Image based visual servoing for eye-in-hand industrial manipulators

Shivam Thukral
Department of Computer Science
University of British Columbia
tshivam2@cs.ubc.ca

Abstract

The use of information acquired by stationary or on-board vision sensors for feedback control of pose and motion of a robot is known as visual servoing. In this paper, an image-based visual servoing technique is developed which uses the images from a RGB camera in eye-in-hand configuration with occlusion handling. The proposed method chooses a set of correctly extracted image features, and obtains an estimate of all the image features from the correctly extracted image features. The estimation procedure makes it possible to track image features even when occlusion occurs. A velocity controller is developed for a 6-DoF robotic manipulator which minimises the error between current and desired image features. At each control loop, end-effector velocity is estimated from image interaction matrix and it is used to generate joint velocities from robot jacobian. An extensive set of experiments are carried out to indicate the feasibility of the proposed approach. Finally, a dexterous manipulation capability is shown by using the robot to play ping-pong.

1 Introduction

Visual servo control relies on techniques from image processing, computer vision, and control theory to control the pose and motion of a robot. This work mainly focuses on Image-Based Visual Servo (IBVS) in which the error is generated directly from image plane features [1] [2]. This set of features usually correspond to the projected image coordinates of several 3D points on the tracked target. For a stable visual servo control law, the difference between actual and desired image feature locations should decrease asymptotically.

The vision data may be acquired from a camera that is mounted directly on the manipulator, in which case motion of the robot includes the camera motion or the camera can be fixed in the work-space so that it can observe the robot motion form a stationary configuration. Other configurations can be considered, such as for instance several cameras mounted on pan-tilt heads observing the robot's motion. The mathematical development of all the cases is similar, and in this work I will focus primarily on the former, so-called eye-in-hand, case.

Broadly, visual servoing methods are divided into two categories: Position Based Visual Servoing (PBVS) and IBVS. IBVS controller directly operates on the error between the current and desired features in the image space, rather than using conversion from image space to Cartesian space as in PBVS. Postion-based control is most suitable when we want to define tasks in a standard Cartesian frame. For PBVS, the control law strongly depends on the optical parameters of the vision system and can become widely sensitive to calibration errors. On the contrary, the image-based control is less sensitive to calibration errors; however, it requires online calculation of the image jacobian which is difficult to evaluate. IBVS methods also have the advantage of avoiding the additional estimation step as required in PBVS. Recently, a new family of hybrid methods are growing which combines advantages of PBVS and IBVS while trying to avoid their shortcomings [3]. The IBVS control scheme has been utilized for a wide range of applications, such as industrial and manufacturing processes, medical robotics, unmanned vehicles guidance and navigation.

The novel contributions made in this paper are as follows:(1) An image based visual servoing technique is presented, which extracts the image features using CAMshift algorithm and uses these features to find image jacobian (2) A velocity controller for UR5 industrial manipulator is developed using ROS and Gazebo which minimises the error between current and desired image features, (3) Experimental validation of the proposed framework is done using a 6-DoF UR5 robotic manipulator. (4) An application based example is demonstrated in which robot plays ping-pong.

2 Background and Related Work

In an IBVS framework with eye-in-hand configuration, the following procedure is pursued. First, visual data with sensor noise is fetched from the camera. In these images, real-valued quantities associated to the geometric primitives (e.g., coordinates of points, gradients of lines, areas of ellipses, etc.) are called *visual features*. In the next stage of servoing, these features are extracted from still images using image processing techniques such as color to grayscale conversion, binarization, noise cancellation, quality optimization, edge detection, and segmentation. After that, the error is calculated by comparing the extracted feature vectors to the desired feature vector. The Cartesian velocity vector of the camera is then computed using the mapping between error signal and camera velocity which is given by image interaction matrix. Finally, this velocity vector is sent to the controller in order to generate the appropriate motion for the robot.

Eye-in-hand IBVS systems suffer from two critical issues. The tracked object features might escape the vision system's FoV during the visual servoing [4] and generation of robot motions which are impossible to execute on robot [5]. Therefore, the satisfaction of visibility constraints and robots physical constraints like joint limits, singularities are required to collectively addressed.

Several approaches have been proposed to circumvent these limitations. Sauvee et al. [6] presented a Nonlinear Model Predictive Control (NMPC) approach, in which dynamic model of the manipulator, its joint and torque limits, visibility constraints and camera projection model is considered. Alibert et al. [7] proposed a visual predictive control strategy based on a nonlinear global model combined with Internal Model Control (IMC) structure. Approaches such as these require a large number of tuning parameters and are application specific.

A recent shift in trend has been seen for using redundant robots because of its ability to perform complex motions by using more degrees-of-freedom than required for the task. In [8], redundancy of a manipulator is exploited to avoid joint limits violation. Such robots can have an infinite number of solutions existing in the configuration space for a point in image. Thus, mapping of image space onto the configuration space is a major challenge in this case. A generic method exploiting redundancy of a space robot for performing visual servoing is recently presented in [9].

This work mainly focuses on handling the first assumption in which the target occluded. Some effort has been devoted to vision-based control with occlusion handling. A vision based control method which predicts occulsion has been presented in [10]. This method works against self-occulsions which can be predicted thorugh Binary Space Partitioning Tree Model[11]. A weighted image features approach is presented in [12]. The weighted approach cannot determine whether visual tracking fails or not. Thus, for each image feature, an error between the measurement and an estimate should be compared. If the error is bigger than a given tolerance, the visual tracking should fail.

3 Image-based Visual Servoing

The aim of all vision-based control schemes is to minimise an error ***e(t)***, which is defined by

$$e(t) = s(m(t),a) - s* \tag{1}$$

The parameters in 1 are defined as follows. The vector *m(t)* is a set of image measurements (e.g., the image coordinates of interest points, or the parameters of a set of image lines or segments). These image measurements are used to compute a vector of k visual features, *s(m(t), a)*, in which *a* is a set of parameters that represent potential additional knowledge about the system (e.g., true or approximate camera intrinsic parameters or a model of the object to be tracked). The vector $s*$ contains the desired values of the features.

Once s is selected, we can design a control scheme. The most straightforward approach is to design a velocity controller. For this, we need to find the relationship between camera velocity v_c and time variation of s. The relationship is given by

$$\dot{s} = J_s v_c \tag{2}$$

in which $J_s \in \mathbb{R}^{kx6}$ is called the interaction matrix or image jacobian. Using 1 and 2, the relationship between camera velocity and the time variation of error is given by

$$\dot{e} = J_e v_c \tag{3}$$

where, $J_e = J_s$, and considering v_c as the input to robot controller, we obtain as controller

$$v_c = -\lambda J_e^+ e(t) \tag{4}$$

where, $J_e^+ \in \mathbb{R}^{6xk}$ is the Moore-Penrose pseudo-inverse of J_e. Here, J_e^+ is given by $J_e^T(J_e J_e^T)^{-1}$.

3.1 The Interaction Matrix

The interaction matrix for an IBVS technique is given by

$$\begin{bmatrix} -f/z & 0 & u/Z & uv/f & -f^2-u^2/f & v \\ 0 & -f/z & v/Z & f^2+v^2/f & -uv/f & -u \end{bmatrix} \tag{5}$$

where, $P = \{X, Y, Z\}$ is a three dimensional point and its projection onto two dimensional image plane is given by $p = \{u, v\}$ and f is the focal length of the camera. A detailed derivation of interaction matrix is presented in [13].

The paper tries to solve the above explained IBVS problem of identifying and extracting the image features. Next, the current and desired features are compared to calculate the error. In cases where the target features are occluded an estimate of these visual features is used to compute the error. The velocity vector of the camera is then computed by applying this error to the visual servoing controller. Finally, this velocity vector is sent to robot controller to generate the joint velocities.

4 Proposed Framework

The proposed solution to problem mentioned in Section 3 is divided into following steps: (1) Image features were extracted using CAMShift [14] algorithm. (2) Kalman filter[15] was used in cases where target feature is occluded to predict the position in image plane. (3) Joint velocities of the manipulator were generated from a velocity controller. These steps are further explained in following subsections:

4.1 Feature Extraction

Object tracking is a key task in the field of computer vision. Real-time performance and low computation requirements, are two key features of a practical tracking algorithm in real-world applications. Mean shift is kernel based object tracking algorithm which uses density-based appearance models for target representation. It has been widely used because of its simplicity and low computation cost, but mean-shift would fail in changing the track window's scale, as targets move toward and away form the camera.

Based on mean-shift, continuous adaptive mean-shift (CAMshift) was proposed to overcome this problem. CAMshift adaptively adjusts the track window's size and the distribution pattern of targets during tracking. Most researchers use color data to represent the targets for CAMshift, this would give a low complexity and practical performance.

For the purpose of this work, a sphere blue ball is used as the target object and blue ball's centroid represents the tracked visual feature.

4.2 Target Occlusion

The CAMshift algorithm is not well suited for tracking objects in the presence of full occlusion. Kalman filter is used to estimate the position of tracked object in the cases of occlusion. The Kalman filter algorithm belongs to a class of state space approaches since it uses the state space equations and measurement equations to track the object. Kalman filter divides the object tracking into two steps: prediction and update. Prediction allows to predict the position of the object knowing its history and is corrected every time a measure of the state of the object is available, this correction makes the update. In order to simplify the problem, we only consider linear motion model of the tracked ball.

The state vector is composed of position (x,y),velocity (v_x, v_y) and acceleration (a_x, a_y). The measurement vector is composed of position x,y. The transition matrix *(A)*, measurement matrix *(H)* and process noise *(Q)* are given by

$$A = \begin{bmatrix} 1 & 0 & dt & 0 & {dt^2}/2 & 0 \\ 0 & 1 & 0 & dt & 0 & {dt^2}/2 \\ 0 & 0 & 1 & 0 & dt & 0 \\ 0 & 0 & 0 & 1 & 0 & dt \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} Q = \begin{bmatrix} 0.01 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 \end{bmatrix}$$

4.3 Velocity Controller

In order to asymptotically decrease the error between current and desired image features, the robot arm needs joint velocities. The camera Cartesian velocity V_c = (v_c, ω_c) is computed from image interaction matrix. The following steps are used to find the joint velocities:

4.3.1 Forward Kinematics

The goal of calculating forward kinematics is to estimate end-effector pose from the position of the joints. A robotic arm is a series of links which connects the base of the robot to its end-effector. Each link is connected to the next by an actuated joint. The relationship between neighbouring links is described by a homogeneous transformation matrix, denoted by T_j^{j-1}.

$$T_j^{j-1} = \begin{bmatrix} cos\theta & -sin\theta cos\alpha & sin\theta sin\alpha & rcos\theta \\ sin\theta & cos\theta cos\alpha & -cos\theta sin\alpha & rsin\theta \\ 0 & sin\alpha & cos\alpha & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}$$

Here, (d, θ, r, α) are the Denavit-Hartenberg (DH) parameters for each joint. The DH parameters break down each joint of the robot into four parameters, each taken with reference to the previous joint

A forward kinematic transform is described using a series of T matrices from base of the robot to its end-effector. For manipulator with 6 joints, the transform from base to end-effector is given by

$$T_E^B = T_0^1.T_1^2.....T_5^6 \tag{8}$$

Finally, we invert the T_E^B matrix to go from end-effector velocity to velocity in the base frame.

4.3.2 Robot Jacobian

The robot jacobian gives a relationship between the joint velocites and the end-effector velocity. In short, the end-effector linear velocity v and angular velocity ω as function of $\dot{\theta}$ is described as

$$V(v, \omega) = J(\theta)\dot{\theta} \tag{9}$$

The manipulator used in experiments has 6 revolute joints thus robot jacobian is a $6x6$ matrix which can be partitioned into $3x1$ column vectors as shown

$$J(\theta) = \begin{bmatrix} J_{p1} & J_{p2} & J_{p3} & J_{p4} & J_{p5} & J_{p6} \\ J_{o1} & J_{o2} & J_{o3} & J_{o4} & J_{o5} & J_{o6} \end{bmatrix} \tag{10}$$

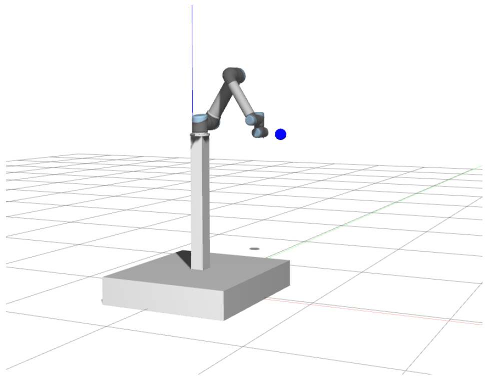


Figure 1: Gazebo Environment

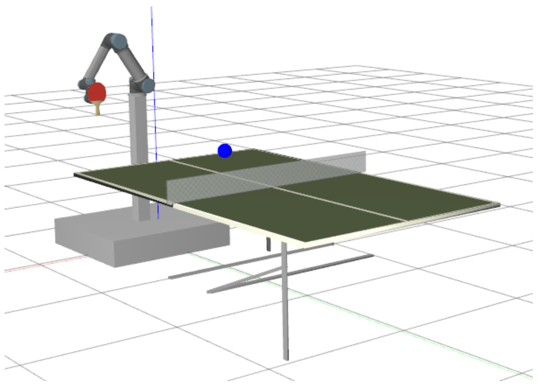


Figure 2: Ping-Pong Experiment

Since all the joints are revolute, J_{pi} and J_{oi} are computed by:

$$\begin{aligned} J_{pi} &= z_{i-1} \times (P - p_{i-1}) \\ J_{oi} &= z_{i-1} \end{aligned} \tag{11}$$

where, P is position of the end-effector with respect to base and p_{i-1} is the position of revolute joint with respect to base frame which can be calculated using forward kinematics.

Kinematic analysis of robots have a high chance of making errors in performing repetitive operations involving matrices. Due to this, I have used a MATLAB Symbolic Robotics Toolbox that performs numerical computations and dynamic analysis in symbolic form. For my purpose of my experiments, I have used the symbolic jacobian generated by the toolbox and substituted numerical values for link parameters and joints to calculate the actual geometric robot jacobian. Finally, the inverted robot jacobian is used to give the joint velocities from end-effector velocities in base frame calculated in the previous sub-section.

5 Simulations

In this section, a set of 4 different experiments are explained and conducted in simulation using Robotics Operating Systems (ROS) and a physics simulation engine Gazebo. I describe the experiments in the subsequent subsection.

5.1 Simulation Setup

For experiments, I have selected 6-DoF Universal Robot 5 (UR5) as the manipulator and a Microsoft Kinect V2 camera which is mounted on the end-effector of the robot. I have modified the URDF of UR5 to attach the camera to its end-effector. A custom deisgned ROS-Gazebo plugin is being used to control the motion of the joints. Lastly, the trajectories of the target blue ball are also modelled through a gazebo plugin.
The experiments are sub-divided as follows:

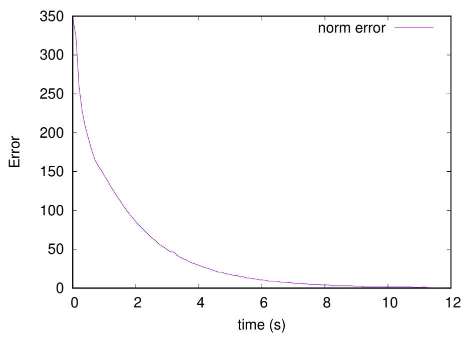
1. *Stationary target*: A stationary target ball is being tracked here. The desired features is the center of the image. The ball can be tracked in both, 2D and 3D. Figure 1, shows the Gazebo environment used.
2. *Moving target without occlusion*: In this case, linear motion of the ball is being tracked by the industrial manipulator.
3. *Moving target with occlusion*: Whenever the target object is occluded, an estimated location is being used from the Kalman filter.
4. *Ping Pong*: For this experiment, a special Gazebo environment is made. In Figure 2, a paddle is mounted on the robots end-effector and a separate gazebo plugin is used to move the ball over the ping pong table.

For the experiments, I have gradually increased the speed of the moving target to test the robustness of the tracking algorithm and velocity controller.

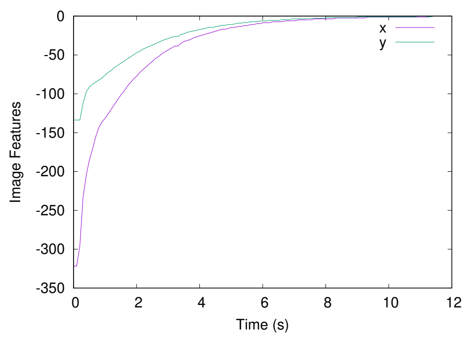
6 Results

6.1 Stationary Target

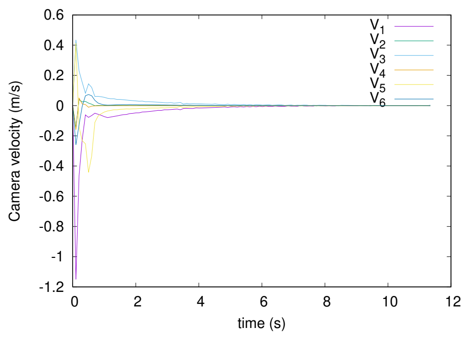
In this experiment, the ball was placed randomly in the gazebo environment and the tracking algorithm was used to track the ball in 2D as well in 3D. Figure 3, summaries the results of tracking in 2D environment. The error in x and y image plane converges to zero. As we reach close to the target, the camera and joint velocities also converges. For tracking results in 3D please refer images in Appendix.



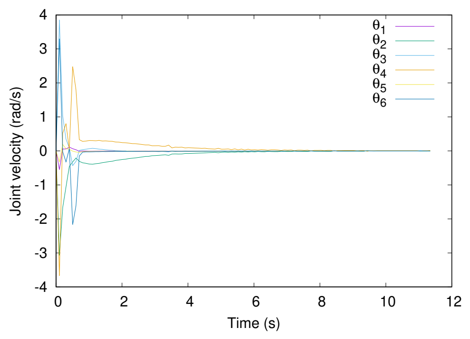
(a) Normalised Error



(b) Tracked Features



(c) Camera Velocity



(d) Joint Velocity

Figure 3: Plots for tracking in 2D

6.2 Moving target without occlusion

A gazebo plugin was used to move the ball with a linear motion model. I gradually increased the speed of the ball to test if the robot is able to track the object. Table 1, summarizes the mean trajectory error in each case. For each case, the desired feature is image center and this proposed approach would give decent results if the speed of the moving object is less than 15 cm/s.

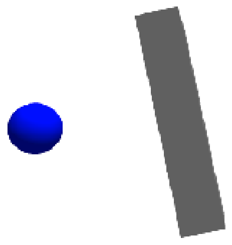
Table 1: Summary of results from tracking a moving ball

Speed(cm/s)	5	10	15	20	25
Error(cm)	3.11	6.25	11.51	16.62	23.39

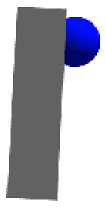
6.3 Moving target with occlusion

In the places where the target object is not visible, an estimate from Kalman filter is used. For this experiment, I randomly occluded the target object to see how well Kalman filter performs.

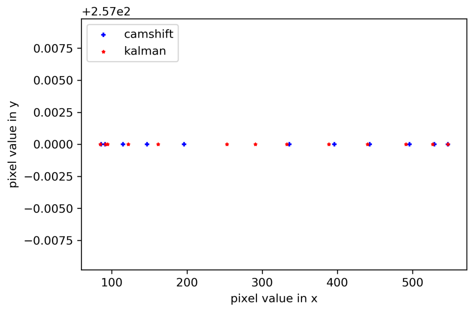
A comparison between Kalman filter estimates and tracking results from CAMShift algorithm is depicted in Figure 4. In both the cases, the occlusion happens roughly in the center of the image.



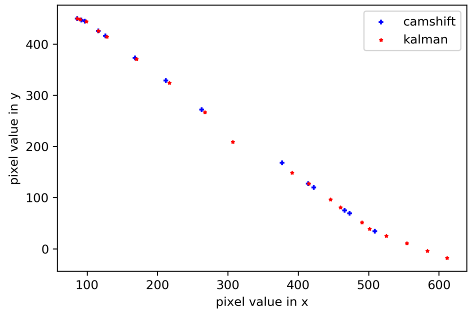
(a) Ball movement with occlusion for (c)



(b) Ball movement with occlusion for (d)



(c) Tracking in x-plane



(d) Tracking in xy-plane

Figure 4: Plots for Kalman filter tracking

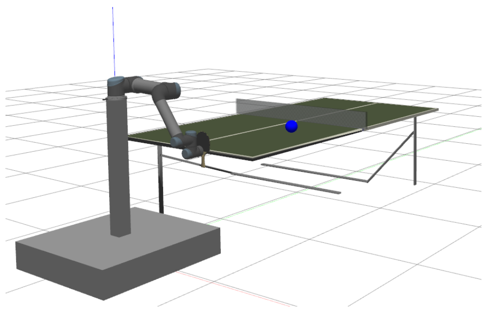
6.4 Ping-Pong Experiment

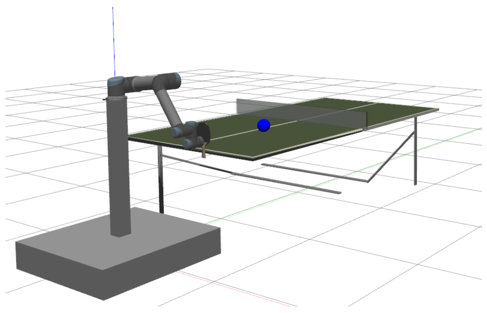
For this experiment, I modelled the trajectory of a ping pong ball using a gazebo plugin. The ball starts from the ping pong table and moves towards the robot. I setup 10 random locations on the table for this task. A successful event occurs when the robot is able to hit the ball using the paddle. All the other events are consider as failure. Figure 5, shows a demonstration of one such run. Out of 10 random runs, the manipulator was able to hit 6 times at a speed 15 m/s.

7 Discussions

The proposed approach was able to track a stationary object in 2D/3D in all the cases. The error in each of the simulations converged to zero and as the manipulator got close to the desired feature, the joint velocities also decreased. For moving target, the speed of the target played a crucial role in its tracking. Slow moving targets have been tracked successfully with small errors while on the other hand high speed targets resulted in a higher tracking error. Better results can be achieved using finely tuned controller parameters and Kalman gain parameters. Currently, only linear motion model is considered for target object. For non-linear motion, Extended Kalman filter can be used.

The success rate of the ping-pong experiment did not match my expectation. One way to increase the hit rate is to properly model the trajectory of the ball at the image processing side. Using this estimated trajectory, we can calculate the fastest interception point between ball and the robot and send this to the controller. I intended to implement this, but due to time constraints I was not able to complete the code required for this.


(a)


(b)

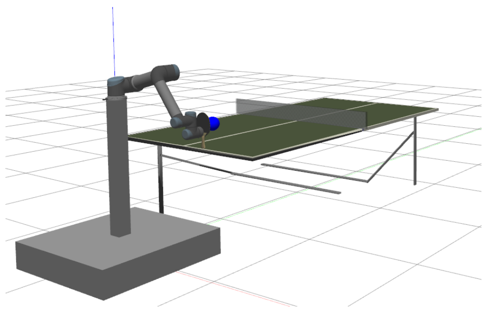

(c)

Figure 5: Robotic Manipulator playing ping-pong from (a)-(c)

8 Conclusions

This paper proposed a real-time image-based visual servoing tracking approach which copes with occlusion with a small computational cost. Results were shown with an industrial manipulator in simulation using ROS and Gazebo. A custom designed velocity controller was used to calculate the robot jacobian and to supply joint velocities. An example of human-robot interaction was shown through ping-pong experiment.In the future, I plan to investigate and demonstrate ways to increase the tracking speed of the robot. Futhermore, I intend to use extended-Kalman filter to track non-linear motions of the target object. In addition to this, I will deliberate upon the use of other tracking algorithm like SIFT, SURF and ORB to track the visual features. All the code can be found here.

References

[1] Seth Hutchinson, Gregory D Hager, and Peter I Corke. A tutorial on visual servo control. *IEEE transactions on robotics and automation*, 12(5):651–670, 1996.

[2] Odile Bourquardez, Robert Mahony, Nicolas Guenard, François Chaumette, Tarek Hamel, and Laurent Eck. Image-based visual servo control of the translation kinematics of a quadrotor aerial vehicle. *IEEE Transactions on Robotics*, 25(3):743–749, 2009.

[3] François Chaumette and Seth Hutchinson. Visual servo control. ii. advanced approaches [tutorial]. *IEEE Robotics & Automation Magazine*, 14(1):109–118, 2007.

[4] G Chesi, K Hashimoto, D Prattichizzo, and A Vicino. Keeping features in the camera's field of view: a visual servoing strategy. In *15th Int. Symp. on Mathematical Theory of Networks and Systems*, volume 68. Notre-Dame Indiana, 2002.

[5] Eric Marchand, Alessandro Rizzo, and Francois Chaumette. Avoiding robot joint limits and kinematic singularities in visual servoing. In *Proceedings of 13th International Conference on Pattern Recognition*, volume 1, pages 297–301. IEEE, 1996.

[6] Mickaël Sauvée, Philippe Poignet, Etienne Dombre, and Estelle Courtial. Image based visual servoing through nonlinear model predictive control. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 1776–1781. IEEE, 2006.

[7] Guillaume Allibert, Estelle Courtial, and Youssoufi Touré. Visual predictive control for manipulators with catadioptric camera. In *2008 IEEE International Conference on Robotics and Automation*, pages 510–515. IEEE, 2008.

[8] Alessandro De Luca, Massimo Ferri, Giuseppe Oriolo, and Paolo Robuffo Giordano. Visual servoing with exploitation of redundancy: An experimental study. In *2008 IEEE International Conference on Robotics and Automation*, pages 3231–3237. IEEE, 2008.

[9] AH Abdul Hafez, P Mithun, VV Anurag, Suril Vijaykumar Shah, and K Madhava Krishna. Reactionless visual servoing of a multi-arm space robot combined with other manipulation tasks. *Robotics and Autonomous Systems*, 91:1–10, 2017.

[10] Vincenzo Lippiello, Bruno Siciliano, and Luigi Villani. An occlusion prediction algorithm for visual servoing tasks in a multi-arm robotic cell. In *2005 International Symposium on Computational Intelligence in Robotics and Automation*, pages 733–738. IEEE, 2005.

[11] Vincenzo Lippiello. Real-time visual tracking based on bsp-tree representations of object boundary. *Robotica*, 23(3):365, 2005.

[12] Nicolás García-Aracil, Ezio Malis, Rafael Aracil-Santonja, and Carlos Pérez-Vidal. Continuous visual servoing despite the changes of visibility in image features. *IEEE Transactions on Robotics*, 21(6):1214–1220, 2005.

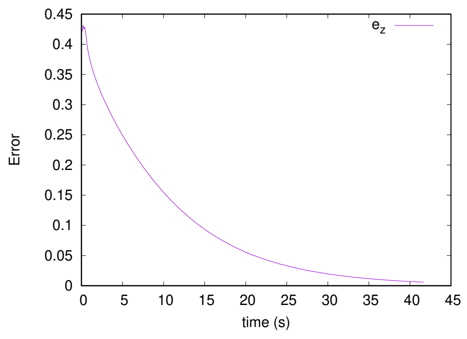
[13] François Chaumette, Seth Hutchinson, and Peter Corke. Visual servoing. In *Springer Handbook of Robotics*, pages 841–866. Springer, 2016.

[14] Gary R Bradski. Computer vision face tracking for use in a perceptual user interface. 1998.

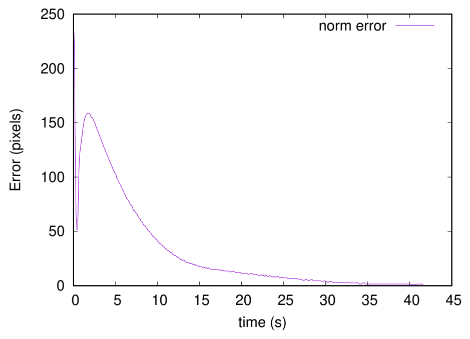
[15] Greg Welch, Gary Bishop, et al. An introduction to the kalman filter, 1995.

9 Appendix

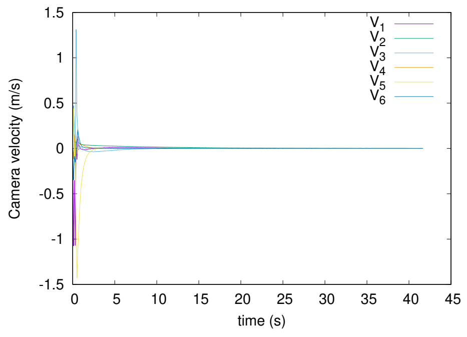
9.1 Tracking in 3D



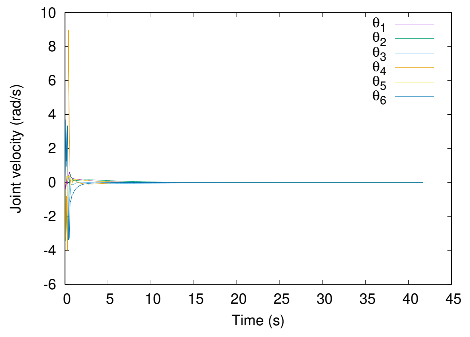
(a) Error in z axis



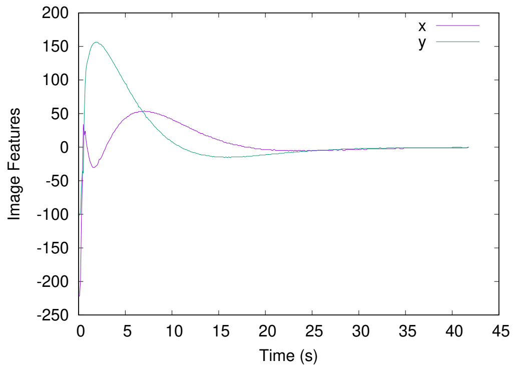
(b) Normalised Error



(c) Camera Velocity



(d) Joint Velocity



(e) Tracked Features

Figure 6: Tracking in 3D