph, notice there is often more than one edge representing the same physical movement.

allowable velocity. This is encoded by having multiple nodes in the graph at each grid location, one for each possible incoming speed. These nodes are not connected to nodes that would be infeasible given the incoming speed it represents. For example, if the nozzle was moving at full speed along the gantry in one direction it may be infeasible for it to move full speed in the other direction during the next time period.

Many of the paths from one node to the next will represent the same physical movement. Hence we calculate the cost for all possible edges between nodes, not accounting for feasibility, the edges on the graph then inherit the costs from this simplified graph. This encoding of feasibility and the inheritance of edge costs is described fully in Figure 4.

C. Calculating Benefit of a Sub-movement

We now need to define an operation that will return a score for a line segment taken from one edge in the graph described in Section IV-B. Given the robot's velocity perpendicular to the gantry, v,and the gradient of the line segment, $\frac{\mathrm{d}}{\mathrm{d}t}(g)$, we can calculate the nozzle velocity, n.

$$n = \sqrt{\frac{\mathrm{d}}{\mathrm{d}t}(g)^2 + v^2} \tag{2}$$

Combining this with the flow rate of the nozzle,flow, and the distribution pattern of the nozzle (simplified to a cross section area at unit distance, A) we can calculate the paint flux at unit distance, f, for this line segment.

$$f = \frac{\text{flow}}{A \cdot n} \tag{3}$$

Comparing this paint flux with a number of samples along this line segment we can evaluate whether this is an appropriate flux value for this location. If the flux is less than that required then the score, w_i of this sample on the line can be positive. If it is more than is required the score is zero, this is because if the nozzle were to perform this movement it would overdose this region with paint. The total score of the line segment, W, is the summation of all the samples along its length. Typically we use 32 samples along each line segment.

$$w_i = \begin{cases} d(g,t) & \text{if } f < d(g,t) \\ 0 & \text{if } f > d(g,t) \end{cases} \tag{4}$$

$$W = \frac{1}{\text{samples}} \sum_{i=0}^{\text{samples}} w_i \tag{5}$$

The positive score given to a sample effects how the system will behave, the optimum score to assign is the calculated flux for this line segment. This will maximise the amount of time that the nozzle can be switched on over the whole path. Alternatively, using the paint required at this position as a score prioritises the robot to visit areas that are more in need of paint. We found that using the target paint quantity as the score achieves better results. The paths chosen have more margin for error compared to picking paths where the calculated flux for the path is very close to the approximated required flux. When there is a small margin for error, the discrepancy between the required paint density approximation and the full simulation described in Section V can lead to the nozzle being left off to avoid an overdose condition. In this case there will be a large discrepancy between the score expected and the actual calculated score for a line segment, meaning that the path found using this method will be far from optimal.

We use this method in Section IV-B to give a weight to each element of the graph representing all possible movements.

D. Solving for Best Path

We now have a graph defined in Section IV-B has edge weights as calculated by the function described in Section IV-C. Using Dijkstra's algorithm we find the path from the start node to the time horizon that maximises score. This should give the route that maximises the opportunity for the nozzle to apply paint to the correct locations. The path selected should be such that it chooses swift diagonal movements through areas that require a sparse covering in paint, for areas that require high densities it will chose a path that maximises the amount of time that the nozzle is within the region, as to maximise the amount of paint distributed over that time period. Solving by Dijkstra's algorithm was implemented using the Boost Graph Library, and this is the only computationally significant part of the algorithm that takes place on the CPU.

V. Simulation of the Selected Path

All of the above calculations rely on the assumption that paint dispensed earlier in the movement will not affect paint dispensed later. For maintaining an accurate internal representation of the state of the paint coverage this is not acceptable. Therefore we must run a simulation that can take account of previous paint coverage affecting the value of applying paint to the same area in the future. We only need to run this section of the algorithm on the first section of the path, therefore this high computational cost is acceptable.

This simulation is achieved by modelling the paint droplets as rays and casting them into the geometry, these rays are used to lookup the corresponding pixel in the texture map, which holds the current state of paint coverage on the object, a similar texture holds the target paint coverage. At each time step r rays are cast, each represents the appropriate amount of paint based on the time step, flow rate and rays per time step. It is often the case that multiple rays within each casting operation will point to the same pixel in the paint state texture map, therefore it is important to use atomic operations to ensure they are totalled correctly on the GPU. After the casting operation we have a list of pixels in the texture map that have received paint, the quantity of which is defined as q. q is compared to the amount of paint required at this location to reach the target. If the amount of paint cast to this location (q) is less than or equal to the target amount required, p, the total amount of paint is recorded as the score, s. If there is an overdose condition the excess is punished by a punishment factor, P.

$$s_i(t) = \begin{cases} q_i & \text{if } q_i < p_i \\ p_i - (q_i - p_i)P & \text{if } q_i > p_i \end{cases} \tag{6}$$

$$S = \sum_{i=0}^{\mathrm{r}-1} s_i \tag{7}$$

At each time step along the path we calculate an aggregate of the score of all the rays cast, S, if this total is greater than zero we can consider the paint distribution at this time instant a success. In this case all of the paint quantities will be added to their respective pixel in the texture map. If the *score* is negative, spraying at this location is detrimental to the painting of the object, and the paint quantities are not written back to the texture. The series of whether each time step was beneficial to the painting task is kept, this is used to produce timings for the valve that controls the paint flow on the robot.

VI. Validation and Comparison

To demonstrate the efficacy of the proposed method we will present two sets of experiments. The results that were gathered from simulation are presented in Section VI-A, those gathered from hardware tests are presented in Section VI-B.

A. Experiments in Simulation

We can use exhaustive evaluation of all possible paths in a particular scenario to give a robust bench mark for the quality of a selected path. The path that is generated can be given a rank out of possible paths, with 1 being the best rank. In order to compare the proposed method with a baseline method a greedy path planner was developed. The greedy algorithm picks the most valuable linear movement over the next time period until it reaches the time horizon. The evaluation of the line segments will be done using the simulation outlined in Section V.

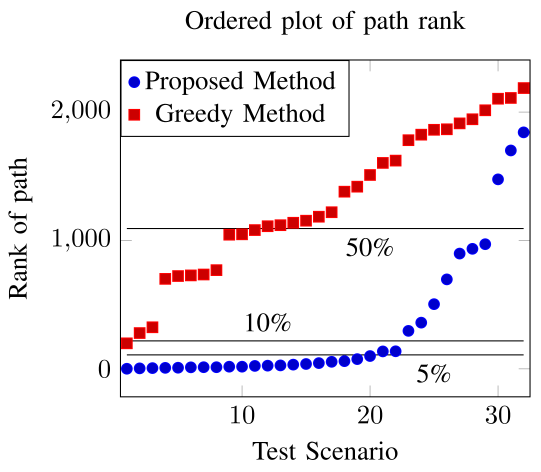


Fig. 5: For each of the 32 trials, the path produced by each algorithm was given a rank for quantity of ink placed out of all possible paths (calculated exhaustively). These ranks were then ordered from best to worst for clarity. We can see that the proposed method picks a good solution for the majority of trials, most in the top 5% (the average being 15%), though the greedy method does not seem to outperform a random pick as it returns paths with an average rank of 53%.

Firstly, both the proposed method and the greedy method are presented with the same 32 scenarios. Each method then generates a path for the airbrush nozzle to take, these are then given a rank against all possible paths. Both methods select paths from the line segments in a tree structure as shown in Figure 2 such that both algorithms obey the same feasibility constraints. The results can be seen in Figure 5. Our method averages a rank of 328 out of 2186 (15%), though the majority of the paths are in the top 5% of all paths. The outliers seem to be in scenarios where there is very little paint to be placed within the time horizon. The path chosen then maximises its time in areas of sparse paint requirement, though due to the approximate nature of the paint density estimate produced in Section IV-A this marginally fails to pass the test of whether the valve should be on presented in Section V.

The greedy method on average produced a path of 1302 out of 2186, which is worse than chance. This is because the greedy method has no capability to bias the path towards fruitful areas that occur outside of the nest time period.

Computational efficiency is important because we wish to use them on a real time spraying robot. The path must be planned whilst the first segment of the previous plan is being acted upon by the robot. Therefore if the run time of the algorithm is shorter we can increase the frequency of commands sent to the robot or we can increase the number of path options considered.

We compared the execution time of the proposed method against the greedy implementation. The times for the proposed method include the path generation and one segment of simulation as described in Section V. For the greedy algorithm the simulation is implicit in the path generation, therefore for this method only the path generation time is included.

TABLE I: Comparison between the proposed method and a simple greedy algorithm. Data taken over 32 test scenarios. The tests all were using a horizon of 7, with a choice of 3 directions at each intersection.

Method	Mean Time(ms)	Standard Deviation
Proposed Method	32.6	3.27
Greedy Method	116.84	3.20

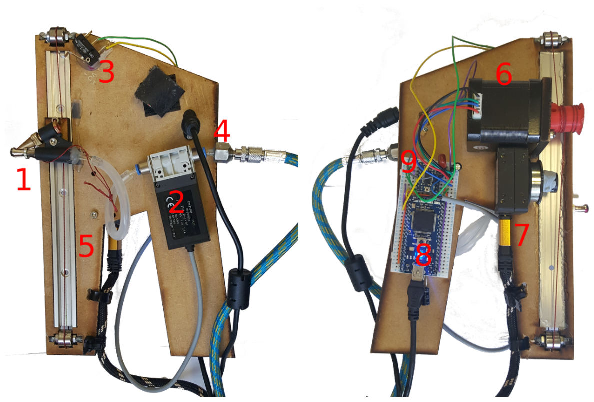


Fig. 6: The robot used for the hardware experiment. 1: Airbrush nozzle. 2: Festo 1ms air valve. 3: Limit switch. 4: 15PSI air input hose. 5: Igus DryLin low profile slide rail. 6: Nema 17 stepper motor. 7: Point Grey Chameleon 3 USB 3 camera. 8: MBED LPC1768. 9: Pololu stepper motor driver.

As shown in Table I the proposed method has a significantly shorter run time. In this set of scenarios the path generation was over a horizon of 7 decision intervals each representing 0.1s and there were 3 divisions after each time period. The proposed method would scale very well into larger solution spaces defined by: number of time periods before the time horizon; quantisations in the gantry direction and the number divisions after each time period. This is because the most computationally expensive part of the method is generating the density map, which is independent of the number of paths at each intersection and scales linearly with gantry quanta and horizon length. In contrast any algorithm that uses the full evaluation for line segments iteratively, such as the greedy algorithm, will scale very badly into bigger solution spaces.

B. *Experiment with Hardware*

To demonstrate the algorithm outlined above we have conducted a preliminary test case where the robot shown in Figure 6 should place paint in a specified area. The robot body is manoeuvred by the user. The algorithm plans the next move at a rate of 10Hz, whilst considering the next 1 second of potential paths available to it. The planning uses a snapshot of the robots measured velocity and orientation for extrapolation into the future. The robot measures its current location and velocity using a camera mounted to the robot, this tracks small markers that are located below the item to be painted, seen in Figure 1.

The camera output is fed back to a desktop computer with a mid-range GPU (NVidea 960) at 100Hz. The robot is also supplied with a compressed air hose for the airbrush and has a USB connected micro-controller to manage the movement of the airbrush head. The software architecture is using ROS (Robotic Operating System)[16] for communication to the robot and between nodes on the desktop. The path calculation is done primarily using OpenCL using the methods described above.

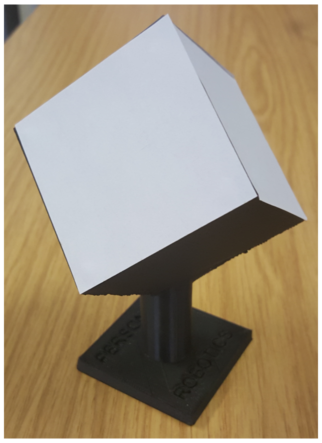
The aim for this demonstration was to paint three small circles on the faces of the 3D printed object shown in Figure 7b. The faces were covered in pieces of paper for reusability. Scans of the paper are presented in Figure 7c, the green overlays represent the ideal target locations. Figure 7d shows the state of the simulation after the demonstration, white sections are where paint is placed. Ideally the real paining should match that of the simulation.

It can be see from these images that the paint was indeed placed in the predominantly in the correct location, though there are some anomalies that should be explained. The small amount of misalignment between the target locations and where the paint was deposited is due to a combination of an offset in the optical positioning and perhaps a lack of precision in the registration of the object to the markers on the base. The offset is around 6mm in the worst case. Additionally to the offset there is some spurious paint placement, this is due to the loss of vision of the markers as the camera becomes very close to the object. To solve this issue the camera should be placed such that it does not lose vision on the markers when close to the target object.

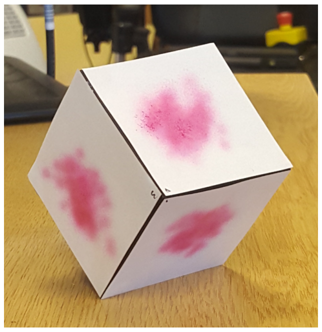
VII. Conclusion

In this paper we outlined a method for generating a trajectory for an airbrush nozzle on an under-actuated handheld robot. This method had four stages: sampling the swept space for required paint density; building a directed graph in this space, sampling from the required paint density to generate the edge weights; solving for the path that can maximised the quantity of paint delivered using the Dijkstra method and finally running a dense simulation of the first segment of the path to discover when to activate the air brush. We demonstrated that the proposed method has sufficiently low run time to run in a real time system by performing a preliminary study on real hardware and by measuring the run time directly. The run time was significantly shorter than a basic greedy algorithm that acted as a base line for algorithms that rely on the full simulation of a line segments to generate a path. For a typical scenario the proposed method took 32.6ms to produce a decision for the next time step, whilst considering a time horizon of 700ms. We showed that the path that was chosen by the proposed method was of high quality when compared to all possible paths, on average returning a path that fell in the top 15% of all paths. The majority of paths were in the top 5% of all paths. This is compared favourably to the greedy algorithm that performed no better than chance.

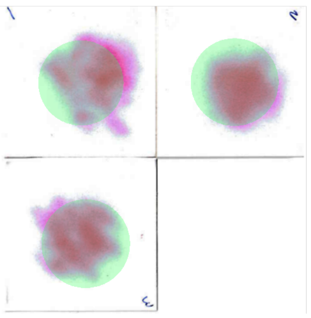
This work should be of use for researchers who are looking for methods to plan painting trajectories when system velocities are not known until run time. Applications could



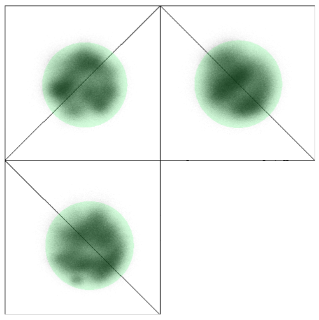
(a) The object to be painted.



(b) The Object after painting.



(c) A scan of the sides of the model that were painted



(d) The equivalent view from Figure 7c taken from the simulation.

Fig. 7: The object is a 60mm wide cube at an angle. The target is three 34mm circles on each of the top faces, as shown in Figure 7c, the green overlay indicates the target areas. Qualitatively it can be seen that the paint is broadly in the correct place, the displacement of each circle is within 6mm of the correct location, this drift is most likely due to registration of the 3D Object and positioning accuracy, anomalies far outside the circles seems to be temporary glitches in the position tracking. Future work will tackle these issues.

include assistive painting technologies for the impaired or for other painting systems where large scale precise movement of the painting head is not feasible.

Future improvements to this work include increasing the precision of the positioning system for the robot and producing an online method of adjusting the registration of the object to be painted. Further we will remove the high acceleration at the corners by adding rounded corners to the path. This would both increase the maximum speed and change of speed at each intersection.

References

[1] G. Kronreif et al. "PlayROB - Robot-Assisted Playing for Children with Severe Physical Handicaps". In: *9th International Conference on Rehabilitation Robotics*. 2005, pp. 193–196.

[2] Yuichi Tsumaki et al. "Development of a skincare robot". In: *2008 IEEE International Conference on Robotics and Automation*. 2008, pp. 2963–2968.

[3] Daniel Hegels, Thomas Wiederkehr, and Heinrich Müller. "Simulation based iterative post-optimization of paths of robot guided thermal spraying". In: *Robotics and Computer-Integrated Manufacturing* 35 (2015), pp. 1–15.

[4] M. A. Sahir Arikan and Tuna Balkan. "Process modeling, simulation, and paint thickness measurement for robotic spray painting". In: *Journal of Robotic Systems* 17.9 (2000), pp. 479–494.

[5] Q. Ye. "Using dynamic mesh models to simulate electrostatic spray-painting". In: *High performance computing in science and engineering '05*. 2006, pp. 173 –183.

[6] Heping Chen, Thomas Fuhlbrigge, and Xiongzi Li. "A review of CAD based robot path planning for spray painting". en. In: *Industrial Robot: An International Journal* 36.1 (2009), pp. 45–50.

[7] Jonathan Konieczny et al. "Automotive Spray Paint Simulation". In: *Lecture Notes in Computer Science*. 2008. Chap. Part 1, pp. 998–1007.

[8] Romain Prévost et al. "Large-Scale Spray Painting of Photographs by Interactive Optimization". In: *Computers & Graphics* 55 (2016), pp. 108–117.

[9] Jonathan Konieczny and Gary Meyer. "Airbrush simulation for artwork and computer modeling". In: *Proceedings of the 7th International Symposium on Non-Photorealistic Animation and Rendering - NPAR '09*. 2009, pp. 61–69.

[10] H.McIlvaine Parsons. "Human factors in industrial robot safety". In: *Journal of Occupational Accidents* 8.1-2 (1986), pp. 25–47.

[11] G.M. Bone. "Multisensor System for Safer Human-Robot Interaction". In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. 2005, pp. 1767–1772.

[12] Ayse Kucukyilmaz and Yiannis Demiris. "One-shot assistance estimation from expert demonstrations for a shared control wheelchair system". In: *2015 24th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*. 2015, pp. 438–443.

[13] Alexander Broad et al. "Trust Adaptation Leads to Lower Control Effort in Shared Control of Crane Automation". In: *IEEE Robotics and Automation Letters* 2.1 (2017), pp. 239–246.

[14] T. Carlson and Y. Demiris. "Collaborative Control for a Robotic Wheelchair: Evaluation of Performance,

Attention, and Workload". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 42.3 (2012), pp. 876–888.

[15] E. W. Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische Mathematik* 1.1 (1959), pp. 269–271.

[16] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*. 2009.