Image Space Path Following Control Using Visual Servoing

Cole Dewis
Department of Computer Science
University of Alberta
Edmonton, Canada
dewis@ualberta.ca

Martin Jagersand
Department of Computer Science
University of Alberta
Edmonton, Canada
mj7@ualberta.ca

***Abstract*—Visual servoing has been well explored in the literature for task specification and planning in image space. Planning tasks and paths in image space can be especially useful in unstructured environments, as a 3D reconstruction is not needed. However, few works have discussed following arbitrary image paths with visual servoing for robotic arms. This paper presents a path following controller for robotic arms based on image based visual servoing that can follow arbitrary paths in image space. The controller uses visual error to generate velocities that smoothly approach the path along the tangent. Additionally, the controller can optionally follow the orientation of the path, and can be applied to both eye-in-hand and eye-to-hand setups. Experiments are conducted on a Kinova Gen3 7DOF arm to evaluate the controller. Benefits of the path following controller over a trajectory-tracking approach are shown. Specifically, our path following controller displays smooth responses to physical disturbances and forced pauses.**

***Index Terms*—Visual path following, Visual Servoing, Path following control.**

I. Introduction

In many robotic applications, a robot must act with limited or no prior information about its environment. For example, when sanding down surface imperfections during manufacturing, the robot must be able to handle a variety of imperfections, where the position may not be known ahead of time. In such cases, vision based control systems can be used to guide robot motion using geometric information received from cameras. One method of specifying motion visually is through visual servoing [1], which maps motion in the image space to motion in the robot's frame of reference. Visual servoing can be run without calibration [2], making it flexible and easy to apply, while other methods such as reinforcement learning require more data or a training process [3] . Visual servoing is typically divided into position based visual servoing (PBVS) which minimizes an error between the 3D positions of the target location and end effector, and image based visual servoing (IBVS), which minimizes an error vector between the visual features of the end effector and target visual features.

To complete tasks, we often need to specify a path that the robot should follow, either to move through free space or to apply a tool in a certain way. This is necessary as with only a target position, we may run into obstacles or undesirable joint configurations [4]. Often, this is done through planning a path in 3D space, such as in [5]. However, to do this accurately, we need an understanding of the 3D structure of the environment, which can be expensive or impossible to obtain [6]. Planning a path in image space is thus appealing as we can do so without any 3D information from the environment. It also allows for paths to be visually extracted from the environment, which can be coupled naturally with computer vision methods such as edge detection for automated task planning [7]. The ability to follow arbitrary paths is important as it allows for more flexible task specification, which is especially important when working in image space.

Once a path has been specified, control methods can be divided into path following controllers (PFC), which set the current target as the closest state on the path to the current state, and trajectory tracking controllers (TTC), which follow a series of timed reference positions [8]. A TTC is the more common selection, and its use has been explored many times in relation to visual servoing, especially in mobile robots [9], [10], [11], [12], [13], [14]. However, these TTC-based methods have drawbacks, as they have undesirable behavior responding to disturbances and may act poorly when catching up to a timed reference [15]. Using a timed reference also requires that the trajectory speed is specified through the references, which reduces flexibility. In comparison, a PFC can specify speed directly from the underlying geometry of the path. Additionally, a PFC tends to respond more smoothly to disturbances and thus be more robust [15]. A PFC is also beneficial as it affords a smooth approach tangent to the path. However, fewer works have discussed a PFC based on visual servoing, and those that do tend to either plan in 3D space [16], require camera intrinsic parameters [17], or lack orientation following [18].

In our controller design, we improve upon these PFC controllers in terms of flexibility by decreasing the amount of scene and setup information required, and by including orientation following. Our image based path following controller can follow arbitrary paths in image space, both for an eye-to-hand and eye-in-hand camera configuration. An extension of the controller is presented that is can additionally follow path orientation. We assume that the target path is achievable by the robot, and that the robot will not leave the field of view of the camera while following the path. Additionally, we assume the robot has a known kinematic model, and that the camera position is known relative to the robot. Our controller is evaluated with experiments in simulation and on a real robot arm. Performance is compared with a TTC controller.

*We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC).

II. Related Works

Many existing works discuss the problem of path planning in image space for visual servoing, such as Mezouar et al [19], where potential fields are used, Chaumette et al [20] where path planning and trajectory tracking are combined, and Krishnan et al [13], where a sliding mode controller is used to assist the visual servoing control law. However, these works have the drawbacks of a trajectory tracking approach, specifically a poor response to disturbances due to the use of a timed reference.

Several existing works discuss the path following problem with visual servoing for mobile robots. In Cherubini et al [21], both image based and pose based controllers are presented for non-holonomic mobile robots, while Safia et al [22] presents a controller based on landmarks.

In Dahroug et al [16], path following through visual servoing is introduced in a medical robotics setting. The path following task is projected on the null space of a remote center of motion constraint. The control design consists of two weighted terms, one bringing the end effector to the path and one bringing it along the path. However, the path is planned in 3D space, rather than in image space.

A similar problem of contour following is discussed in Duy Cong et al [17] and Chang et al [18], where the former uses a combination of a 3D shortest path visual servoing controller and image based visual servoing to first move parallel to the contour plane. However, this requires the camera intrinsic parameters. The latter uses a SCARA robot alongside PH splines to incorporate a feedforward term into the control law to reduce tracking error.

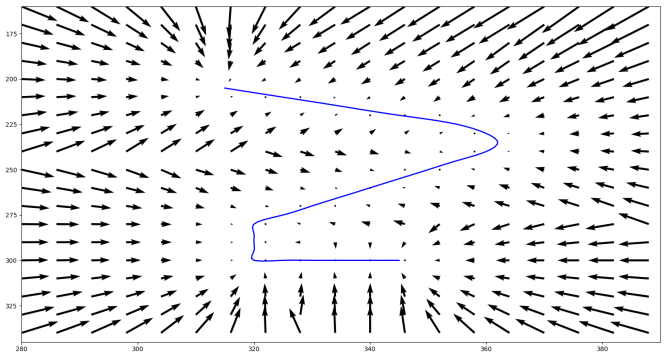
III. Methodology

Given a path in image space with a defined start and end point, our objective is to generate end effector velocities that minimize the error between the current position and the path, and then move the end effector along the path.

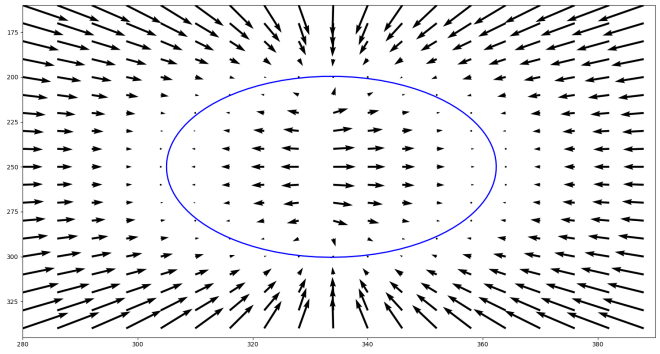
A. Path Definition

First, we discuss how to define the path for the controller. We are looking for an equational curve or function that defines the path in image space. This function can be defined in various ways. One way is through projective geometry, which can be particularly useful in the case of paths that can be represented as lines or conics. In this case, line paths are described as $\mathbf{p} = \{\mathbf{x}|\mathbf{x}^T l = 0\}$, while conic paths are described as $\mathbf{p} = \{\mathbf{x}|\mathbf{x}^T C\mathbf{x} = 0\}$ [23]. Alternatively, for arbitrary curves, it can be convenient to parameterize the curve as $\mathbf{p}(t) = [x(t), y(t)]$, typically with $t \in [0, 1]$ or $t \in [0, \infty)$. Examples of these types of paths are shown in Fig. 1 along with attractive fields of the paths.

Once we have a path function, the visual path error is defined as the error between the current end-effector position in image space, $\mathbf{x} = [x, y] \in \mathbb{R}^2$, and the path, $\mathbf{p}$. To do this, we define a function $\theta(\mathbf{p}, \mathbf{x}) \to \mathbb{R}^2$ which determines the point on the path closest to the current state, and set $\mathbf{x}^* = \theta(\mathbf{p}, \mathbf{x})$. Path error is then defined as $\mathbf{e} = \mathbf{x} - \mathbf{x}^*$



(a) Arbitrary Parameterized Path $\mathbf{p}(t)$



(b) Conic ($\mathbf{x}^T C\mathbf{x}$ = 0) path

Fig. 1: Attractive Vector Fields for Parameterized and Conic Paths

In the case of a parameterized path, one can choose $\theta(\mathbf{p}, \mathbf{x})$ as [24]:

$$\theta(\mathbf{p}, \mathbf{x}) := \mathbf{p}(\arg\min_{t \in T} ||\mathbf{p}(t) - \mathbf{x}||). \tag{1}$$

While in the case of a projective conic, $\theta(\mathbf{p}, \mathbf{x})$ is the function that finds the point that minimizes the orthogonal distance to the conic. One method of doing this is described in Wijewickrema et al [25]. In general, the path can be described by any function in image space, so long as it is possible to define $\theta(\mathbf{p}, \mathbf{x})$ and we can determine the derivative of the path function, $\mathbf{p}'$.

Note that the difference that distinguishes the path function in our PFC from a TTC is that a TTC would define $\mathbf{x}^* = r(t)$, where $r(t)$ is a function that gives a reference target on the path for a given time.

B. Visual Servoing Control Law

We start with the visual servoing controller described in "Visual Servo Control Part I: Basic Approaches" by F. Chaumette and S. Hutchinson [26]. In this section, we assume an eye-to-hand camera configuration. The interaction matrix that relates end effector velocity to feature velocity is given by:

$$\mathbf{L}_{x_c} = \begin{bmatrix} \frac{-1}{Z} & 0 & \frac{x}{Z} & xy & -(1^2 + x^2) & y \\ 0 & \frac{-1}{Z} & \frac{y}{Z} & 1 + y^2 & -xy & -x \end{bmatrix}. \tag{2}$$

Where Z describes the depth of the point relative to the camera frame. As this matrix describes the interaction

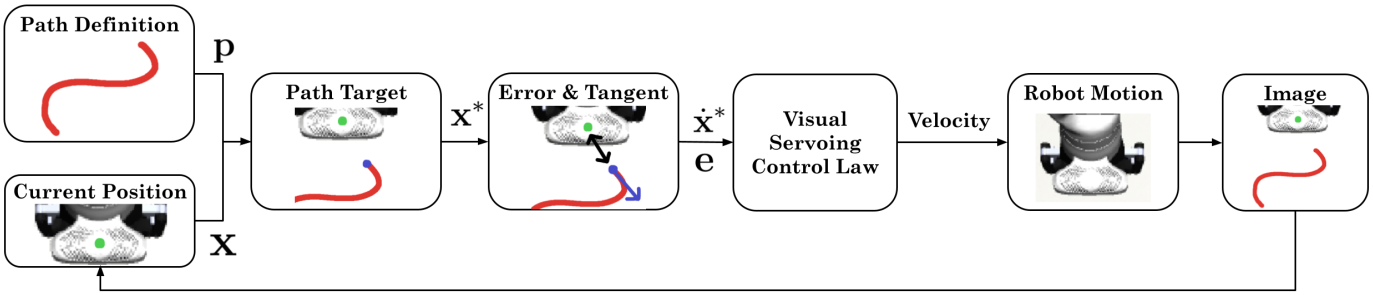


Fig. 2: Overview of visual control loop. Path definition and current robot position provide a nearest path state, which is used to generate error and tangent vectors. The control law then maps these to robot motion.

between camera motion and feature motion, we then apply a transformation to the frame in which we control the robot. In practice, a rough approximation of this transformation is typically sufficient [20]. Assuming we control the robot from the base frame:

$$\mathbf{L}_{x_b} = -\mathbf{L}_{x_c}\, {}^{c}\mathbf{V}_b. \tag{3}$$

Where ${}^{c}\mathbf{V}_b$ is the adjoint representation [27] of the transformation between camera frame and base frame of the robot, $(\mathbf{R}, \mathbf{t}) \in SE3$, given by:

$${}^{c}\mathbf{V}_b = \begin{bmatrix} \mathbf{R} & [\mathbf{t}]_\times \mathbf{R} \\ \mathbf{0} & \mathbf{R} \end{bmatrix}. \tag{4}$$

Since we typically do not have an exact measurement of the depth of a point, Z, we do not have the true value of the interaction matrix. As such, we use an approximation of $\mathbf{L}_{x_b}$. In our work, we choose the approximation presented in [28]:

$$\mathbf{L}_{x_b} = \begin{bmatrix} \alpha_1 \bar{\mathbf{L}}_{x_b} + \alpha_2 \mathbf{L}^*_{x_b} \\ \beta_1 \bar{\mathbf{L}}_{x_b} + \beta_2 \mathbf{L}^*_{x_b} \end{bmatrix}, \tag{5}$$

where $\bar{\mathbf{L}}$ is the interaction matrix at the current point with an approximated depth (Z) value, and $\mathbf{L}^*$ is the interaction matrix at the desired position. $\alpha_1, \alpha_2, \beta_1, \beta_2$ are constants we set to $0.75, 0.25, 0.25$, and 0.75 respectively. As we have doubled the rows of the interaction matrix, we also modify the error vector as follows:

$$\mathbf{e}_{new} = \begin{bmatrix} (\alpha_1 + \alpha_2)\mathbf{e} \\ (\beta_1 + \beta_2)\mathbf{e} \end{bmatrix}. \tag{6}$$

Note that in practice, our choices of $\alpha_1, \alpha_2, \beta_1, \beta_2$ do not affect the magnitude of the error vector as $\alpha_1 + \alpha_2 = 1.0$ and $\beta_1 + \beta_2 = 1.0$.

We now apply the standard control law for IBVS from [26] to drive the arm to the closest point on the path, setting $\dot{\mathbf{e}} = -\lambda \mathbf{e}$:

$$\mathbf{v} = -\lambda \mathbf{L}^+_{x_b} \mathbf{e}, \tag{7}$$

where $\mathbf{v} = [v_x, v_y, v_z, \omega_x, \omega_y, \omega_z]$ is an end effector twist expressed in base frame, λ is a constant, and $\mathbf{L}^+_{x_b}$ indicates the pseudo-inverse of $\mathbf{L}_{x_b}$.

C. Base Path Following Scheme

Now that we have a control that moves the robot toward the path, we introduce a second term that moves the arm smoothly along the path. As the interaction matrix relates image velocities and the end effector twist, if we have a desired image feature velocity, denoted $\dot{\mathbf{x}}^*$, we can obtain a twist by passing it through the same interaction matrix.

$$\mathbf{v}_{des} = \mathbf{L}^+_{x_b} \dot{\mathbf{x}}^* \tag{8}$$

To obtain $\dot{\mathbf{x}}^*$, we take the derivative of $\mathbf{p}$ at $\mathbf{x}^*$. We can then influence the speed at which we move along the path by setting $\dot{\mathbf{x}}^*$ to be a scaled unit vector of the derivative:

$$\dot{\mathbf{x}}^* = s \frac{\mathbf{p}'}{||\mathbf{p}'||}. \tag{9}$$

Since the units of $\mathbf{p}'$ is pixels, the actual end effector speed that a specific value of s corresponds to depends on the distance from the path to the camera.

This target would suffice for most simple paths. However, for paths with tight turns and curves, we can improve our target by taking into account path curvature or the second derivative if they are non-zero. For a parameterized path, curvature can be calculated as:

$$\kappa = \frac{|x'(t)y''(t) - y'(t)x''(t)|}{(x'(t)^2 + y'(t)^2)^{3/2}}, \tag{10}$$

which describes the rotation of the unit tangent vector at t. This can be used to rotate the tangent vector to anticipate the curve. Alternatively, the value of $\mathbf{p}''$ can be added at some scale to our tangent vector to act as an acceleration compensation.

To include the second term in our control law, we weigh the term bringing the arm to the path from (7) and the term bringing the arm along the path. The former decreases in magnitude as $||\mathbf{e}||$ decreases, and so we need to weigh the latter to increase as $||\mathbf{e}||$ decreases. We do this with an exponential function, giving the following control law:

$$\mathbf{v} = -\lambda \mathbf{L}^+_{x_b} \mathbf{e} + \frac{\mathbf{L}^+_{x_b} \dot{\mathbf{x}}^*}{\exp\left(||\mathbf{e}||/d\right)}, \tag{11}$$

where d is a constant that affects the rate at which the second term decays as $\mathbf{e}$ increases. The second term also improves the approach to the path, as once $||\mathbf{e}||$ becomes small, the second

term begins to move along the tangent line, causing a smooth approach in the tangent direction.

D. Orientation Path Following Scheme

For certain tasks, the orientation along the path is important. Therefore, we also provide a modified control law that matches the path orientation as the arm follows the path. To do this, we will modify our control law using the ideas from a 2-1/2d visual servoing control law. [9]

Specifically, we want to control our orientation with the PBVS control law for orientation from [20]:

$$\boldsymbol{\omega} = -\lambda_\omega \boldsymbol{\theta u}, \tag{12}$$

where $\boldsymbol{\theta u}$ is the angle-axis representation of the difference between our current and target orientations. To get a target orientation, we consider the rotation matrix in the camera frame created by the angle of the tangent vector of our current target point, $\mathbf{R}_{cam-path}$. Using the rotation $\mathbf{R}_{base-cam} \in SO3$ from the camera frame to base frame to convert $\mathbf{R}_{cam-path}$ to base frame ($\mathbf{R}_{base-path} = \mathbf{R}_{base-\cancel{cam}}\mathbf{R}_{\cancel{cam}-path}$), we obtain a rotation that we can convert to angle-axis form to obtain $\boldsymbol{\theta u}$. In this way, we obtain a 3D orientation for the path without any knowledge of the 3D position of the path.

Now, if we consider the linear and angular components of an interaction matrix separately:

$$\begin{aligned}\dot{\mathbf{e}} &= \mathbf{L}\mathbf{v} \\ &= \begin{bmatrix}\mathbf{L}_v & \mathbf{L}_\omega\end{bmatrix}\begin{bmatrix}\boldsymbol{v} \\ \boldsymbol{\omega}\end{bmatrix} \\ &= \mathbf{L}_v\boldsymbol{v} + \mathbf{L}_\omega\boldsymbol{\omega}.\end{aligned}$$

Then, by once again setting $\dot{\mathbf{e}} = -\lambda\mathbf{e}$, we can solve for linear velocity:

$$\boldsymbol{v} = -\mathbf{L}_v^+(\lambda\mathbf{e} + \mathbf{L}_\omega\boldsymbol{\omega}). \tag{13}$$

In this control law, the $\mathbf{L}_\omega\boldsymbol{\omega}$ attempts to compensate for the error created by our angular velocity. However, in our experiments, we found that it is better to drop this term, as in practice we found it caused poor path following behavior. This was due to the magnitude of the orientation compensation combining poorly with the path following velocities. As such, to obtain our new control law, we drop the $\mathbf{L}_\omega\boldsymbol{\omega}$ and apply the remainder of (13) to our existing path following control law in (11):

$$\boldsymbol{v} = -\mathbf{L}_{v_b}^+(\lambda_v\mathbf{e}) + \frac{\mathbf{L}_{v_b}^+\dot{\mathbf{x}}^*}{\exp\left((||\mathbf{e}|| + ||\boldsymbol{\theta u}||)/d\right)}, \tag{14}$$

Note that we have also updated the decay term to include $\boldsymbol{\theta u}$ to slow down our movement along the path our orientation error increases. This $\boldsymbol{\theta u}$ can also be scaled in the exponential to change the priority of orientation. Finally, note that we split λ into $\lambda_v, \lambda_\omega$ so that we may choose different constants for our linear and angular convergence.

In practice, this control law functions best if the end effector is rotated to the initial orientation for the path before path following begins.

E. The Eye-In-Hand Case

The control laws described here can also be applied to an eye-in-hand camera setup. In this case, the end effector point, $\mathbf{x}$, should be set to a constant point in the image, and the path must be tracked. This can be achieved either by tracking a contour in the image to serve as a path, or by tracking a set of points and interpolating a path between them.

However, orientation following with this setup is more challenging. In most cases, any rotation of the camera will result in a large change in the image. This can greatly increase the path error and also change the path target point, which then changes the orientation target.

One way to reduce the effect of this issue is to set $\mathbf{x}$ to always be the center of the image. Since our orientation target attempts to keep the camera frame aligned to the path, a change in direction of the path will primarily cause a change in the rotation around the camera's Z axis. If the point we are moving to the path is the center of the camera image, then rotating about the Z axis of the camera will cause little to no movement of this point, assuming our X and Y axes are aligned. This allows the control law to function as normal, at the cost of being able to choose the alignment point in the image freely.

IV. Experimental Results and Discussion

In our experiments, we evaluate the flexibility of the controller by testing a variety of paths with different camera setups. Additionally, we evaluate the performance of our PFC controller in comparison to a TTC when faced with disturbances.

A. Controller Performance

First, we evaluate the performance of the controller on a real robot arm on a variety of paths. For the experiment, we use a Kinova Gen3 7DOF arm. A RealSense D435i depth camera is used as an overhead camera. A RealSense D405 depth camera is used as the eye-in-hand camera. For both cameras, the aligned depth to color image from the realsense camera is used to estimate the value of Z in (2) The camera pose is roughly estimated relative to the robot. A marker is placed on the arm for tracking.

To start, we test paths without orientation matching. Experiments were run with $\lambda = 1.5$, $d = 5.0$, $s = 40.0$. The paths followed are b-splines that give a smooth parameterized path. The control loop is run at a rate of 30 Hz on an AMD Ryzen 7 4700U CPU. In our testing, we have found the control loop can be run at rates upwards of 60Hz, indicating that it is practical for real time control. Results can be seen in Fig 3. We see that path error can be held below 5 pixels on both paths.

Next, to evaluate the controller with orientation following, two curved paths were evaluated with the same parameters as above. Results can be seen in Fig. 4. Fig. 5 shows the velocity profiles for the second curved path. We see that as angular velocity increases to reduce the orientation error, the linear velocities decrease as the second term of the control law

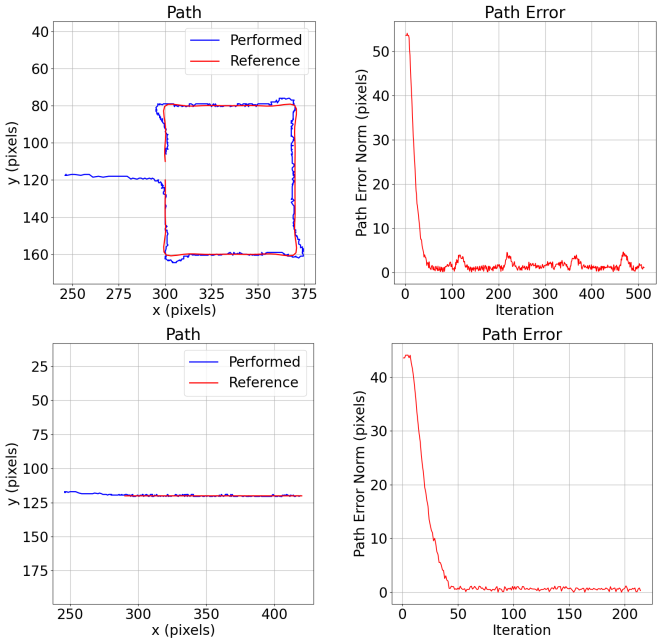


Fig. 3: Controller performance on straight line and square paths

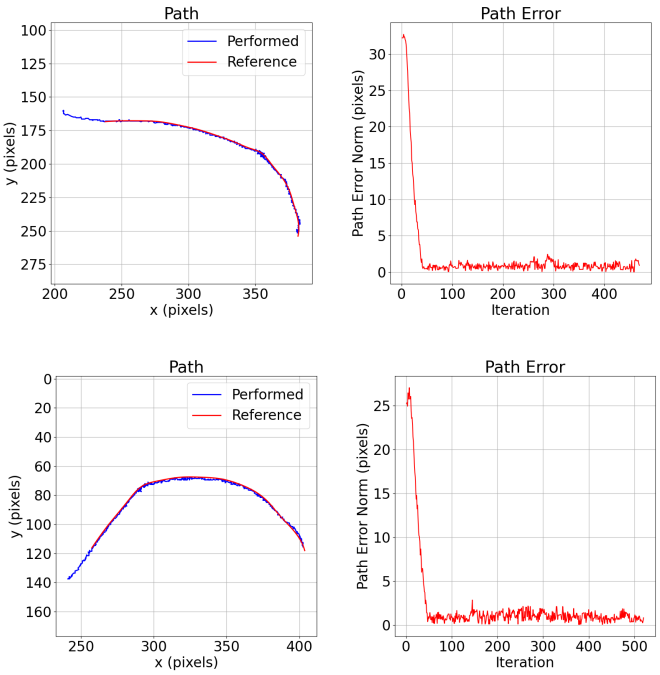


Fig. 4: Controller performance with orientation matching on two curved paths.

is decayed to allow the controller to catch up its orientation. This leads to the peaks and valleys in the velocity profile. It is important to note that the path orientation is not followed as strictly as the path itself. This is because even if we have error in our orientation, we still continue moving along the path unless d is set very low. Thus, orientation sometimes lags behind the path.

Finally, the eye-in-hand case of the controller is evaluated on a contour following problem. Fig. 6 shows the extracted contour and the error following the contour, with the path being tracked and updated at each iteration. The end effector point is selected as the center of the image.

Table I shows a summary of the errors for the paths in this section. Mean path error norm is defined as the mean of the path error once the robot is deemed to have reached the start of the path. For this we begin the mean once error initially drops below 5 pixels. In the case of orientation following, the mean orientation error indicates the mean norm of the orientation error between the end effector orientation and desired orientation. The paths with the highest error are the square and the eye-in-hand contour. For the square, this is due to short periods of time where error spikes at the corners of the square, while for the eye-in-hand contour, it is likely caused by increased instability due to the tracking of the path.

B. Manufacturing Example

We show the use of the controller in a manufacturing example. The end effector of the robot is given a sanding block, and the last joint is spun continuously to act as an automatic sanding tool. The controller is used to sand along a path obtained from the image. Orientation following is not

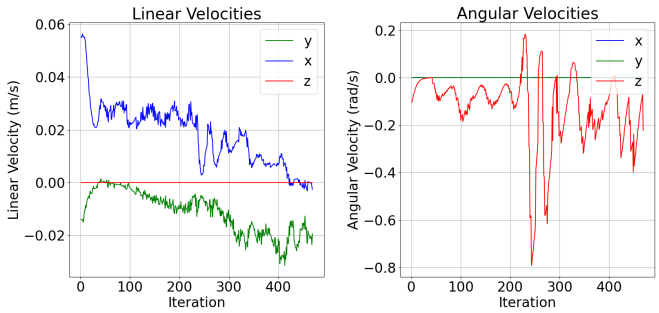


Fig. 5: Velocity profile for the first curved path.

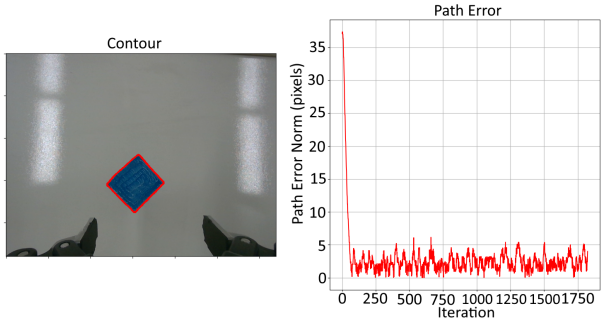


Fig. 6: Controller performance on an eye-in-hand contour following task.

Path Type	Mean Norm Path Error (pixels)	Mean Orientation Error (radians)
Line	0.69896	N/A
Square	1.55830	N/A
Curve 1	0.85025	0.07394
Curve 2	1.00932	0.07293
Eye-In-Hand Contour	3.34503	N/A

TABLE I: Summary of Path Errors in section IV.A

used in this experiment. The experimental setup is shown in Fig. 7 For this task, a visual contour was placed in the

Fig. 7: Setup for sanding task.

view of the camera next to the robot's target workspace. The contour is extracted using edge detection, rescaled, and sent as a path target in the robot's work area. This illustrates one way that paths can be generated from visual information in the environment. Path following control is then applied. Results from this task can be seen in Fig. 8. Despite the surface contact from the sanding, the controller is still able to follow the path with low error. This shows the ability of the control law to function despite the additional disturbances from friction.

C. Comparison with a Trajectory Tracking Controller

Next, we analyze the choice of a path following controller (PFC) by comparing it to a simple trajectory tracking controller (TTC). The TTC uses the standard IBVS control law to follow a series of timed targets. To compare the controllers, we consider a situation where the robot being forced to temporarily stop along the path. This is shown in Fig. 9, where the diamond shows the location of a forced pause. We can see that the TTC has undesirable performance due to falling behind the timed reference, while the PFC is unaffected and behaves as normal. Fig. 10 shows a similar result in responding to a disturbance. The TTC has undesirable behavior and cuts across the path while catching up to the timed reference, while

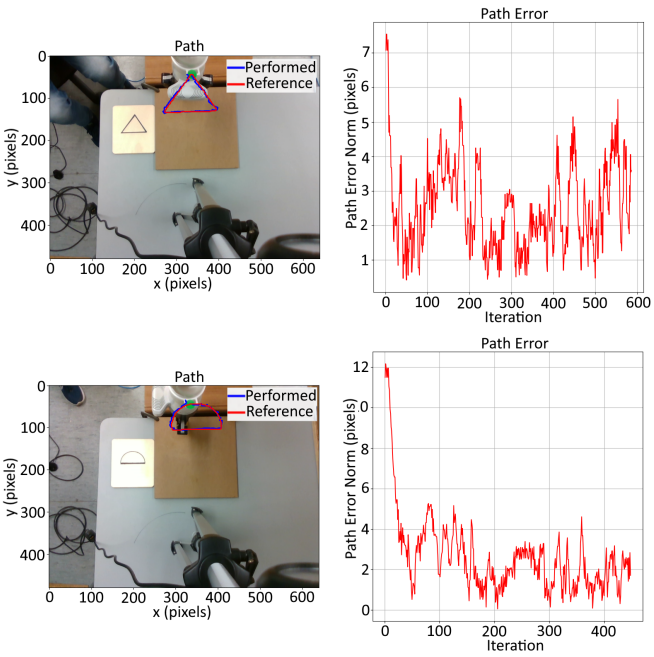
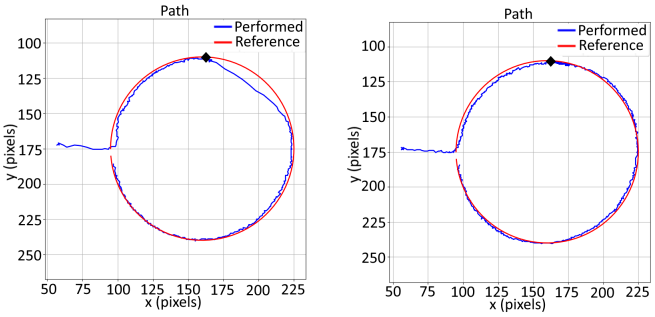


Fig. 8: Sanding path followed for semicircle and triangle contours.

the PFC reapproaches the path smoothly along the tangent direction. In both cases, we can see that the use of a PFC leads to a better response.



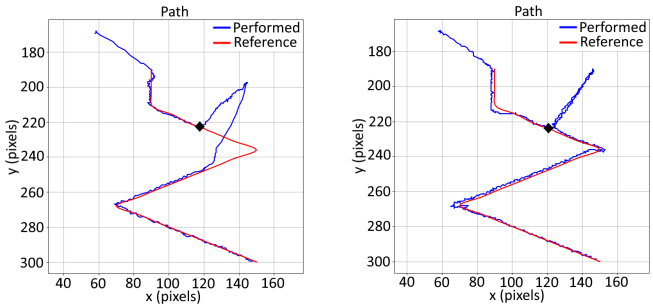
(a) TTC (b) PFC

Fig. 9: TTC and PFC Performance on circular path after a forced pause

V. Analysis of Parameters

To discuss the design of the controller, we analyze the effect of the 3 main parameters in our control law: λ, d, and s. Recall that λ affects the gain that brings us to the path, d affects the decay of the term moving us along the path as $\mathbf{e}$ changes, and s affects the speed we move along the path.

To run hundreds of experiments, a simulation environment was created using Robosuite [29] with a Kinova Gen3 arm. The Robotics Toolbox for Python [30] is also used to obtain additional kinematic information from the simulation.



(a) TTC (b) PFC

Fig. 10: TTC and PFC Performance on zigzag path after a disturbance

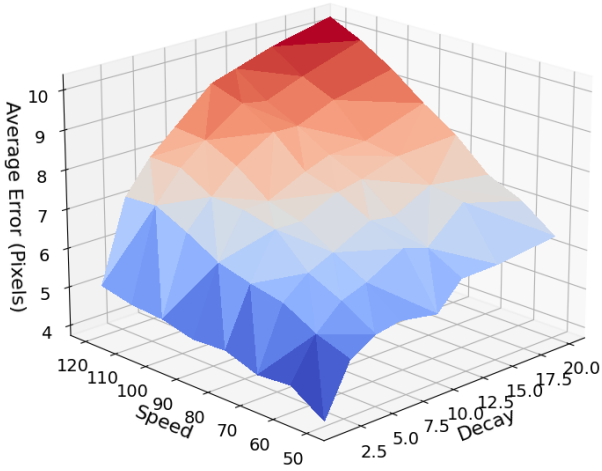
Fig. 11 shows that increasing s and decreasing d both increase the average path error, while decreasing the total number of iterations. Increasing speed generally leads to less reaction time for the controller, leading to overshooting on curves or turns. This can be partially compensated for by decreasing d, as a lower d will lead to a more accurate path approach, and will reduce the effect of overshooting by slowing down if we leave the path.

Fig. 12 shows the relationship between λ, d, and error. Note that in this data, the robot starts on the path. A higher λ decreases path following error as it increases the strength the term that pulls us to the path. However, if λ is set too high, it can lead to undesirable behavior when approaching the path, as it will increase approach speed and may lead to overshooting the path or higher than desired velocities.

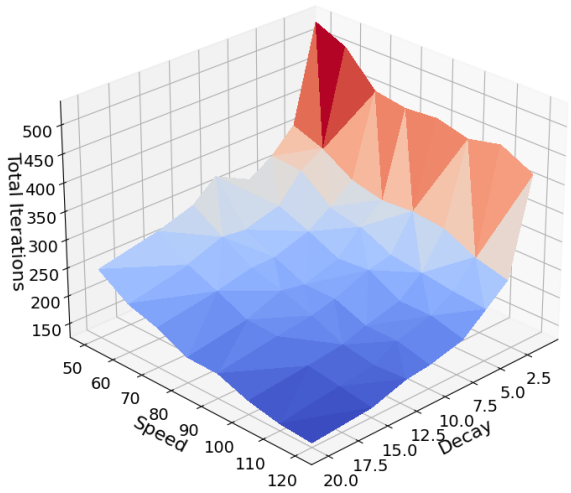
In general, path error can be kept low if s is low, as the controller will have more reaction time to update velocities as the path changes. Thus, when it comes to choosing values for these parameters, one should first determine the desired s value, and then tune λ and d until path error is sufficiently low. Note that since s and d are measured in image pixels, parameter values will be different depending on how close the camera is to the path.

VI. Conclusion and Future Work

This paper presented a solution to the problem of following a path defined in image space. The PFC framework chosen is used to follow the path with a configurable speed parameter. The results show the controller is able to follow a variety of paths with pixel error under 5 pixels. Future research include investigating alternate methods of controlling orientation in a non-decoupled way. Use of other scene features may allow for more effective orientation following schemes. Incorporating a method for projecting a desired 3D velocity profile into image space could allow for more precise specification for path following speed. Finally, combining this controller with path planning methods that ensure a visibility constraint and joint limit constraints could lead to a more complete and robust system for path following task specification.



(a) Z: Average Error



(b) Z: Total Iterations

Fig. 11: Effect of decay and speed on control performance

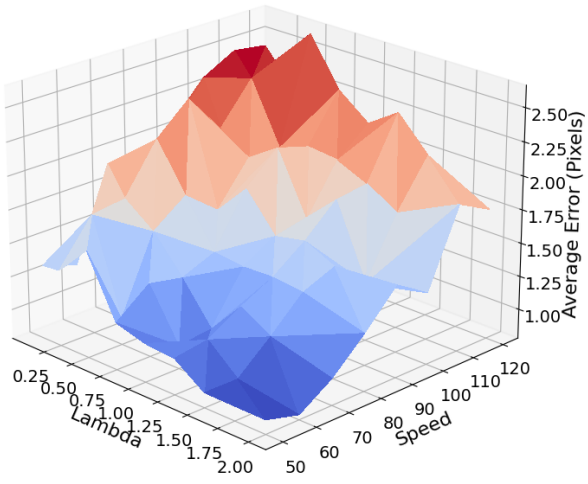


Fig. 12: Effect of lambda and speed on average error

REFERENCES

[1] S. Hutchinson, G. Hager, and P. Corke, "A tutorial on visual servo control," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 5, pp. 651–670, 1996.

[2] M. Jagersand, O. Fuentes, and R. Nelson, "Experimental evaluation of uncalibrated visual servoing for precision manipulation," in *Proceedings of International Conference on Robotics and Automation*, vol. 4, pp. 2874–2880 vol.4, 1997.

[3] T. Lampe and M. Riedmiller, "Acquiring visual servoing reaching and grasping skills using neural reinforcement learning," in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2013.

[4] T. Laliberte and C. Gosselin, "Efficient algorithms for the trajectory planning of redundant manipulators with obstacle avoidance," in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pp. 2044–2049 vol.3, 1994.

[5] S. Klanke, D. Lebedev, R. Haschke, J. Steil, and H. Ritter, "Dynamic path planning for a 7-dof robot arm," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3879–3884, 2006.

[6] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt, "Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration," *ACM Trans. Graph.*, vol. 36, may 2017.

[7] R. Al-Jarrah, M. Al-Jarrah, and H. Roth, "A novel edge detection algorithm for mobile robot path planning," *Journal of Robotics*, vol. 2018, no. 1, p. 1969834, 2018.

[8] A. P. Aguiar, D. B. Dačić, J. P. Hespanha, and P. Kokotović, "Path-following or reference tracking?: An answer relaxing the limits to performance," *IFAC Proceedings Volumes*, vol. 37, no. 8, pp. 167–172, 2004. IFAC/EURON Symposium on Intelligent Autonomous Vehicles, Lisbon, Portugal, 5-7 July 2004.

[9] E. Malis, F. Chaumette, and S. Boudet, "2 1/2 d visual servoing," *IEEE Transactions on Robotics and Automation*, vol. 15, no. 2, pp. 238–250, 1999.

[10] K. Wang, Y. Liu, and L. Li, "Visual servoing trajectory tracking of nonholonomic mobile robots without direct position measurement," *IEEE Transactions on Robotics*, vol. 30, no. 4, pp. 1026–1035, 2014.

[11] G. Allibert, E. Courtial, and Y. Touré, "A flat model predictive controller for trajectory tracking in image based visual servoing," *IFAC Proceedings Volumes*, vol. 40, no. 12, pp. 993–998, 2007. 7th IFAC Symposium on Nonlinear Control Systems.

[12] F. Yan, B. Li, W. Shi, and D. Wang, "Hybrid visual servo trajectory tracking of wheeled mobile robots," *IEEE Access*, vol. 6, pp. 24291–24298, 2018.

[13] M. G. Krishnan and A. Sankar, "Image space trajectory tracking of 6-dof robot manipulator in assisting visual servoing," *Automatika*, vol. 63, no. 2, pp. 199–215, 2022.

[14] J. Fried, A. C. Leite, and F. Lizarralde, "Uncalibrated image-based visual servoing approach for translational trajectory tracking with an uncertain robot manipulator," *Control Engineering Practice*, vol. 130, p. 105363, 2023.

[15] M. A. Lawati and A. F. Lynch, "Path-following control for a slung load system," in *2023 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, pp. 247–254, 2023.

[16] B. Dahroug, B. Tamadazte, and N. Andreff, "Visual servoing controller for time-invariant 3d path following with remote centre of motion constraint," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3612–3618, 2017.

[17] V. Duy Cong and L. Duc Hanh, "Combination of two visual servoing techniques in contour following task," in *2021 International Conference on System Science and Engineering (ICSSE)*, pp. 382–386, 2021.

[18] W.-C. Chang, M.-Y. Cheng, and H.-J. Tsai, "Image feature command generation of contour following tasks for scara robots employing image-based visual servoing—a ph-spline approach," *Robotics and Computer-Integrated Manufacturing*, vol. 44, pp. 57–66, 2017.

[19] Y. Mezouar and F. Chaumette, "Path planning for robust image-based control," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 4, pp. 534–549, 2002.

[20] F. Chaumette and S. Hutchinson, "Visual servo control, Part II: Advanced approaches," *IEEE Robotics and Automation Magazine*, vol. 14, no. 1, pp. 109–118, 2007.

[21] A. Cherubini, F. Chaumette, and G. Oriolo, "Visual servoing for path reaching with nonholonomic robots," *Robotica*, vol. 29, no. 7, p. 1037–1048, 2011.

[22] F. Safia and C. Fatima, "Visual path following by an omnidirectional mobile robot using 2d visual servoing," in *2017 5th International Conference on Electrical Engineering - Boumerdes (ICEE-B)*, pp. 1–7, 2017.

[23] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second ed., 2004.

[24] J. Hauser and R. Hindman, "Maneuver regulation from trajectory tracking: Feedback linearizable systems*," *IFAC Proceedings Volumes*, vol. 28, no. 14, pp. 595–600, 1995. 3rd IFAC Symposium on Nonlinear Control Systems Design 1995, Tahoe City, CA, USA, 25-28 June 1995.

[25] S. Wijewickrema, C. Esson, and A. Paplinski, "Orthogonal distance least squares fitting: A novel approach," vol. 68, pp. 255–268, 01 2010.

[26] F. Chaumette and S. Hutchinson, "Visual servo control, Part I: Basic approaches," *IEEE Robotics and Automation Magazine*, vol. 13, no. 4, pp. 82–90, 2006.

[27] K. Lynch and F. Park, *Modern Robotics: Mechanics, Planning, and Control*. Cambridge Univeristy Press, 2017.

[28] E. Nematollahi and F. Janabi-Sharifi, "Generalizations to control laws of image-based visual servoing," *International Journal of Optomechatronics*, vol. 3, no. 3, pp. 167–186, 2009.

[29] Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, A. Joshi, S. Nasiriany, and Y. Zhu, "robosuite: A modular simulation framework and benchmark for robot learning," in *arXiv preprint arXiv:2009.12293*, 2020.

[30] P. Corke and J. Haviland, "Not your grandmother's toolbox–the robotics toolbox reinvented for python," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11357–11363, IEEE, 2021.