École Centrale de Nantes

MASTER ARIA - ROBA
"AUTOMATIQUE, ROBOTIQUE ET INFORMATIQUE APPLIQUÉE"

2015/ 2016

Thesis Report

Presented by

Muhammad Qumar Zaman Tufail

On 30 / 08 / 2016

Title

Visual Servoing of a High Speed Parallel Robot

Jury

President:	Philippe Martinet	Professor, Ecole Centrale de Nantes
Evaluators:	Sebastien Briot	Full-time CNRS researcher at IRRCyN
	Philippe Martinet	Professor, ECN
	Olivier Kermorgant	Assistant Professor, ECN
	Abdelhamid Chriette	Assistant Professor, ECN

Supervisors:

Sebastien Briot and Olivier Kermorgant
Laboratory : Institut de Recherche en Communications et Cybernétique de Nantes

Abstract

This thesis work is aimed at investigating the possibilities of controlling a high speed parallel robot using computer vision techniques. For this purpose, different methods that have been proposed in the literature, are studied.Different possibilities like controlling by observation of leg directions and leg edges are considered here. The efficacy of these models with and without noise has been studied.

For the purposes of simulation, a model of the robot has been developed in ADAMS and linked with MATLAB/Simulink in order to control. At the end of the thesis work, reader can get an idea that the control of parallel robot using vision is indeed an innovative and useful way and the information provided can be used to further improve the results

Contents

List of Figures

Acronyms

POS Pose from orthography and scaling

POSIT POS with iterations

RRRP Revolute Revolute Revolute Prismatic joints

PID Proportional Integral Derivative

DOF Degrees of Freedom

MIMO Multi input Multi Output

ROI Regions of Interest

GS Gough Stewart

STL Stereolithography

IRRCyN Institut de Recherche en Communications et Cybernétique de Nantes

Chapter 1

Introduction

1.1 Structure of the report

This thesis work is divided into two parts. In part 1 ,basic concepts and techniques related to robotics,vision, visual servoing and control are presented. In part 2 , the work that has been performed during this project is presented. In third part, conclusion is presented and possible future work is suggested.

- In chapter 2 , basics of pose and velocity in the robotics literature are covered. How these parameters can be estimated using vision
- In chapter 3 ,state of the art in Robotics is described. Different robotics architectures and different control algorithms available in literature, are described briefly.
- In chapter 4 , the concepts of Vision based control are described. What is the basis of such kind of control . What kind of techniques are being used in literature to achieve vision based control, all this is covered in this chapter.
- Chapter 5 describes all the work that has been done in the context of this project. The results are presented and analyzed.
- Chapter 6 concludes the work done and describes the future work that could be done on the basis of this work

1.2 Objective

Aim is to propose a vision-based controller for IRSBot-2 , a high speed parallel robot designed at IRCCyN, based on the observation of the legs, rather than using the proprioceptive sensors.

To achieve this objective, during the first phase of this project, bibliographic study was performed in order to get acquainted with the work being done in this field and algorithms available in literature. In the second phase of the project, different vision based algorithms have been developed and investigated for the robot.

Part 1

Preliminary Concepts & Literature Review

Chapter 2

Pose and Velocity

Pose and velocity information about a body in space can help understand the motion of the body better. Pose of a body defines the complete position and orientation of a body in space. Velocity of a body defines how fast this pose is changing with respect to time. Depending on the physical resources available, there can be different methods to estimate these parameters. Nature has blessed different organisms with different kind of sensors to help them evolve and survive in their respective environments.
The animals evolving in the environments where light is not sufficient, tend to use other means. It is necessary for them to localise and estimate the motion of other animals, in order to survive and hunt. They usually use methods based on sound waves. Equipped with sensors that respond to sound waves, they send sound signals and then capture it back. From the captured signal, they are able to localise other animals, how far they are and their motion profile.
The animals evolving in suitable light conditions tend to depend more on the light based sensors,mainly vision. Human beings are also equipped with vision sensors,in the form of eyes. Our vision system is binocular, because of the difference of position of eyes on the head. The vision system of humans is a very efficient one,at the same time being a complicated one.

Another technology available for determining the pose and velocity of an object is RADAR technology. Radar is an object-detection system that uses radio waves to determine the range, angle, or velocity of objects. A radar transmits radio waves or microwaves that reflect from any object in their path. A receive radar, which is typically the same system as the transmit radar, receives and processes these reflected waves to determine properties of the object(s)[1]. It basically uses the Doppler effect.Based on the Doppler shifts,it can estimate the velocity of the objects. It is being used in many different applications in the world,such as:

- Air traffic Control and Air Defence
- Police Department
- Geology
- Weather prediction

To emulate any kind of system in real life, we need to have a thorough understanding of its working. Unfortunately, the vision system of humans is not understandable to a level of being able to completely emulate it. The issues of processing a huge amount of information while communicating between many different modules in real time is well performed by humans, but not well understood. The artificial correspondence to the natural system of eyes is camera. But there are many issues that need to be addressed while using the camera,such as:

- Limited Processing power
- Interfacing the camera with other processing units(Brain of the system)
- Operating frequency of the system
- Accuracy

These and many other issues related to artificial vision system are dealt with in the field of computer vision. The aim of this project is to extract the information related to the pose and velocity of an object in a scene, from the images provided by the camera,using the field of computer vision. Then we can use this information for our purpose (Designing a controller for the robot).

2.1 Pose in robotics literature

Since a robot uses its end-effector to manipulate different objects and interact with its environment, it is necessary to know the exact coordinates

of the end-effector. The position and orientation of it's *end-effector* is completely described by it's *pose*. The pose means coordinates of a specific point on the end-effector defined with respect to a frame attached to the robot's base. So we attach an imaginary frame to the end-effector, and we describe the coordinates of robot by describing the translation and rotation of this frame with respect to the reference frame.
For understanding purposes, we can assume two frames attached to two different bodies,named as R_G and R_H ,as shown in figure 2.1 . So now we need to find the transformation between these two frames,${}^{G}T_H$,that could define the position and orientation of one frame with respect to another frame.:

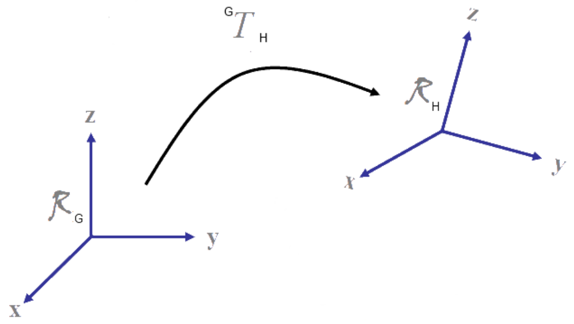


Figure 2.1: Transformation from frame G to H

This transformation matrix between two frames is called homogeneous transformation matrix. The general form of this matrix is given as:

$$ {}^{G}T_H = \begin{bmatrix} {}^{G}Rot_H & {}^{G}Pos_H \\ 000 & 1 \end{bmatrix} \tag{2.1} $$

In the above matrix ,${}^{G}Rot_H$ shows the orientation part,while ${}^{G}Pos_H$ shows the position part. Consider that we have a 3D point in space defined by the coordinates $\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}_{R_H}$ expressed in the frame R_H . If we want to express the same point but in frame R_G,we will have to write:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}_{R_G} = {}^{G}T_H \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}_{R_H} \tag{2.2}$$

The rotation part of the homogeneous transformation matrix can be defined in several ways. In literature,there exist many different parametrizations , like *Euler angles , direction cosines, quaternions,Bryant angles,Roll pitch yaw angles, Rodrigues'rotation formula* etc.

2.2 Pose estimation using vision

The most commonly used vision sensor is the camera. So if we are using camera to obtain visual features, it is obvious that we are dealing with 2D images. So the problem here is that we need to reconstruct the 3D world using the 2D information provided by camera. So we need to find a relation between 2D coordinates of points in images and 3D coordinates of same points in the world.
The most commonly used model for camera in literature is that of a pinhole camera. Pinhole camera model is based on the idea of perspective projection. Basic idea of perspective projection is that the images which are far appear smaller,while the images which are near appear larger,and the real image is inverted upside down. An illustration of perspective projection is provided in the figure 2.2:

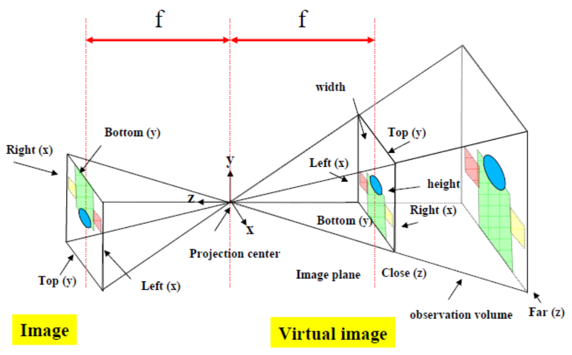


Figure 2.2: Perspective projection, taken from [10]

Using the idea of perspective projection, if we consider an object in the view of a pinhole camera,its real and virtual images can be visually depicted as shown in figure 2.3:

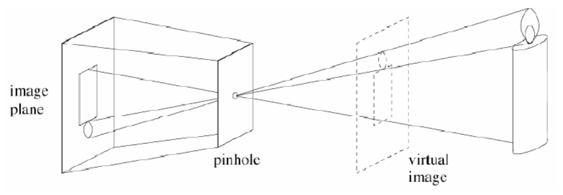


Figure 2.3: Pinhole camera with real and virtual images, taken from [10]

As can be seen in the figure 2.3, the camera aperture is described as a point,so lenses are not used to focus light in an ideal pinhole camera model. A geometric view of pinhole camera model can be represented as shown in figure 2.4:

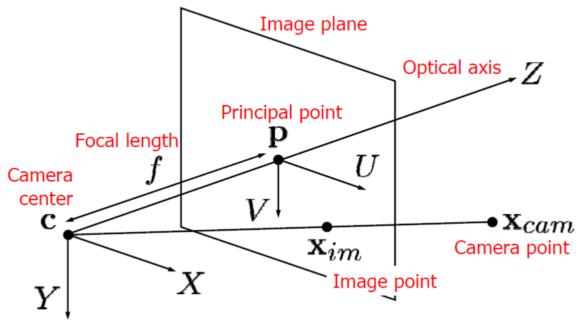


Figure 2.4: Geometric view of pinhole camera model, taken from [10]

In general form, the relationship between the image points in pixels and the 3D points can be written as:

$$\mathbf{w}.m_p = \mathbf{K}(I_{3\times3} \quad 0_{3\times3}) \begin{pmatrix} R_0 & t_0 \\ 000 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}_{F_0} \tag{2.3}$$

In the equation 2.3 , F_0 represents the object frame,or the frame in 3 D world and (X,Y,Z) are the coordinates of 3 D point in this frame. R_0 and t_0 represent the translation and rotation of the object frame with respect to the camera frame. They are called extrinsic parameters. I denotes the identity matrix. **K** is the collineation matrix, which is composed of the inertial parameters of the camera. **w** is the scaling factor ,which is inversely proportional to the depth of 3 D point.m_p are the coordinates of the point in the image plane, in pixels. The figure 2.5 provides a general overview of how we obtained the equation 2.3.

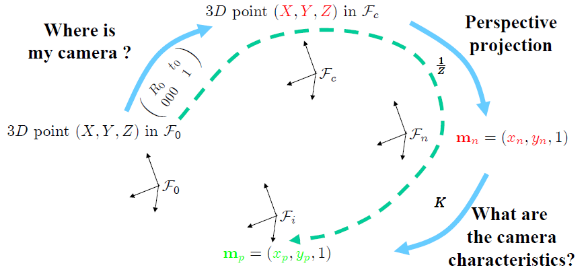

Where is my camera ?
$3D$ point (X,Y,Z) in $\mathcal{F}_c$
Perspective projection
R_0 t_0 000 1
$\mathcal{F}_c$
$\frac{1}{Z}$
$3D$ point (X,Y,Z) in $\mathcal{F}_0$
$\mathcal{F}_n$
$\mathbf{m}_n = (x_n, y_n, 1)$
$\mathcal{F}_0$
$\mathcal{F}_i$
K
What are the camera characteristics?
$\mathbf{m}_p = (x_p, y_p, 1)$

Figure 2.5: From 3D to image coordinates, taken from [10]

The collineation matrix **K** is a matrix of the following form:

$$K = \begin{bmatrix} f_u & \gamma & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

where:
f$_u$, f$_v$:Focal lengths in pixels
γ:Skew factor
u_0 and v_0 :Principal point

Camera calibration

If we define a matrix P, which is composed of intrinsic as well as extrinsic parameters,we can write:

$$P = \mathbf{K}(I_{3\times3} \quad 0_{3\times3}) \begin{pmatrix} R_0 & t_0 \\ 000 & 1 \end{pmatrix} \tag{2.4}$$

So equation 2.3 can be written as:

$$w.m_{pi} = P.m_i \tag{2.5}$$

where $m_{pi} = \begin{bmatrix} x_{pi} \\ y_{pi} \\ 1 \end{bmatrix}$ is the image point of i-th point. $m_i = \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}$ represents the 3 D coordinates of i-th point of the object. So P will be a 3×4 matrix. Hence we need to estimate 12 unknowns,which means that by estmating P matrix, we can extract the matrix $\mathbf{K}$, R_0 and t_0. So from equation 2.5,we can estimate P , taking number of images n$\geq$ 6 ,knowing the model of object m_i and image coordinates m_{pi}.
This was the linear approach for camera calibration. There also exists a non-linear approach. The basic idea behind non-linear approach is that it initializes from the estimate provided by linear approach. Then a correction is computed at each iteration until the convergence of the system. Convergence of the system is defined by minimization of a chosen criteria.

Distortion

Generally, there exists some distortion when we are considering pinhole camera model. So this also needs to be taken into account. For this purpose, two kinds of distortions are defined [10]:

- Radial distortion
- Tangential distortion

Radial distortion results because of the failure of a lens to image straight lines as straight lines. It is composed of three components,which can be denoted as k_1,k_2 and k_3. Tangential distortion is produced when a lens is not parallel to the imaging plane. It is composed of two components,which can be denoted by p_1 and p_2. In this case, the equation for pinhole camera model will be modified to take into account the distortions. The equation is not detailed here. Interested reader can find it in [10].

2.2.1 Estimating Pose

With a calibrated camera, the pose can be estimated using different algorithms available in literature. The most commonly used algorithm in case of non-planar objects is Dementhon algorithm [11] . The basic idea of Dementhon algorithm is this :
The algorithm is defined to estimate the pose of a non coplanar object from a single object. The model of object must be known. Four or more non-coplanar feature points of the object are used. In this method, basically two

algorithms algorithms are used, namely POS(Pose from orthography and scaling) and POSIT(POS with iterations) .
POS algorithm considers a scaled orthographic projection and finds the rotation matrix and translation vector by solving a linear system. Scaled orthographic projection means that we assume that all the points on the object have same depth and we replace Z_i for each point by just a value Z . In POSIT algorithm, first an estimation of pose is obtained using the POS algorithm. Then in the iteration loop, applies POS to scaled orthographic projections of feature points ,rather than to original image projections. POSIT converges to good accuracy within few iterations. As shown in [11] , POSIT can be written in 25 lines of code in Mathematica. Dementhon algorithm was used in [2] by Paccot et al. in order to estimate the pose. This method is not feasible in our case, because the required operating frequency of robot is quite high . So other ways to estimate pose will be explored in next chapters.

2.2.2 Velocity estimation

A precise estimate of velocity along with pose information can help us to understand the motion better and design a more efficient controller in terms of twist associated.In literature, even though efficient algorithms exist for pose estimation, but those algorithms do not provide a way to estimate velocities. In many research works, like [2], velocity estimation is produced by direct numerical differentiation of pose information. THis introduces noise,which at high speeds could become very high. Hence this way of computing velocity information is not suitable for our cause and alternate methods will be explored in order to obtain it in a more useful way.

Chapter 3

State of the art in Robotics

3.1 Concept of a robot

A robot can be described as a mechanical assembly of different parts whose combined action allows it to perform motions with a certain degree of autonomy. It is a programmable device and the region it can operate in, is defined by its workspace. It is different from automated machines in the fact that it does not necessarily perform same motion again and again and it can be programmed to do different tasks.
The different segments or links in a robot are connected to each other with the help of *joints* .These joints could be of many types,namely,prismatic,revolute,cylindrical,spherical,universal etc. The type of joints determine the kind of motion that could be performed. The combined effect of these series of segments helps the robot to place and manoeuvre it's end-effector in the environment.

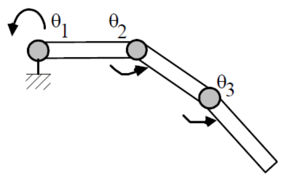
While giving us many advantages over automated machines, robots are limited in their operation by different constraints imposed by physical limits of different parts being used. This fact makes the design and control of robots an interesting research and engineering field. While a good design and control algorithm can provide the industry with a very useful tool, the constraints and limits imposed could also pose a great challenge.
In the following sections of this chapter, a general classification of robots will be described,that is currently being used in literature. Also,different constraints imposed by physical parts will be described and different control strategies being used will be briefly covered.

3.2 Classification of robots

Robots can be generally classified on the basis of types of joints,number of joints, types of motions that it can perform etc. Here we will describe the two general classifications made in literature,namely *serial* and *parallel* robots. These classifications are made on the basis of the way a robot's end-effector is connected to it's base. We will see that both classes have quite different properties and constraints. This fact leads us to adopt different control strategies for both classes.

3.2.1 Serial robotic architectures

In serial robots, the end-effector of the robot is attached to it's base with the help of several links,connected to each other in a serial way. This architecture is analogous to serial circuits in electrical engineering. A close resemblance to serial architecture can be thought of a human arm,where the links are connected in a serial way. A serial architecture is shown in the figure 3.1a and a serial robot in 3.1b :



(a) Serial architecture

(b) Serial Robot

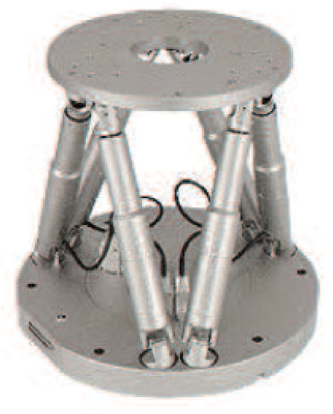
Figure 3.1: Visual description of serial robots, taken from [4]

Serial robots are widely used in the industry. The main reason of their widespread use is that they have a large workspace and relatively simple structure.
However, serial robots have their own disadvantages. Because of their serial architecture , each actuator has to carry the inertia of it's link as well as inertia of the following links and actuators. This leads to a very high inertia at the last link and hence relatively small load can be manipulated by the

robot. This means that serial robots have high less payload to mass ratio as compare to their parallel counterparts. Also, the errors tend to accumulate in serial robots because of their architecture. This leads to a reduced accuracy. These facts urge us to look for alternative solutions and hence the role of parallel robots becomes very important.

3.2.2 Parallel Robotic architectures

As opposed to serial robots, the end-effector and base of the robot are connected by several kinematic chains in parallel robots. Each kinematic chain is composed of a number of serial links connected by joints, just like serial robots. All these kinematic chains combine to give a parallel robot. Each of these kinematic chains are called *legs* in the architecture and they form a close loop. Unlike serial robots,where usually all of the joints are actuated, parallel robots have some motorized joints(active joints) and some non-motorized joints(passive joints) in each leg . A parallel architecture is shown in figure and a parallel robot in figure :


(a) Stewart platform

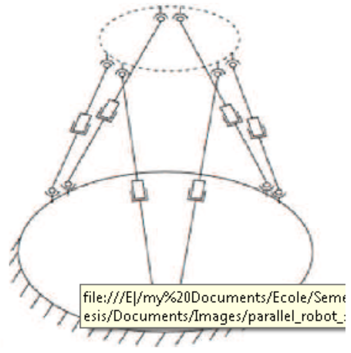

file:///E|/my%20Documents/Ecole/Seme
esis/Documents/Images/parallel_robot_
(b) Kinematic scheme

Figure 3.2: Visual description of Parallel robots, taken from [5]

Advantages

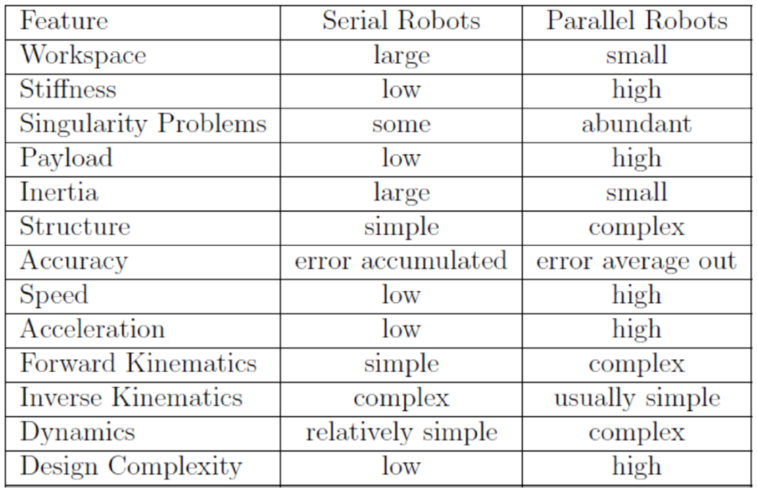
Parallel robots have several advantages as well as drawbacks,when compared to serial robots. Since motion of end-effector is a result of the combined

motion of several light weight kinematic chains, parallel robots are able to handle weights more than their weights so their payload to mass ratio is higher. Actuators can be mounted onto the base and this results in faster movements and light weight kinematic chains.
Another advantage of parallel robots is that the errors tend to average out as opposed to the serial case,where they tend to accumulate. Also,because of several supporting chains, the stiffness of the structure is high.

Drawbacks

However, parallel robots are not without their disadvantages. The workspace of parallel robots is smaller as compared to serial robots. Another disadvantage is that their workspace is constrained by many kind of singularities which leads to a complex design analysis and hence a complex structure. At a singularity, the robot loses one or more degrees of freedom or the motion in a particular direction becomes uncontrollable. Hence singularity analysis is a major challenge for the designer and effectiveness of a robot can be described in terms of singularities in its workspace.

Another drawback is that the forward kinematic model of the parallel robots is usually complex and does not have an analytical form. Hence we have to resort to numerical methods which can introduce errors in the modelling and these small errors can lead to major problems in model based control algorithms. Hence the problem of control of parallel manipulators is a research issue and will be briefly covered in the following sections.
We can see a non-exhaustive comparison of serial and parallel robots in figure3.3 :



Feature	Serial Robots	Parallel Robots
Workspace	large	small
Stiffness	low	high
Singularity Problems	some	abundant
Payload	low	high
Inertia	large	small
Structure	simple	complex
Accuracy	error accumulated	error average out
Speed	low	high
Acceleration	low	high
Forward Kinematics	simple	complex
Inverse Kinematics	complex	usually simple
Dynamics	relatively simple	complex
Design Complexity	low	high

Figure 3.3: Comparison of serial and parallel robots,taken from [4]

Singularities in Parallel Robots

Singularities in parallel robots usually arise from the degeneracy of kinematic model of the robot. If we represent the pose of the robot by x active joint variables by q_a,then the corresponding velocities are related by following mathematical equation:

$$A\dot{x} + B\dot{q}_a = 0$$

Where:
$\dot{x}$ is twist of the robot
$\dot{q}_a$are velocities of active joint variables
A is reduced form of parallel jacobian matrix which contains actuation and constraint wrenches
B is serial jacobian matrix
This form of equation is also called first-order kinematic model of the robot. **Serial** singularities correspond to the situation when the matrix B degenerates. Physically,it means that the robot is at the boundary of it's workspace or an internal boundary limiting different subregions of the workspace. This kind of singularity is also called **Type** 1 **singularity**. In this situation, the output link loses one or more degrees of freedom. This situation is depicted in figure 3.4:

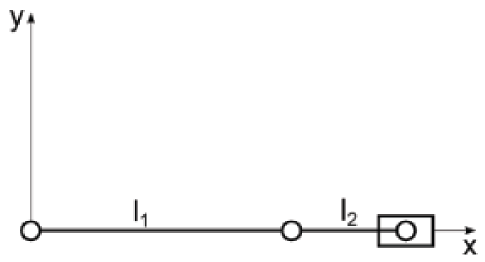


Figure 3.4: Planar RRRP mechanism in type1 singualrity,taken from [5]

Type 2 singularities arise when the matrix A degenerates. This corresponds to a situation when output link gains one or more degrees of freedom or more precisely, one or more degrees of freedom become uncontrollable. This situation is depicted in figure 3.5:

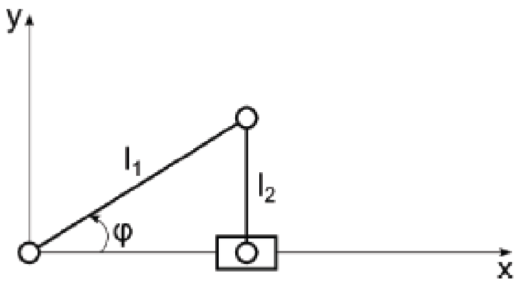


Figure 3.5: Planar RRRP mechanism in type2 singualrity ,taken from [5]

Here R means revolute joint,which can only produce rotational motion,while P means prismatic joint,which can only produce translational motion.

Another type of singularities exist,which are called **constraint singularities**. These singularities occur when a system of leg constraints degenerates. This means that the platform gains an additional degree of freedom, outside of it's number of degrees of freedom. There exist some other kind of singularites in parallel robots, which are less common and will not be detailed here.
This general overview about the singularities in parallel robots gives us an idea about how many problems we can face in design and analysis of a parallel robot.

3.3 Classical Control algorithms

3.3.1 PID Control

PID control corresponds to proportional,integral,derivative control. It means the control signal is composed of three terms, a proportional term,a derivative term and an integral term. PID controllers are the most basic kind of controllers and are very useful when the systems are linear. The control signal corresponds to terms composed of difference between the feedback term and desired value. A basic mathematical equation of a PID controller can be written in the following form:

$$\tau = K_p(q^d - q) + K_d(\dot{q}^d - \dot{q}) + k_i \int (q^d - q)dt \tag{3.1}$$

The schematic of PID controller is shown in figure 3.6:

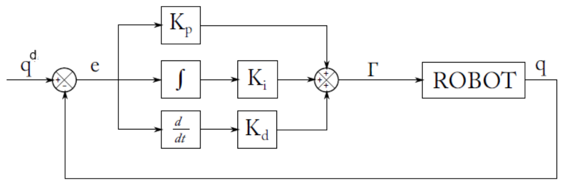


Figure 3.6: PID Controller schematic

Here **q** is the feedback signal from sensor, q^d is the desired value and K_p, K_d, K_i are tuning parameters and there exist different tuning techniques in literature. Interested readers about tuning of these parameters are referred to [6] .
As for the case of parallel robots, the dynamics of different legs are coupled which lead to a non linear behaviour of the system. An assumption of linear behaviour can be applied at very low velocities ,but in our case, we want to work at very high velocities and hence this assumption is no more valid. Hence this class of controllers are not useful in our case of work. So we will explore other methods available.

3.3.2 Computed Torque Control

This method is based on input-output linearization using the state feedback of a non-linear system. It is a model based approach,which means it is susceptible to modelling errors.

Considering the general form of dynamic model of a robot,given by equation below:

$$\tau = A(q)\ddot{q} + H(q, \dot{q}) \tag{3.2}$$

Where:
τ is the torque
A(q) denotes the Inertia matrix of the robot
q is the set of joint variables
H denotes the matrix taking into account the gravitational,Coriolis and centrifugal effects
Considering the torque control law described by:

$$\tau = \hat{A(q)}w(t) + \hat{H}(q, \dot{q}) \tag{3.3}$$

So a straightforward linearization is obtained by having $\ddot{q} = w(t)$. This means that the system has become a double integrator. We can consider different cases ,as described in the following sections.

Motion completely specified

In the case,motion is completely specified, means we know the desired joint positions, velocities and accelerations, we can write the control law as:

$$w(t) = \ddot{q}^d + K_v(\dot{q}^d - \dot{q}) + K_p(q^d - q) \tag{3.4}$$

Then:

$$\ddot{q} = \ddot{q}^d + K_v(\dot{q}^d - \dot{q}) + K_p(q^d - q)$$

If we write error as:

$$e = (q^d - q)$$

Then ,using equation 3.3 ,the control signal becomes:

$$\tau = A(\ddot{q}^d + K_v\dot{e} + K_pe) + \hat{H}(q, \dot{q}) \tag{3.5}$$

where:
K_v is derivative gain tuning term
K_p is proportional gain tuning term

Only desired position specified

In this case, we can write the control signal as:

$$w(t) = K_p(q^d - q) - K_v\dot{q} \tag{3.6}$$

Then using equation 3.3, we can write the control law as:

$$\tau = \hat{A}(K_p e - K_v\dot{q}) + \hat{H}(q, \dot{q}) \tag{3.7}$$

The control scheme in this case is shown in figure 3.7 :

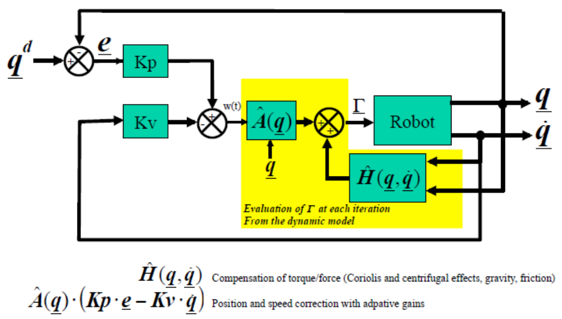

$\underline{q}^d$
$\underline{e}$
Kp
Kv
w(t)
$\hat{A}(\underline{q})$
$\underline{q}$
$\underline{\Gamma}$
Robot
$\underline{q}$
$\dot{\underline{q}}$
$\hat{H}(\underline{q},\dot{\underline{q}})$
Evaluation of Γ at each iteration
From the dynamic model
$\hat{H}(\underline{q},\dot{\underline{q}})$ Compensation of torque/force (Coriolis and centrifugal effects, gravity, friction)
$\hat{A}(\underline{q})\cdot(Kp\cdot\underline{e} - Kv\cdot\dot{\underline{q}})$ Position and speed correction with adpative gains

Figure 3.7: Computed Torque Controller schematic,taken from[7]

The computed torque control method gives a nice compensation for dynamics of the robot. It is also quite accurate,given an accurate model. However, because of its dependence on model of the system, modelling errors could cause problems and hence lead to unsatisfactory results.
Since we are dealing with parallel robots, it is valid to question whether we should use joint space control or cartesian space control. This issue will be deal with in the next part.

3.4 Joint space or Cartesian space

In the case of serial robots, we know that joint configuration of the robot completely describes the configuration of end-effector. In other words, we have analytical expression for forward geometric models of the robot . It means we can control in joint space and this will be the state space.
However,in case of parallel robots, the joint configuration does not completely specify the end-effector configuration. This is because for a given joint configuration, there can be many end-effector configurations which

correspond to that same joint configuration. For example, in case of Gough-Stewart platform, a single joint configuration may correspond to 40 real configurations. So in the case of parallel robots, it is desirable to control in cartesian space. This was shown in [8] . Also the superiority of computed torque control law was shown in this paper.
However,as shown previously,computed torque control law is a model based approach and hence it is susceptible to errors. In the next chapter, control techniques which do not depend on models,will be explored.

Chapter 4

Vision Based Control

The classical control techniques described previously, all depend on model of the system. It means that the model will estimate the relationship between joint configurations and corresponding end-effector configuration. So in order to get accurate results,model of the system has to be as accurate as possible. This accuracy in modelling is usually achieved through identification process,which is quite rigorous. Implementing identification on complex systems is quite time consuming and expensive. Hence an alternate solution needs to be found. One way is to totally bypass the model. But how could this be achieved? The answer lies in the sections ahead.
In order to totally bypass the effects of modelling errors, we could think of schemes which do not require the use of models. It means we can control in the cartesian space, without having to use the forward geometric models. This could be achieved by the use of an exteroceptive sensor,which could provide the measure of the configuration of end-effector. One of the possible candidates in this case,could be to use vision as a sensor. A general scheme in the case of sensor-base control scheme could be as shown in figure 4.1:

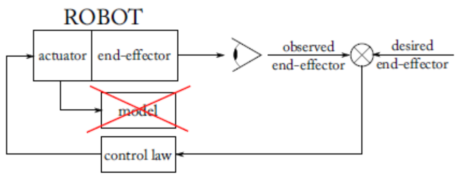


Figure 4.1: General schematic for sensor based control,taken from[5]

4.1 Basics of Vision based control

The basic idea in vision based control is to minimize an error defined by the following equation [9]:

$$e(t) = s(m(t), a) - s^* \tag{4.1}$$

Here s denotes the set of visual features that we are interested in. It depends on the set of image measurements,**m**, as well as the intrinsic parameters or the knowledge about the object model **a**.s^* represents the desired visual features
In vision based control,usually a velocity controller is designed based on the concept of interaction matrix An interaction matrix relates the rate of change of visual features with the instantaneous relative motion between the camera and scene. If we denote the interaction matrix by L_s , then we can write:

$$\dot{s} = L_s V_c \tag{4.2}$$

Here V_c represents the instantaneous relative motion between the camera and scene. So it can be defined as $V_c =^c v_c - {}^c v_s$. Here${}^c v_c$ and ${}^c v_s$ represent the kinematic screw of the camera and the scene respectively, both represented in camera frame.
Then if we want to impose exponential decay of the error,we can write $\dot{e} = -\lambda e$. If we assume that desired visual features are not changing with respect to time, then we can define a velocity controller given by the following equation:

$$V_c = -\lambda \, L_s^+ (s(m(t), a) - s^*) \tag{4.3}$$

4.2 Different techniques in visual servoing

4.2.1 Image based visual servoing

The basic idea in image based visual servoing is to achieve servoing in the image. We do not know how the robot will move in 3 D space, we try to achieve convergence in image space. For this purpose, a relation between 2D points in the image in pixel coordinates and velocity of camera must be defined. This relationship is defined by an interaction matrix. If we represent image point coordinates in pixels as m_p ,from [17]we can write:

$$\dot{m}_p = \begin{bmatrix} -f_u/Z & 0 & u/Z & (u*v)/f_v & -f_u - u^2/f_u & (-f_u*v)/f_v \\ \\ 0 & -f_v/Z & v/Z & f_v + v^2/f_v & -(u*v)/f_u & (-f_v*u)/f_u \end{bmatrix} \quad V_c$$

Where:
V_c :the camera velocity vector ,which is composed of 6 components, 3 for translational velocity and 3 for rotational velocity
u,v: image point coordinates in pixels
Z is the depth of the point
Here, the skew factor "l" is assumed to be equal to zero . $f_v = fr$ and $f_u = f$, where "f" denotes focal length and "r" denotes aspect ratio.
This was 2D point based visual servoing. There exist many other 2 D techniques which differ on the basis of features considered. To name a few, we can consider 2D segment features, 2D moments etc.

4.2.2 Position based visual servoing

In this case,we know how the robot will behave in 3D but we do not know what will happen in the image. In this case, the set of visual features"s" is defined on the basis of the pose of the camera with respect to some reference frame. So in this case, knowledge of the model of object is essential. We can then write $s = (\mathbf{t}, \mathbf{y}\theta)$. Here "$\mathbf{t}$" is the translation vector,while $\mathbf{y}\theta$ represents the rotation of θ around an axis defined by unit vector $\mathbf{y}$. Now we can use two different approaches based on how the translation vector is defined.
If we define $\mathbf{t}$ relative to an object frame F_o, we can write the interaction matrix as:

$$L_e = \begin{bmatrix} -I_3 & [{}^c\mathbf{t}_o]_\times \\ 0_3 & L_{\mathbf{y}\theta} \end{bmatrix}$$

where $L_{\mathbf{y}\theta}$ is defined as:

$$L_{\mathbf{y}\theta} = I_3 - \frac{\theta}{2}[\mathbf{y}]_\times + (1 - \frac{sinc\theta}{sinc^2\frac{\theta}{2}})\,[\mathbf{y}]^2_\times$$

Where: I_3 is 3×3 identity matrix while 0_3 is 3×3 zero matrix
$[\mathbf{y}]_\times$ is the skew symmetric matrix associated with the unit vector
$[{}^c\mathbf{t}_o]_\times$ is the skew-symmetric matrix associated with translation vector.
sinc$\theta = \frac{\sin\theta}{\theta}$
Note that here we could define the rotation part as $\mathbf{y}\sin\theta$ or $\mathbf{y}\frac{\theta}{2}$. But

those parametrizations have singularities,while $\mathbf{y}\theta$ parametrization has no singularity.

4.3 Vision based computed torque control of parallel kinematic machines[2]

Usually, parallel robots are controlled in cartesian space. This is because of the fact that joint space is not the state space in case of parallel robots. Many end-effector poses exist corresponding to a specific joint configuration. Hence Parallel robots have been shown to be better controlled in cartesian space. This paper introduced a novel approach in the control of parallel kinematic machines. Instead of writing the models of the robot in the form of joint configurations, the models were written only as a function of end-effector configurations. Having achieved this, the control loop used measures from a fast exteroceptive sensor and hence it reduced the complexity involved. So in this way, the problem of solving the forward kinematic model is avoided completely. The basic control scheme used in this paper is as shown below:

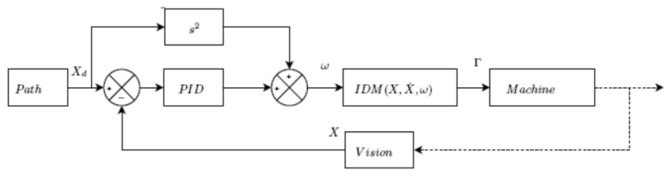


Figure 4.2: Cartesian space computed torque control,where $\omega = \ddot{X}$[2]

This was one of the first papers to practically implement the idea of using vision in dynamic control of parallel kinematic machines. The results obtained were tested on Orthoglide robot. It was shown practically that the results obtained were comparable to those obtained with computed torque control based on forward kinematic model. The following figure was obtained for a trajectory of $60mm$ circle :

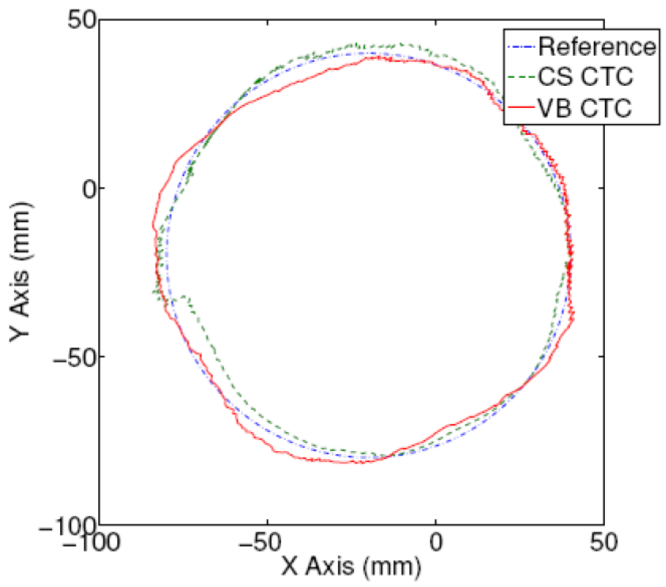


Figure 4.3: $60mm$ circle at $3ms^{-2}$ achieved by the Cartesian space computed torque control with the forward kinematic model and the vision-based computed torque control in the XY plane

We can see that the results obtained using visual sensor are comparable. But this method is not useful for our purposes. In this method, Dementhon algorithm is used for computing the pose and then velocity is derived by simple numerical derivation. Since we are interested in working at high frequencies, the numerical derivation is not a useful option in our case, hence this method will not be useful. Nevertheless, it encourages the use of vision as sensor and with more precise visual sensors and at lower velocities, this method can provide useful results.

4.4 Problems with vision based control

Since we have clearly established the advantages associated with using vision based control, we are hopeful to get an improvement in results. The problem in vision based control comes when we look into the practical implementation of these techniques. We see that the dynamic control frequency is usually quite high,in our case we want to operate the robot at 1 kHz . The problem is that we have to synchronize the vision sensor with the robot. The vision sensors usually operate at lower frequencies. Even with high technology cameras, the problem of huge amount of data to be transmitted and processed cannot be ignored. Moreover,the transferring system will have limited bandwidth. All this results in loss of speed of vision based controller. One solution is proposed in [12] , where the authors designed a multivariable predictive controller for a 6 DOF manipulator robot. The next section will describe this controller briefly.

4.5 MIMO predictive controller

MIMO stands for multi-input,multi-output .This work is based on the linearized model of the dynamics of a robot. The purpose of this controller was to increase the bandwidth of servo loop.In the visual servo loop, the reference pose vector is compared with estimated pose obtained using the vision system.The controller is a velocity controller, means that it generates a control signal that is a velocity vector of size 6×1 . It represents a velocity reference signal for each component of pose. The open loop model of visual servoing can be considered as a multi-input,multi-output (MIMO) system, whose input is a reference velocity vector and whose output is an estimated vector of pose, having 6 components. The controller is predictive controller because it takes into account the future references.
It was shown in this paper, with the help of practical implementations, that predctive controller always yields a larger bandwidth as compared to PID. However,In this work, the authors did not consider the torque control. In our case, while dealing with higher dynamics, we have to use the torque control . So this type of approach is not valid for torque control,because in that case, the model of the robot cannot be linearized.

4.6 Techniques based on Regions of Interest acquisition

4.6.1 3D Pose and Velocity Visual Tracking Based on Sequential Region of Interest Acquisition[3]

This method is based on non-simultaneous sub-images acquisition. Instead of grabbing the whole image and using it for the purpose of visual servoing, this paper introduces the concept of regions of interest. Sequential acquisition of regions of interest(ROI) means that we capture only the parts of the image which have useful information. This method has many benefits as compared to classical visual servoing schemes, in terms of visual control sampling frequency. It allows to increase the visual control sampling frequency as well as manages to reduce the amount of data to be acquired and sent by camera. Also ,the associated image projection model depends on pose and velocity of observed object. Based on this property, a new control law was defined in this paper, whose outputs are kinematic and dynamic twists.

A very important assumption taken by the authors in this paper was that the velocity of the tracked object is constant during the image acquisition. In other words, the velocity was considered piecewise constant ,and image acquisition period was considered small enough to make this assumption valid. Practically, the algorithm was implemented using two threads,as shown in figure 4.4. One thread was dedicated to ROI acquisition and another thread was dedicated to control processing and pose prediction. This idea allowed to achieve 4kHz ROI acquisition frequency and 400Hz vision control sampling frequency.

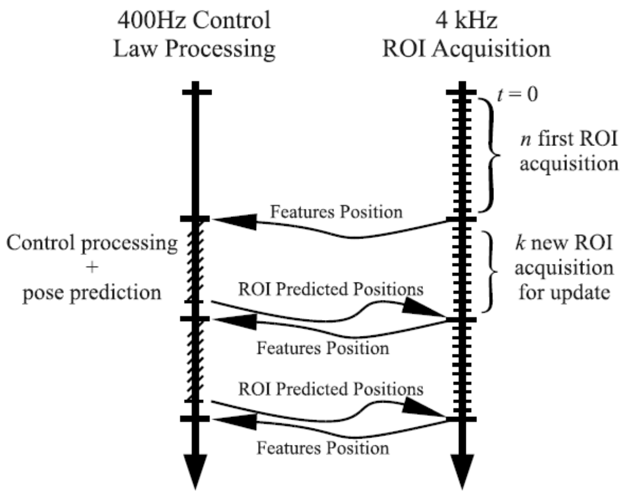


Figure 4.4: High speed vision system chronogram where processing is based on the sequential acquisition of small sub-images containing the features.

Proposed visual servoing approach

We consider a camera and a rigid known object in its visual field. The object can be represented by a set of 3D points. The motion of this object can be analysed by sequentially grabbing one single sub-image where just one point is located. The cartesian coordinates of the set of points with respect to the object frame is represented as ${}^{o}P_i, \forall i = 1, ..., n$. Their corresponding image projections in the camera frame are denoted by $m_i = (u_i, v_i)^T$ .If we use the pinhole projection model, then we can compute the coordinates of these points in camera frame as:

$$\forall i = 1, ..., n, w_i \tilde{m}_i(t_i) = K({}^{c}R_{oi} \quad {}^{c}t_{oi}){}^{o}\tilde{P}_i \tag{4.4}$$

where:
${}^{c}R_{oi}$ is rotation between the reference and grabbing times
${}^{c}t_{oi}$ is translation between the reference and grabbing times
K is the intrinsic parameters matrix of camera
${}^{o}\tilde{P}_i = (P_i^T, 1)^T$ is the homogeneous representation of ${}^{o}P_i$
$\tilde{m_i} = (m_i^T, 1)^T$ is the homogeneous representation of m_i
w_i is the scale factor ,which is inverse of the depth of 3D point
Now considering the assumption of constant velocity between the image

acquisitions, we can obtain the translation of the object by simple integration:

$$^{c}\delta t = \int_{t0}^{ti} {}^{c}V dt =^{c} V \Delta t_i \tag{4.5}$$

The rotation can be defined using Rodrigues formula,expressed in the form of rotational velocities:

$$^{o}\delta R_i = I + \frac{sin(\| {}^{o}\omega\| \Delta t_i)}{\| {}^{o}\omega\|}[^{o}\omega]_{\times} + \frac{1 - cos^2(\| {}^{o}\omega\| \Delta t_i)}{\| {}^{o}\omega\|^2}[^{o}\omega]_{\times}^2 \tag{4.6}$$

where:
$\| {}^{o}\omega\|$ represents the magnitude of rotational velocity $^{o}\omega$
$[^{o}\omega]_{\times}$ represents the skew symmetric matrix associated with the rotational velocity vector
Δt_i is the integration time, or the time elapsed between each acquisition
In order to define the aim of this method, a corresponding task function was defined. A task function of the general form was defined as below:

$$e = C(s(r, \tau_0) - s^*(t)) \tag{4.7}$$

where $s(r, \tau_0)$ is the vector of features in the image plane. It depends on the object pose 'r' as well as the kinematic twist τ_0(translational and rotational velocities). C is the combination matrix. We should note here that task function defined here will have 12 entries, 6 for pose and 6 for velocity. Combination matrix will correspondingly be a 12x2n matrix, where n is the number of features. As shown in [3] , we can take the time derivative of equation 4.7,and write it as:

$$\dot{e} = \frac{\partial e}{\partial r}\frac{dr}{dt} + \frac{\partial e}{\partial \tau_0}\frac{d\tau_0}{dt} + \frac{\partial e}{\partial t} \tag{4.8}$$

After some simplifications ,equation 4.8 can be written as:

$$\dot{e} = CL_{2d}\begin{bmatrix}\tau \\ \dot{\tau}\end{bmatrix} - C\frac{ds^*(t)}{dt} \tag{4.9}$$

Here L_{2d} is an interaction matrix which will be of size 2nx12 . It will relate the velocities of n image features to translational and rotational velocities and acceleration of the object. It should be noted here that in classical vision based algorithms, size of interaction matrix is usually 2nx6 because in that case we only relate translational and rotational velocities to the features. Here, we also have a relation of rotational and translational accelerations. A first order exponential decrease of the task function was imposed,which could be mathematically expressed as:

$$\dot{e} = -\lambda e \tag{4.10}$$

where λ was defined as a positive proportional gain parameter which will determine the convergence speed of this law.
From equations 4.9 and 4.10, we can write :

$$\begin{bmatrix} \tau \\ \dot{\tau} \end{bmatrix} = (CL_{2d})^{-1}(-\lambda e + C\dot{s}^*(t)) \tag{4.11}$$

The condition for convergence of this control law can be written as:

$$C = \hat{L_{2d}}^+$$

where $\hat{L_{2d}}^+$ is the pseudo inverse of the estimate of interaction matrix L_{2d}. If we assume that estimate of interaction matrix is correct i.e,:

$$\hat{L_{2d}}^+ L_{2d} = I$$

where I is the identity matrix, then the control law can be written in the following form:

$$\begin{bmatrix} \tau \\ \dot{\tau} \end{bmatrix} = -\lambda \hat{L_{2d}}^+ (s(r, \tau_0) - s^*(t)) + \hat{L_{2d}}^+ s^*(t)) \tag{4.12}$$

The proposed control scheme based on ROI acquisition is as shown in figure4.5:

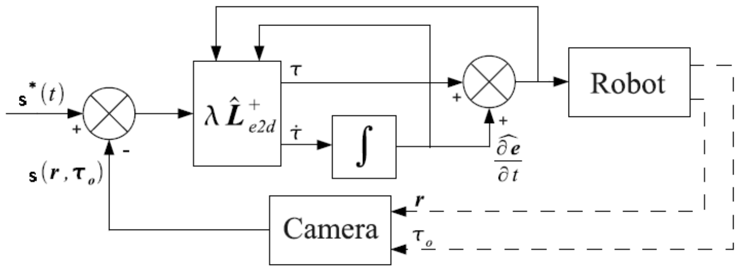
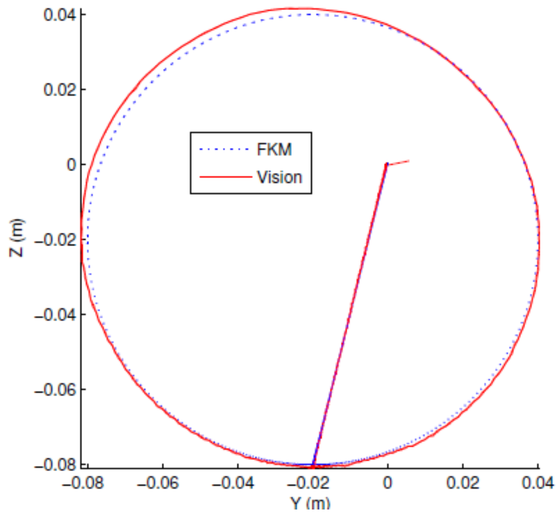


Figure 4.5: Control scheme using ROI acquisition

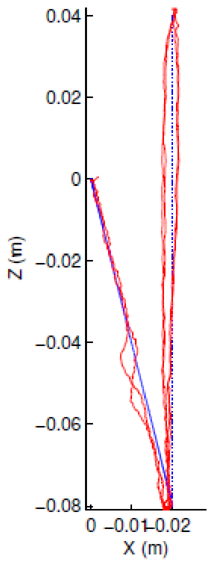
In this control method, the control output provided 12 elements, 6 for kinematic twise and 6 for dynamic twist. So integration of kinematic twist provides the estimation of current pose, and integration of dynamic twist provides an estimation of current velocity. The relative target velocity with respect to the camera was then estimated by integrating the dynamic twist over the sampling period .
Interaction matrix is computed on the basic principles of vision based control described previously. It is detailed in [3] .

Results

The virtual visual servoing scheme was implemented by authors in [3] in C++ language . The acquisition process was performed using a "Photon focus Track Cam" based on ROI grabbing method. The algorithm was tested on "Orthoglide" robot. Maximum speed reached was $1m/s$ and maximum tangential acceleration of the robot was $5m/s^2$. The results obtained were as shown in the figure 4.6:



(a) Trajectories projection on XZ plane



(b) Trajectories projection on XZ plane

Figure 4.6: 3D trajectories from robot joint sensors and Vision, taken from [3]

We can see that the results provided by vision sensor are quite comparable to those obtained using the model of system. With better vision sensors ,this response could be improved.

4.6.2 Efficient High-speed Vision-based Computed Torque Control of the Orthoglide Parallel Robot [19]

In [19] , authors have improved the work done in [3] . The basic assumption in [3] was that velocity is constant during the image acquisition time. Because of this assumption,the velocity was estimated with a constant delay. This assumption is not quite valid when considering very high speed parallel robot. So in [19], authors have made a more coherent assumption of constant acceleration. The basic idea is same as the one used in [3] . The authors showed that the results obtained were improved and were superior to the ones obtained with model based approach.

Part 2

Chapter 5

Visual servoing using legs observation

We have seen in the previous chapters that classically, computed torque control law is used to control parallel robots. This control law performs well in joint space,in case of serial robots. However,in case of parallel robots, we usually try to avoid working in joint space because one has to solve for forward kinematic problem at each step.
So in case of parallel robots, we try to work in cartesian space,as was shown in previous chapter. One way to control in cartesian space is to estimate the end-effector pose and estimate its velocity directly,using different pose estimation algorithms available in literature. But these pose estimation algorithms are costly in terms of computations and hence not very feasible in very high speed manipulators.

5.1 Plücker coordinates for visual servoing

Plücker coordinates are another way to represent a line in space. We know that a line L in 3-dimesnional Euclidean space can be determined by two distinct points. If we consider $\mathbf{x}_1 = (x_1, y_1, z_1)$and $\mathbf{x}_2 = (x_2, y_2, z_2)$ be the two points on line L, then the vector displacement from $\mathbf{x}_1$ to $\mathbf{x}_2$ represents the direction of the line. by this definition, every displacement between points on L is a scalar multiple of $\mathbf{d} = \mathbf{x}_2 - \mathbf{x}_1$. If we suppose that a physical particle of unit mass moves from $\mathbf{x}_1$ to $\mathbf{x}_2$, it would have a moment about origin. The geometric equivalent is a vector with the direction perpendicular to the plane containing line L and the origin, with the length equal to double the

area of triangle formed by the segment of displacement and origin. Therefore, moment is the vector cross product $\mathbf{m} = \mathbf{P} \times \mathbf{d}$, where $\mathbf{P}$ is any point on the line. The area of triangle is proportional to the length of segment between $\mathbf{x}_1$ and $\mathbf{x}_2$. By definition, the moment vector is perpendicular to each displacement along the line, so the vector dot product is $\mathbf{d} \cdot \mathbf{m} = 0$. The two vectors, the direction vector of the line and the moment vector, are sufficient to uniquely determine line L. Therefore, plücker coordinates are given by :
$(\mathbf{d} : \mathbf{m}) = (d_1 : d_2 : d_3 : m_1 : m_2 : m_3)$.

5.2 Leg Observation

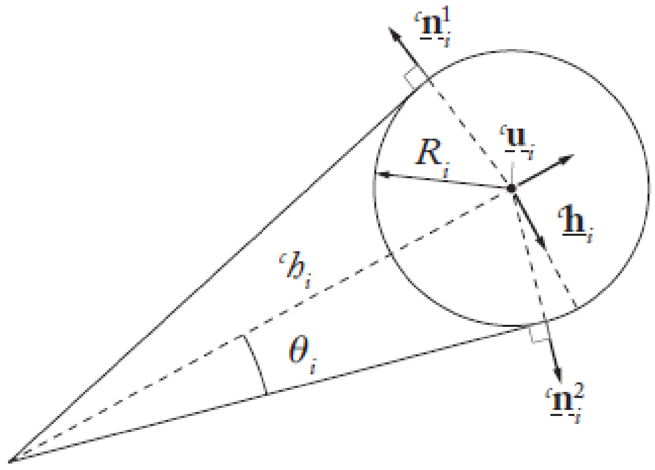
The control schemes developed and studied during this work i.e, line orientation and edges based, are defined based on the fact that it is possible to control by observing the legs of the robot. The following subsections describe the ways to extract leg orientation and edges, using geometrical relations.

5.2.1 Line Modelling

A line L expressed in the camera frame, is define by its binormalized plücker coordinates [21] :

$$L = ({}^c\underline{\mathbf{u}}, {}^c\underline{\mathbf{h}}, {}^c h)$$

Here, ${}^c\underline{\mathbf{u}}$ is the unit vector giving the spatial orientation of the line, ${}^c\underline{\mathbf{h}}$ is the unit vector defining the interpretation plane of line L and ${}^c h$ is a non negative scalar. The latter are defined by ${}^c h {}^c\underline{\mathbf{h}} = {}^c\mathbf{P} \times {}^c\underline{\mathbf{u}}$, where ${}^c\mathbf{P}$ is position of any point P on the line, expressed in camera frame. Using this notation, the well known normalized plücker coordinates [22] are the couple $({}^c\underline{\mathbf{u}}, {}^c h {}^c\underline{\mathbf{h}})$.



${}^{c}\underline{\mathbf{n}}_i^1$
R_i
${}^{c}\underline{\mathbf{u}}_i$
${}^{c}\underline{\mathbf{h}}_i$
${}^{c}h_i$
θ_i
${}^{c}\underline{\mathbf{n}}_i^2$

Figure 5.1: Plücker Coordinates and edges of a cylinder [16]

The projection of such a line in the image plane, expressed in the camera frame, has for characteristic equation [21]:

$$ {}^{c}\underline{\mathbf{h}}^{T\,c}\mathbf{p} = 0 \tag{5.1} $$

where ${}^{c}\mathbf{p}$ are the coordinates of a point P lying on the line, in the image plane, expressed in camera frame.
If we denote the intrinsic parameter matrix of the camera as $\mathbf{K}$, we can obtain the line equation in pixel coordinates ${}^{p}\underline{\mathbf{h}}$ from:

$$ {}^{p}\underline{\mathbf{h}}^{T\,p}\mathbf{p} = 0 \tag{5.2} $$

Replacing ${}^{p}\mathbf{p}$ with $\mathbf{K}{}^{c}\mathbf{p}$ in this expression yields:

$$ {}^{p}\underline{\mathbf{h}}^{T}\mathbf{K}{}^{c}\mathbf{p} = 0 $$

From equations 5.1 and 5.2, we can write:

$$ {}^{p}\underline{\mathbf{h}} = \frac{\mathbf{K}^{-T\,c}\underline{\mathbf{h}}}{\|\mathbf{K}^{-T\,c}\underline{\mathbf{h}}\|} \tag{5.3} $$

$$ {}^{c}\underline{\mathbf{h}} = \frac{\mathbf{K}^{-T\,p}\underline{\mathbf{h}}}{\|\mathbf{K}^{-T\,p}\underline{\mathbf{h}}\|} \tag{5.4} $$

5.2.2 Cylindrical leg observation

The legs of parallel robots usually have cylindrical cross-sections [23]. Edges of i-th cylindrical leg in the camera frame, are given by [16] (Fig 5.1):

$$\begin{aligned} {}^{c}\underline{\mathbf{n}}_i^1 &= -\cos\theta_i {}^{c}\underline{\mathbf{h}}_i - \sin\theta_i {}^{c}\underline{\mathbf{u}}_i \times {}^{c}\underline{\mathbf{h}}_i \\ {}^{c}\underline{\mathbf{n}}_i^2 &= +\cos\theta_i {}^{c}\underline{\mathbf{h}}_i - \sin\theta_i {}^{c}\underline{\mathbf{u}}_i \times {}^{c}\underline{\mathbf{h}}_i \end{aligned}$$

Where:

$$\cos\theta_i = \frac{\sqrt{{}^{c}\underline{h}_i^2 - R_i^2}}{{}^{c}h_i} \qquad \sin\theta_i = \frac{R_i}{{}^{c}h_i}$$

R_i is the radius of the cylinder and $({}^{c}\underline{\mathbf{u}}, {}^{c}\underline{\mathbf{h}}, {}^{c}h)$ are the binormalized Plücker coordinates of the cylinder axis.
We can also write a relationship between the leg orientation and its edges, expressed in camera frame, as given by [16]:

$${}^{c}\underline{\mathbf{u}}_i = \frac{{}^{c}\underline{\mathbf{n}}_i^1 \times {}^{c}\underline{\mathbf{n}}_i^2}{\|{}^{c}\underline{\mathbf{n}}_i^1 \times {}^{c}\underline{\mathbf{n}}_i^2\|}$$

5.3 Simulator Design

A necessary tool to test the efficacy of developed algorithms in simulation was ADAMS model of the robot.Since SolidWorks model of the robot was available, so in order to obtain an ADAMS model, a systematic process was followed.
First, the SolidWorks model was saved as a Step file. This type of file could be opened in CATIA. In CATIA, each part of the model was saved as a separate CATProduct. Then for each CATProduct, a corresponding STL file was generated. Now these STL files could be imported as separate bodies in ADAMS.
After performing these steps, all the bodies were available in ADAMS. Now, the aim was to define corresponding markers at exact joint locations. There are two things to be considered here. The marker's location and its orientation. For locations of joints, I obtained the information using measuring tools available in CATIA. For the orientation part, the orientation of revolute joints was straightforward. But for universal joints, it was a bit tricky and hence took a big chunk of time. After some unsuccessful attempts, i eventually obtained the correct orientation for universal joints from motion analysis study of SolidWorks model. The architecture of

a leg of IRSBot-2 is shown in figure 5.2,with all the joints.The lengths $l_1 = l_2 = 321mm, l_{41} = l_{42} = 458mm$. The angle $\beta = 45$.

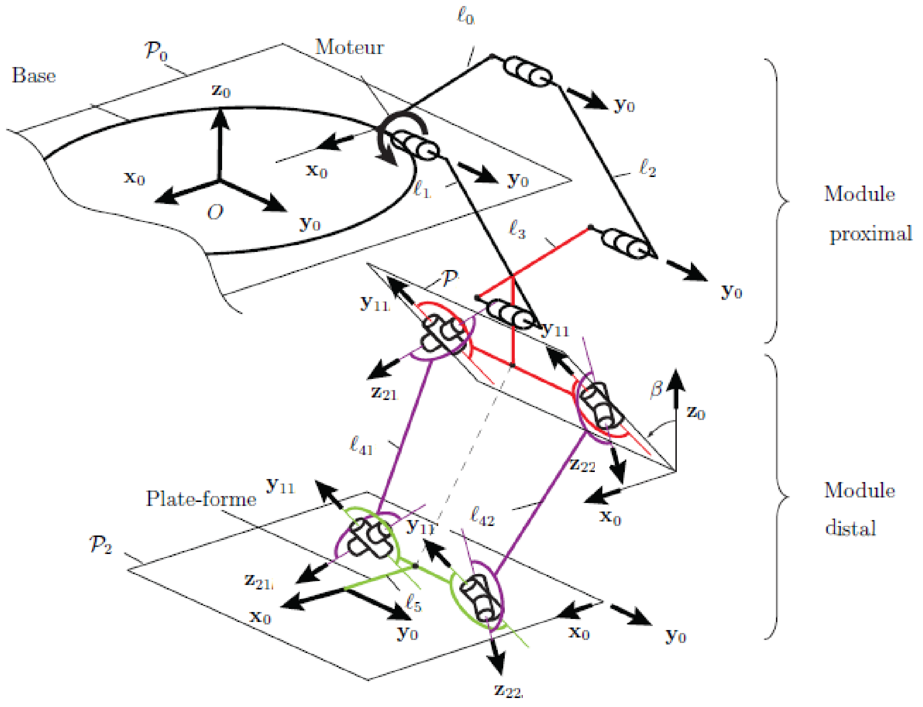


Figure 5.2: IRSBot-2 Architecture of 1 leg[24]

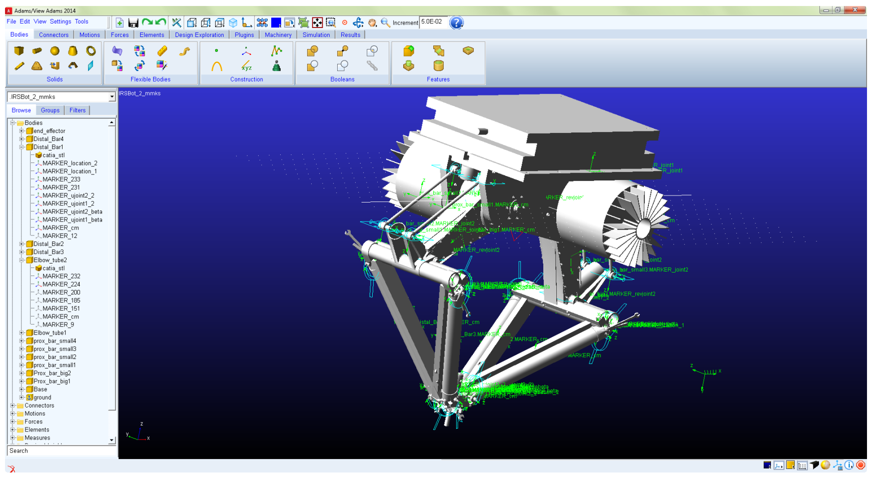


Figure 5.3: ADAMS model of IRSBot-2

The model finally obtained in ADAMS, is shown in figure 5.3

5.3.1 Linking ADAMS and MATLAB/Simulink

After obtaining the ADAMS model, the ADAMS plant was exported in order to use it with MATLAB/Simulink. The figure 5.4 shows the final model in Simulink.

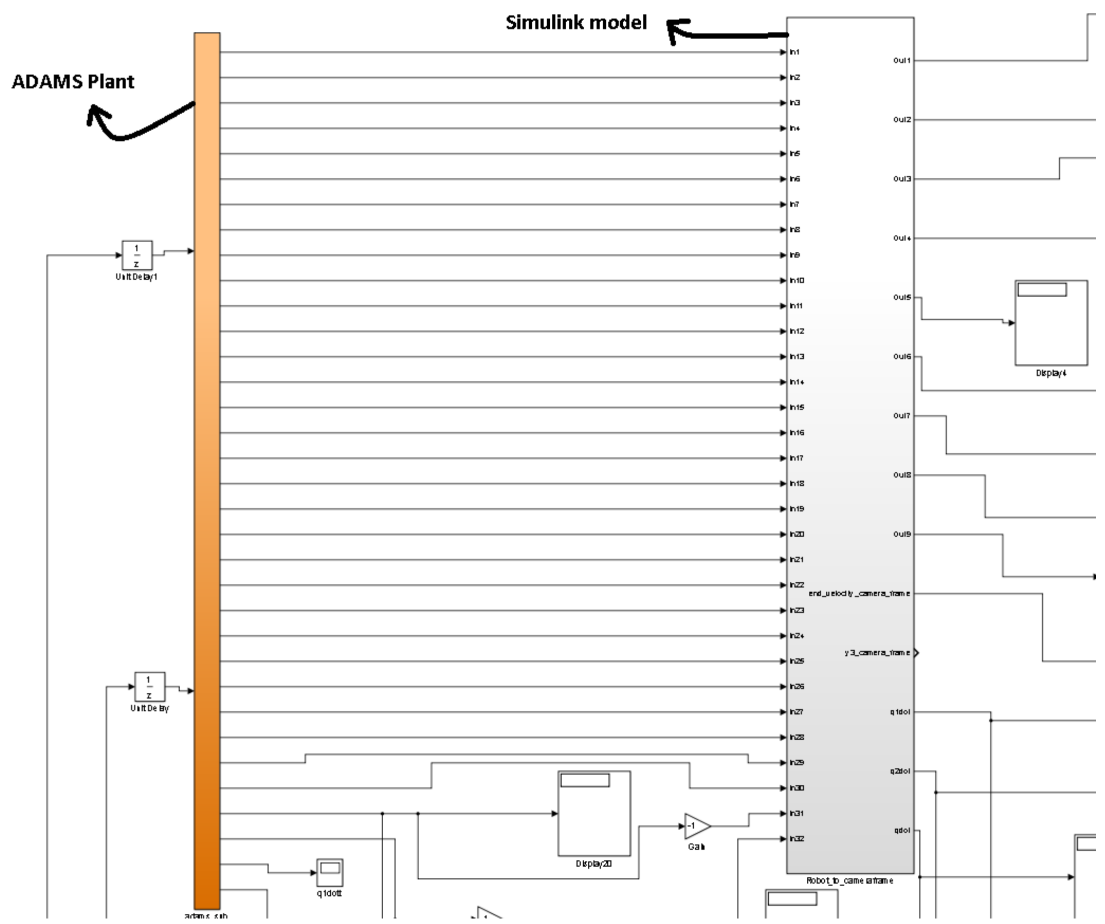


Figure 5.4: Simulation setup in simulink

5.4 Control Algorithms developed

5.4.1 Leg orientation based visual servoing

Kinematics of IRSBot-2 using leg orientation based visual servoing

In this part, we will systematically develop the kinematic model of IRSBot-2 in order to use it for the visual servoing purpose. It should be noted here that all the equations are in camera frame unless mentioned otherwise.

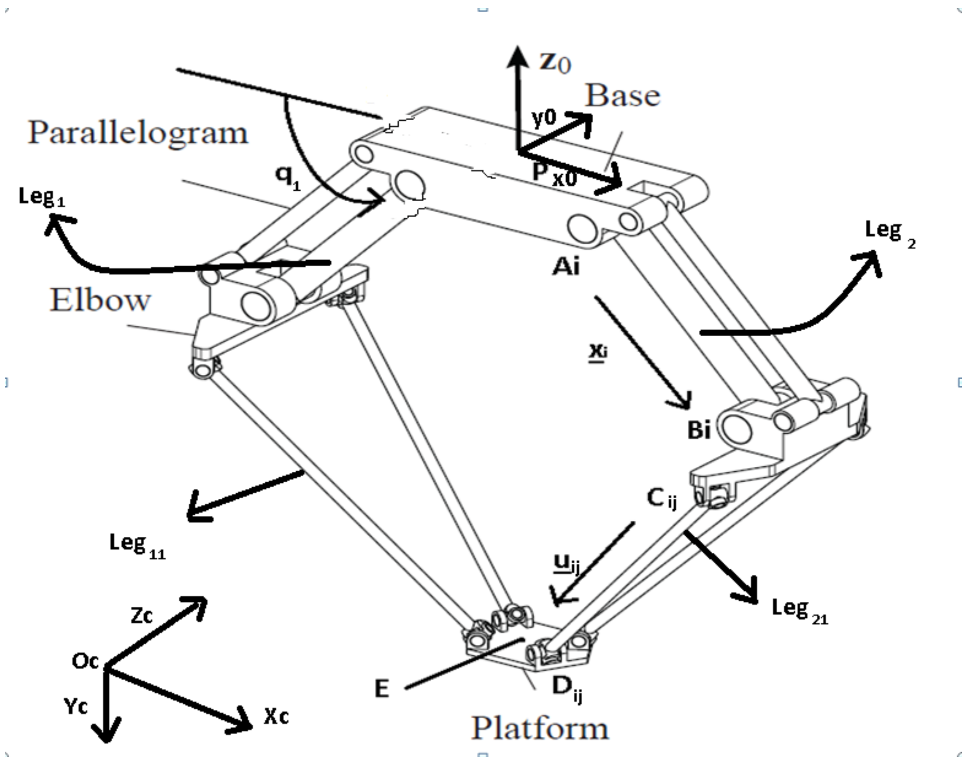


Figure 5.5: IRSBot-2 Schematic

The figure 5.5 shows the simplified architecture of IRSBot-2 and will be used to develop a kinematic model. IRSBot-2 is a two degrees of translational parallel manipulator. The robot can be basically categorized as composed of two legs. Each leg can be further subdivided into a proximal part and distal part, linked by an elbow. The proximal part is a parallelogram as shown in figure 5.5.In this part, the revolute joints that are actuated are located at point A_i, which move the arm attached at points A_i and B_i. The direction vector of this arm is given by the vector $\underline{x}_i$. The distal part can be further split into two distal bars or cylinders. An elbow joints the distal part with the proximal part and the end-effector. A distal leg, or cylinder is attached at points C_{ij} and D_{ij} with the help of universal joints. Here, i$= 1, 2$ denotes the index of the proximal part and j$= 1, 2$ denotes the index of the distal part.

Differential Inverse Kinematic model

Considering the figure 5.5, we can write, for a leg 'ij',the following equation:

$$L\underline{u}_{ij} = \underline{CD}_{ij} = D_{ij} - C_{ij} \tag{5.5}$$

where:
L is the length of distal bar
$\underline{u}_{ij}$ is the direction or orientation vector of leg 'ij' in space
C and D are the attachment points of the leg under consideration
Differentiating eq 5.5 with respect to time, we obtain:

$$L\dot{\underline{u}}_{ij} = \dot{D}_{ij} - \dot{C}_{ij} \tag{5.6}$$

From figure 5.5, we can also write

$$C_{ij} = \underline{PA}_i + l\underline{x}_i + \underline{BC}_{ij} \tag{5.7}$$

where l is the length of the proximal bar.
In eq 5.7, the terms $\underline{PA}_i$ and $\underline{BC}_{ij}$ are constant. So if we differentiate this equation with respect to time, we will obtain:

$$\dot{C}_{ij} = l\dot{\underline{x}}_i$$

or we can write in the following form:

$$\dot{C}_{ij} = l\dot{q}_i\underline{y}_i \tag{5.8}$$

Here $\underline{x}_i$ can be written for each leg under consideration and a corresponding analytical expression was obtained for $\underline{y}_i$. For instance,$\underline{x}_2$ could be written in vector form , in camera frame as:

$$\underline{x}_1 = \begin{bmatrix} \cos q_1 \\ \sin q_1 \\ 0 \end{bmatrix} \tag{5.9}$$

Similarly, we can write the expression for leg 1 .From there, we can derive the vector $\underline{y}_i$, for corresponding legs.
From figure5.5, we can also write:

$$D_{ij} = E + \underline{ED}_{ij}$$

Time differentiating the above equation, we obtain:

$$\dot{D}_{ij} = V_e + \dot{\underline{ED}}_{ij}$$

Since $\underline{ED}$ is constant, the above equation simplifies to:

$$\dot{D}_{ij} = V_e \tag{5.10}$$

Putting 5.8 and 5.10 in equation 5.6, we obtain:

$$L\underline{\dot{u}}_{ij} = V_e - l\dot{q}_i\underline{y}_i \tag{5.11}$$

Since $\underline{u}$ is a unit vector , so the following relation holds:

$$\underline{u}_{ij}^T\underline{\dot{u}}_{ij} = 0$$

Hence we can obtain the following relation after simplification:

$$\dot{q}_i = \frac{\underline{u}_{ij}^T V_e}{l\underline{y}_i^T \underline{u}_{ij}} \tag{5.12}$$

This could be written in simplified form as:

$$\dot{q}_i = J_i^{inv} V_e \tag{5.13}$$

Where J_i^{inv} is the inverse jacobian matrix relating the end-effector velocity to the joint value of the leg considered.

Interaction Matrix Computation

Since we want to use visual servoing, determination of interaction matrix relating the rate of change of features to the end effector velocity is essential. For this purpose,if we insert eq 5.12 into eq 5.11, we obtain:

$$\dot{u}_{ij} = \frac{1}{L}\left(I_3 - \frac{\underline{y}_i\underline{u}_{ij}^T}{\underline{y}_i^T \underline{u}_{ij}}\right)V_e$$

The above equation can be written in simplified form as:

$$\dot{u}_{ij} = M_{ij} V_e \tag{5.14}$$

Here, M_{ij} denotes the interaction matrix relating the rate of change of features to the end effector velocity.

Control Law

Since we are dealing with unit vectors, geodesic error is a better way to consider error as compared to the difference of vectors. If we consider that u_{ij} is our current leg direction at some point in time and u_{ij}^* is our desired leg direction, then we can write the error as:

$$e_{ij} = u_{ij} \times u_{ij}^*$$

Taking the time derivative of above equation, we can write:

$$\dot{e_{ij}} = -[u_{ij}^*]_\times \dot{u}_{ij}$$

Using equation 5.14, we can write the above equation as:

$$\dot{e_{ij}} = -[u_{ij}^*]_\times M_{ij} V_e$$

or

$$\dot{e_{ij}} = N_{ij} V_e \tag{5.15}$$

where:

$$N_{ij} = -[u_{ij}^*]_\times M_{ij}$$

For a simple control strategy, if we impose proportional decrease of the error, we can write:

$$\dot{e_{ij}} = -\lambda e_{ij}$$

Using this in eq 5.15, we get following pseudo control law:

$$V_e = -\lambda N^+ e \tag{5.16}$$

Notice that matrix N is a compound matrix obtained by stacking individual matrices N_{ij} of each leg and e is the compound error matrix obtained by stacking the individual errors of each leg. Using eq 5.13 in eq 5.16, we have the final control law of the form:

$$\dot{q} = -\lambda J^{inv} N^+ e$$

where q is a vector obtained by stacking the values from both actuators. J^{inv} is the matrix obtained by stacking the inverse jacobian matrices of both legs.

5.4.2 Results without noise

The following section shows some of the results obtained using the Simulink / ADAMS View based simulator. Noise is not added in this part. The desired leg orientation vectors of both legs are provided, which correspond to a certain pose of end-effector. For simulation purposes, as shown in figure 5.5, the camera is supposed to be placed at some location away from the robot and the camera frame is rotated for 90° around x-axis. The general form of homogeneous transformation matrix between camera and base frame can be written as:

$$ {}^{c}\mathbf{T}_{o} = \begin{bmatrix} {}^{c}\mathbf{R}_{o} & & & {}^{c}\mathbf{t}_{o} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

So the homogeneous transformation matrix between the robot and camera frame is as follows:

$$ {}^{c}\mathbf{T}_{o} = \begin{bmatrix} 1 & 0 & 0 & 250 \\ 0 & 0 & -1 & -250 \\ 0 & 1 & 0 & 1000 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

The translation part of the homogeneous transformation matrix is in millimetres.
The initial leg orientations of each leg were:

$$ \underline{u}_{21} = \begin{bmatrix} -0.8056 \\ 0.5092 \\ 0.3032 \end{bmatrix} \quad \underline{u}_{11} = \begin{bmatrix} 0.5150 \\ 0.8019 \\ 0.3032 \end{bmatrix} $$

These initial leg orientations correspond to the initial end effector pose of:

$$ X = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 100 \\ 342.9 \\ 1000 \end{bmatrix} $$

The desired orientations of each leg are as shown below:

$$ \underline{u}^{*}_{21} = \begin{bmatrix} -0.7131 \\ 0.6322 \\ 0.3032 \end{bmatrix} \quad \underline{u}^{*}_{11} = \begin{bmatrix} 0.4163 \\ 0.8573 \\ 0.3032 \end{bmatrix} $$

These desired leg orientations correspond to the end effector pose of:

$$X = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 88.1 \\ 413.1 \\ 1000 \end{bmatrix}$$

It should be noted that all the vectors described here are in camera frame.Figure 5.6 demonstrates the results obtained for leg$_{11}$.

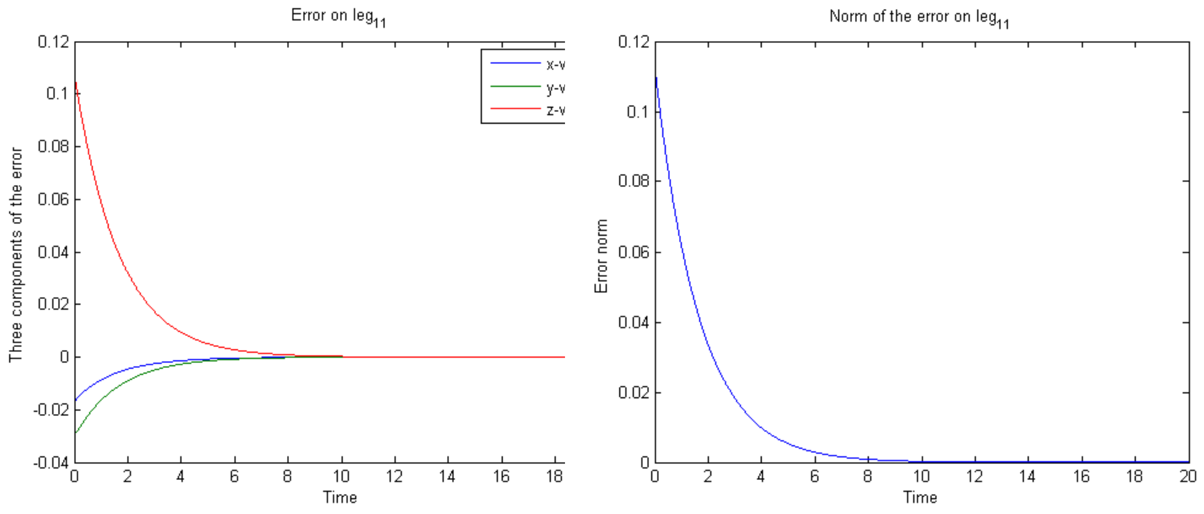


Figure 5.6: Error on Leg$_{11}$

Similarly, figure 5.7 shows the error and norm of error on the leg $_{21}$. It can be noted that in both cases, the desired leg orientation was achieved and there was an exponential decay of error, as desired.

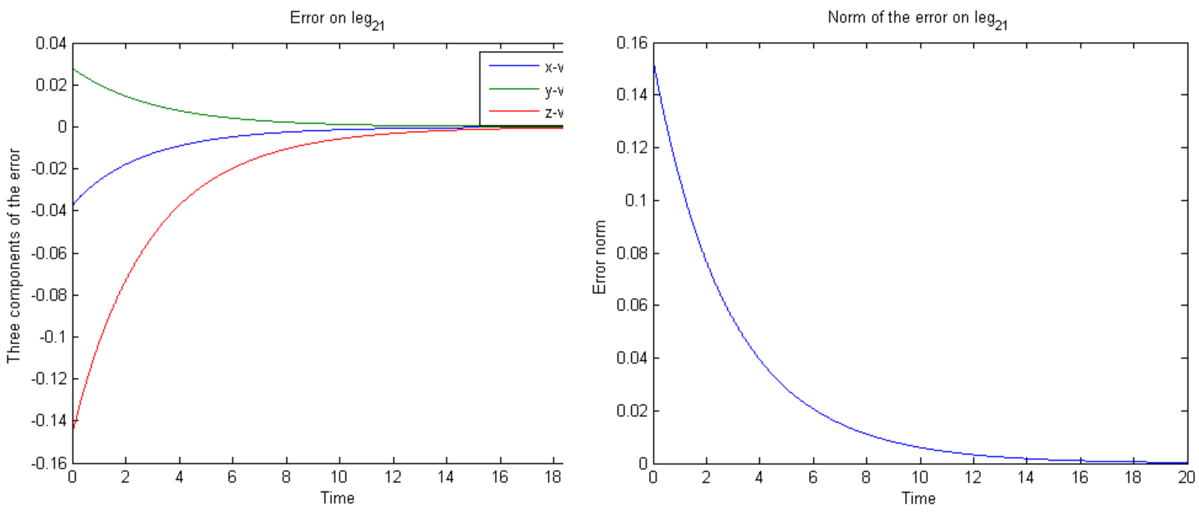


Figure 5.7: Error on Leg$_{21}$

Secondly, I changed the desired leg orientations and see if these new values could be attained. Now, the desired orientations of each leg are as below:
The desired orientations of each leg are as shown below:

$$\underline{u}_{21}^* = \begin{bmatrix} -0.5939 \\ 0.7453 \\ 0.3032 \end{bmatrix} \quad \underline{u}_{11}^* = \begin{bmatrix} 0.3250 \\ 0.8959 \\ 0.3032 \end{bmatrix}$$

These desired leg orientations correspond to the end effector pose of:

$$X = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad = \begin{bmatrix} 86.88 \\ 469.1 \\ 1000 \end{bmatrix}$$

Figure 5.8 demonstrates the results obtained for leg$_{11}$.

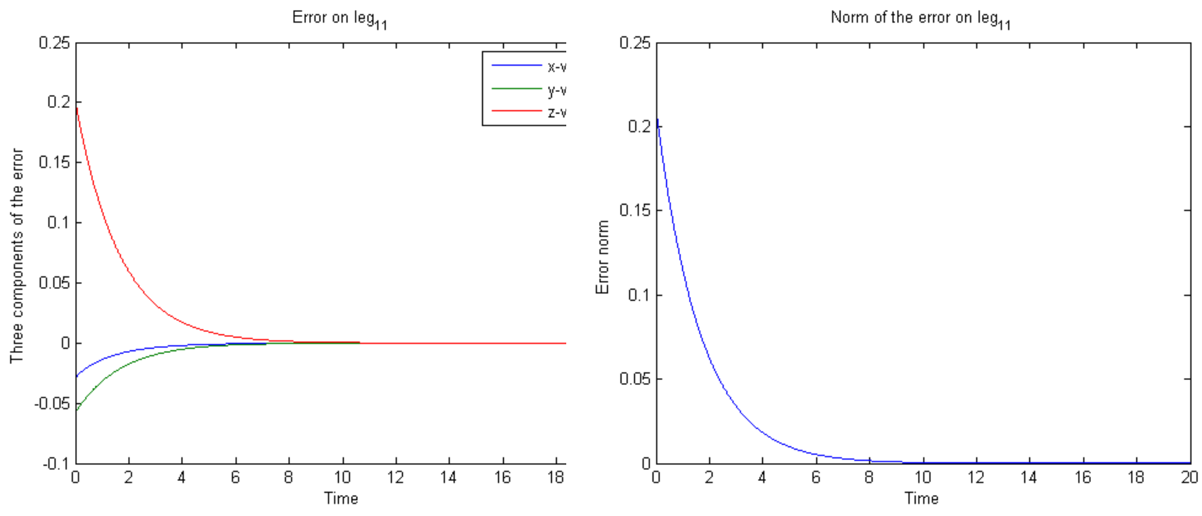


Figure 5.8: Error on Leg$_{11}$

Similarly, figure 5.9 shows the error and norm of error on the leg $_{21}$. It can be noted that in both cases, the desired leg orientation was achieved and there was an exponential decay of error, as desired.

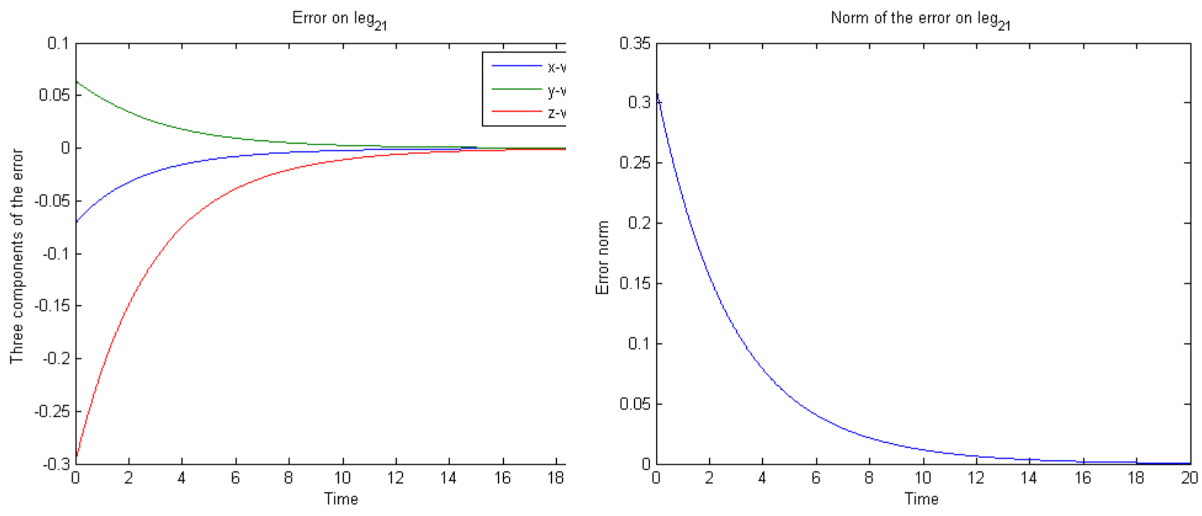


Figure 5.9: Error on Leg$_{21}$

5.4.3 Results with noise

In this subsection, we will observe the effect of a small noise added. This will be closer to practical scenario, since there will be some noise in determination of leg edges or leg orientation in real life.
This effect was mimicked in simulation by introducing additive white gaussian noise in our model, at the location where we are determining the orientation vectors. The noise was added with a variance of 0.001 to the original vector measured. The results are as shown below in the figures:

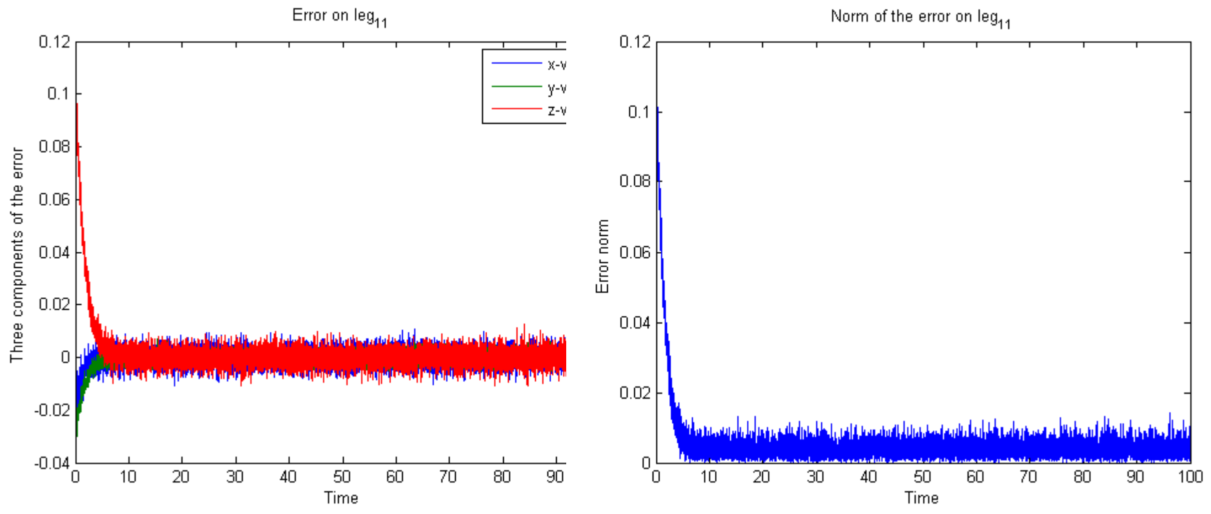


Figure 5.10: Error on Leg_{11}

Similarly, figure 5.11 shows the error and norm of error on the leg $_{21}$. It can be noted that in both cases, the desired leg orientation was achieved and there was an exponential decay of error, as desired.

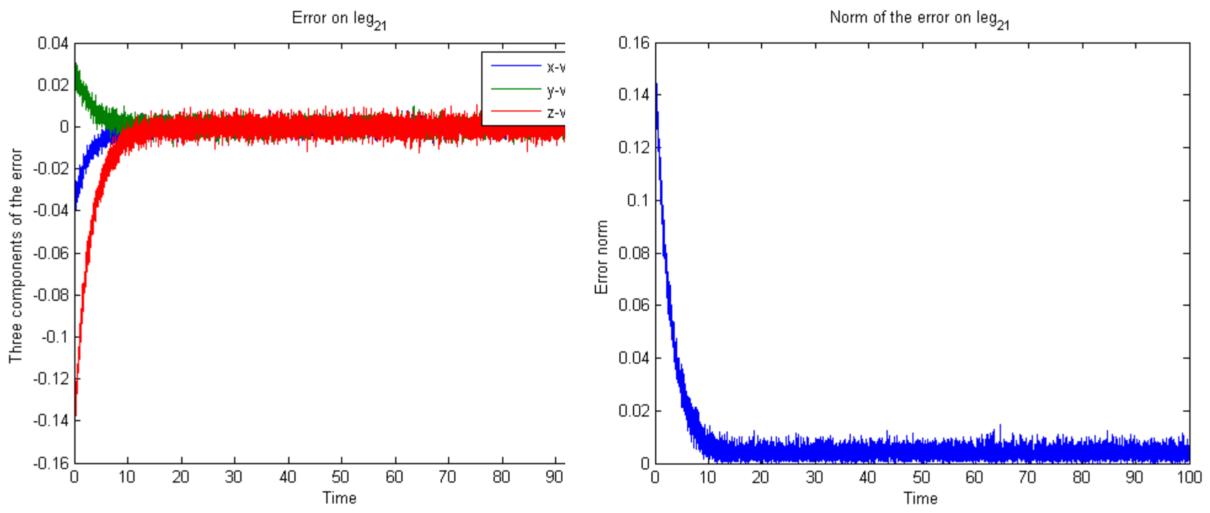


Figure 5.11: Error on Leg_{21}

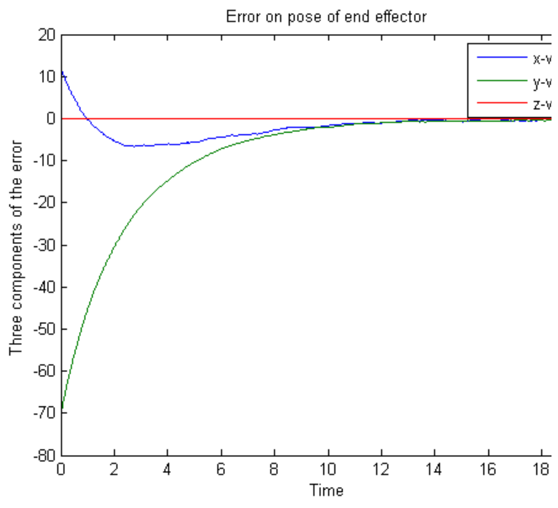
The pose reached by end-effector in this case was

$$X = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 87.78 \\ 412.8 \\ 1000 \end{bmatrix}$$

As described previously the desired pose was :

$$X = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 88.1 \\ 413.1 \\ 1000 \end{bmatrix}$$

The figure 5.12 shows that despite the noise introduced in the measurement of orientation vectors, the desired end-effector pose was reached with minimal error.



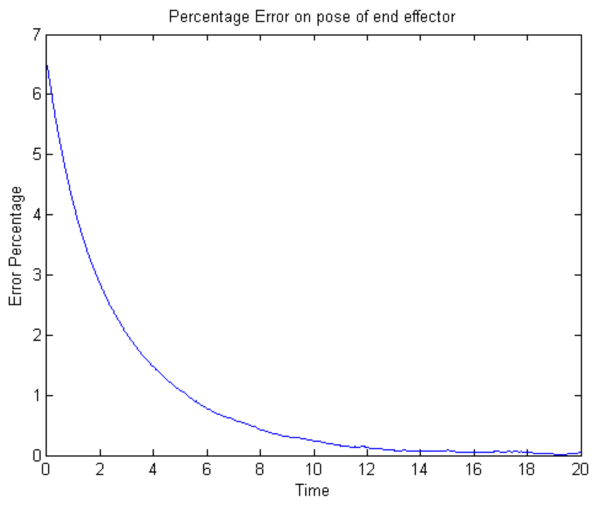


Figure 5.12: Error on pose of end-effector

We can see that the percentage error on the end-effector pose is around 0.2%, which is negligible.

5.4.4 Leg edges based visual servoing

As discussed previously, if we want to control using leg directions in a practical scenario, we will make use of the following relation between a leg direction and its edges:

$$^{c}\underline{\mathbf{u}}_{ij} = \frac{^{c}\underline{\mathbf{n}}^{1}_{ij} \times {}^{c}\underline{\mathbf{n}}^{2}_{ij}}{\|^{c}\underline{\mathbf{n}}^{1}_{ij} \times {}^{c}\underline{\mathbf{n}}^{2}_{ij}\|}$$

where $\underline{\mathbf{n}}^{1}_{ij}$ and $\underline{\mathbf{n}}^{2}_{ij}$ denote the two edges of the leg and $^{c}\underline{\mathbf{u}}_{ij}$ denotes the corresponding leg direction. This relation is useful but not always. Consider a scenario where the two edges of the leg appear parallel in the image. So it means they will appear to intersect at infinity and hence the leg direction cannot be determined. For this purpose, we will try to control using leg edges rather than the leg directions, in this part. Note that leg directions based visual servoing is close to position based visual servoing while leg edges based visual servoing is closer to image based visual servoing.
Since each cylinder edge is a line in space, it can be expressed by binormalized plücker coordinates $(^{c}\underline{\mathbf{u}}_{ij}, {}^{c}\underline{\mathbf{n}}^{k}_{ij}, {}^{c}n^{k}_{ij})$. Also, the attachment pont C_{ij} is lying at a distance R (radius of the cylinder) from the edge. Consequently,

a cylinder edge can be fully defined by following constraints [16]:

$$C_{ij}^T \underline{n}_{ij}^k = -R \tag{5.17}$$

$$\underline{n}_{ij}^{kT} \underline{n}_{ij}^k = 1 \tag{5.18}$$

$$\underline{u}_{ij}^T \underline{n}_{ij}^k = 0 \tag{5.19}$$

The interaction matrix $\mathbf{W}$ relating the end effector velocity to the edges in the pixel frame can be written as:

$$\dot{\underline{n}}^k = \mathbf{W} V_e \tag{5.20}$$

The matrix R can be decomposed into three parts, as follows:

$$\mathbf{W} = {}^p\mathbf{Q}_c {}^n\mathbf{Q}_u \mathbf{M} \tag{5.21}$$

The matrix $\mathbf{M}$, as derived previously, relates the time rate of change of leg orientation to the end effector velocity. The matrix ${}^n\mathbf{Q}_u$ relates the leg orientation velocities and leg edges velocities, in camera frame. The matrix ${}^p\mathbf{Q}_c$ is used to change from camera to pixel frame. The two latter matrices will be derived in the following sections.

Edge velocity in the camera frame

Here, we will derive the time derivative of a cylinder edge, in camera frame, under the kinematic constraint that cylinder is attached at point C_{ij} . For that purpose, taking time derivative of the constraints expressed in equations 5.17, 5.18 and 5.19, we have the following relations. Note that the legs index 'ij ' is dropped from following calculations for simplicity:

$$\dot{\underline{n}}^{kT} C = 0 \tag{5.22}$$

$$\dot{\underline{n}}^{kT} \underline{n}^k = 0 \tag{5.23}$$

$$\dot{\underline{n}}^{kT} \underline{u} + \underline{n}^{kT} \dot{\underline{u}} = 0 \tag{5.24}$$

Using the equation 5.18 and the fact that the vectors $(\underline{\mathbf{u}}, \underline{\mathbf{n}}, \underline{\mathbf{u}} \times \underline{\mathbf{n}})$ form the orthonormal basis (Andreff et al. 2002), we can state that:

$$\dot{n}^k = \alpha \underline{u} + \beta \underline{u} \times \underline{n}^k$$

Now inserting this expression into equations 5.17 and 5.19 yields :

$$\alpha = -\underline{n}^k \dot{\underline{u}}, \quad \beta = \frac{C^T \underline{u}}{C^T(\underline{u} \times \underline{n}^k)} (\underline{n}^{kT} \dot{u})$$

Consequently, we obtain the relationship between the time derivative of a leg edge, expressed in the camera frame, and the time derivative of the leg orientation:

$$\dot{\underline{n}}^k = {}^n\mathbf{Q}_u \dot{\underline{u}} = {}^n\mathbf{Q}_u \mathbf{M} V_e$$

where:

$${}^n\mathbf{Q}_u = \left(\frac{C^T \underline{u}}{C^T(\underline{u} \times \underline{n}^k)} (\underline{u} \times \underline{n}^k) - \underline{u} \right) \underline{n}^{kT}$$

Image line velocity in pixel coordinates

In this section, we will derive a jacobian associated with the change of frame in which the time derivative of an image line is expressed, from camera frame to the pixel frame. These calculations hold for any image line, not only for edges.
Rewriting 5.3, we have:

$$ {}^p\underline{n} = \mu({}^c\underline{n}) K^{-T} {}^c\underline{n} \tag{5.25}$$

Time differentiating the above equation:

$$ {}^p\dot{\underline{n}} = \frac{d\mu({}^c\underline{n})}{dt} K^{-T} {}^c\underline{n} + \mu({}^c\underline{n}) K^{-T} {}^c\dot{\underline{n}} \tag{5.26}$$

Since ${}^p\underline{n}$ is a unit vector, so using equation 5.18, we get :

$$\left(\frac{d\mu({}^c\underline{n})}{dt} K^{-T} {}^c\underline{n} \right)^T {}^p\underline{n} + \mu({}^c\underline{n}) {}^p\underline{n}^T K^{-T} {}^c\dot{\underline{n}} = 0$$

Using 5.3 again, this simplifies into :

$$\frac{d\mu({}^c\underline{n})}{dt} = -\mu^c(\underline{n})^2 \, {}^p\underline{n}^T K^{-T} \, {}^c\dot{\underline{n}}$$

Inserting this result into equation 5.26, we have:

$$ {}^p\dot{\underline{n}} = \left(-K^{-T} \, {}^c\underline{n}^T \mu({}^c\underline{n})^2 \, {}^p\underline{n}^T + \mu({}^c\underline{n}) I_3 \right) K^{-T} {}^c\dot{\underline{n}}$$

This simplifies into:

$$ {}^p\dot{\underline{n}} = \mu\left({}^c\underline{n}\right) \left(I_3 - {}^p\underline{n} \, {}^p\underline{n}^T \right) K^{-T} \, {}^c\dot{\underline{n}}$$

Inserting eq. 5.4 into eq. 5.25 proves that:

$$\mu({}^c\underline{n}) = \| K^T \, {}^p\underline{n} \|$$

From this, we finally obtain the relationship between the time derivative of a line, expressed in the pixel frame and in camera frame:

$$ {}^{p}\dot{\underline{n}} = {}^{p}\mathbf{Q}_c \, {}^{c}\dot{\underline{n}} $$

$$ {}^{p}\mathbf{Q}_c = \|K^T \, {}^{p}\underline{n}\| \left(I_3 - {}^{p}\underline{n}{}^{p}\underline{n}^T\right) K^{-T} $$

Thus we have all the matrices needed to use equation 5.21.

Controlling in pixel coordinates

In our case,geodesic error is a better way to consider error as compared to the difference of vectors. If we consider that $\underline{n}_{ij}^{k}$ is one of our current leg edge at some point in time and $\underline{n}_{ij}^{k*}$ is our desired leg edge , then we can write the error as:

$$ e_{ij}^{k} = \underline{n}_{ij}^{k} \times \underline{n}_{ij}^{k*} $$

Taking the time derivative of above equation, we can write:

$$ \dot{e}_{ij}^{k} = -[\underline{n}_{ij}^{k*}]_{\times} \dot{\underline{n}}_{ij}^{k} $$

Using equation 5.20, we can write the above equation as:

$$ \dot{e}_{ij}^{k} = -[\underline{n}_{ij}^{k*}]_{\times} \mathbf{W}_{ij}^{k} V_e $$

or

$$ \dot{e}_{ij}^{k} = \mathbf{S}_{ij}^{k} V_e \tag{5.27} $$

where:

$$ \mathbf{S}_{ij}^{k} = -[\underline{n}_{ij}^{k*}]_{\times} \mathbf{W}_{ij}^{k} $$

For a simple control strategy, if we impose proportional decrease of the error, we can write:

$$ \dot{e_{ij}} = -\lambda e_{ij} $$

Using this in eq 5.27, we get following pseudo control law:

$$ V_e = -\lambda \mathbf{S}_{ij}^{+} e \tag{5.28} $$

Notice that matrix $\mathbf{S}$ is a compound matrix obtained by stacking individual matrices $\mathbf{S}_{ij}^{k}$ of all the edges of all legs and e is the compound error matrix obtained by stacking the individual errors of each edge. Using eq 5.13 in eq

5.28, we have the final control law of the form:

$$\dot{q} = -\lambda J^{inv} \mathbf{S}^{+} e$$

where q is a vector obtained by stacking the values from both actuators. J^{inv} is the matrix obtained by stacking the inverse jacobian matrices of both legs.

5.4.5 Results without noise

The following section shows some of the results obtained using the Simulink / ADAMS View based simulator. Noise is not added in this part. The desired leg edges vectors of both legs are provided, which correspond to a certain pose of end-effector. The initial values of the edges of leg$_{11}$ were:

$$\underline{n}_{11}^{1} = \begin{bmatrix} 0.8091 \\ -0.5715 \\ 0.137 \end{bmatrix} \quad \underline{n}_{11}^{1} = \begin{bmatrix} -0.7931 \\ 0.5799 \\ -0.1865 \end{bmatrix}$$

Similarly, for leg$_{21}$, the initial values for edges were:

$$\underline{n}_{21}^{1} = \begin{bmatrix} 0.3884 \\ 0.8401 \\ -0.3788 \end{bmatrix} \quad \underline{n}_{21}^{1} = \begin{bmatrix} -0.4082 \\ -0.8477 \\ 0.3388 \end{bmatrix}$$

These initial values for leg edges correspond to the initial end effector pose of:

$$X = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 100 \\ 342.9 \\ 1000 \end{bmatrix}$$

The desired edges of leg$_{11}$ are as shown below:

$$\underline{n}_{11}^{1*} = \begin{bmatrix} 0.4973 \\ 0.7611 \\ -0.4165 \end{bmatrix} \quad \underline{n}_{11}^{2*} = \begin{bmatrix} -0.5195 \\ -0.767 \\ 0.3765 \end{bmatrix}$$

Similarly, for leg$_{21}$, the desired values for edges were:

$$\underline{n}_{21}^{1*} = \begin{bmatrix} 0.3884 \\ 0.8401 \\ -0.3788 \end{bmatrix} \quad \underline{n}_{21}^{2*} = \begin{bmatrix} -0.4082 \\ -0.8477 \\ 0.3388 \end{bmatrix}$$

These desired leg edges values correspond to the end effector pose of:

$$X = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 88.1 \\ 413.1 \\ 1000 \end{bmatrix}$$

It should be noted that all the vectors described here are in camera frame.Figure 5.13 demonstrates the results obtained for first edge of leg$_{11}$ and figure 5.14 demonstrates the results obtained for second edge.

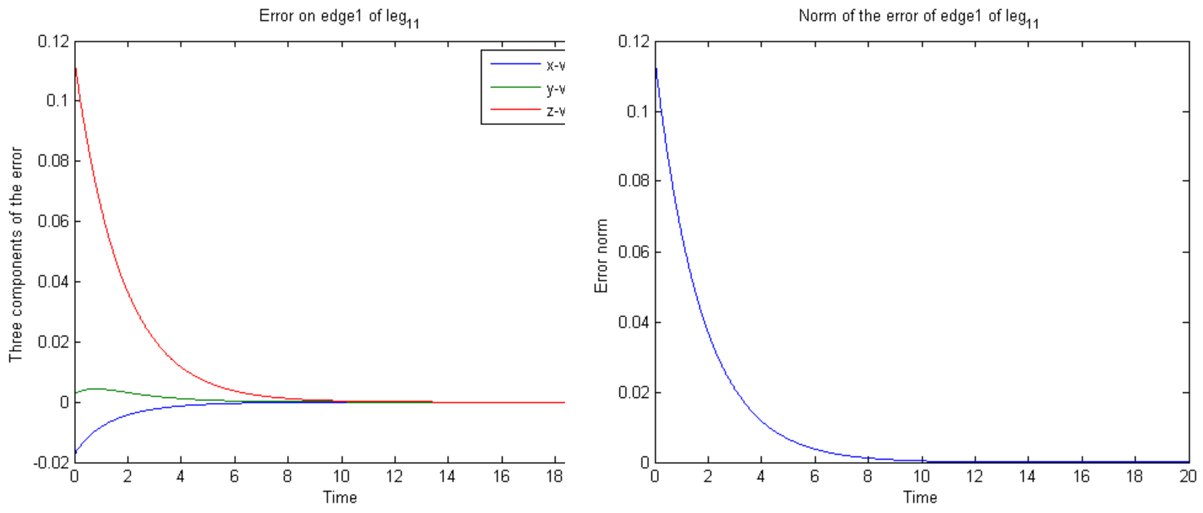


Figure 5.13: Error on edge1 of Leg$_{11}$

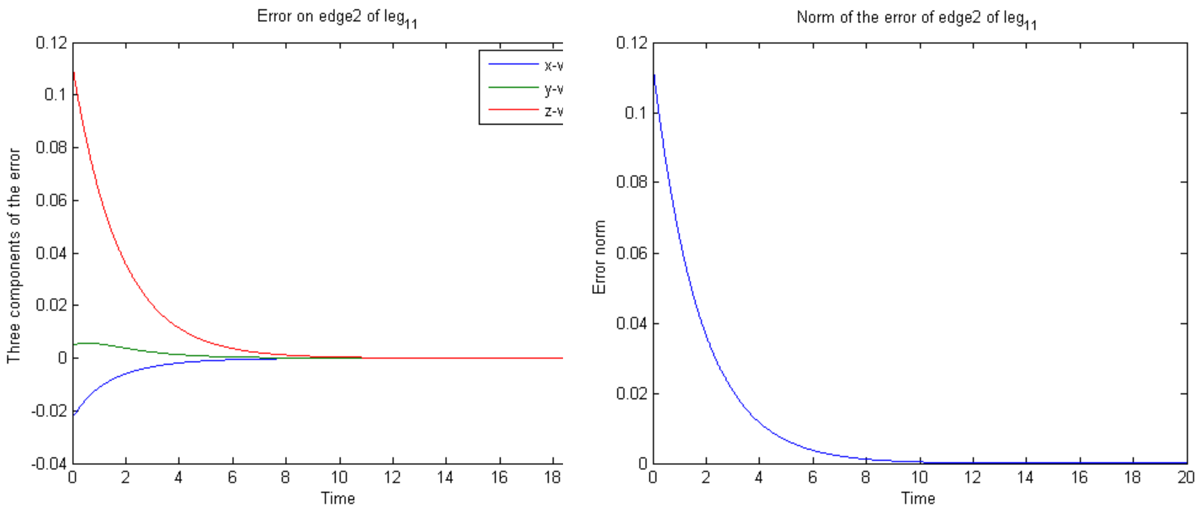


Figure 5.14: Error on edge2 of Leg$_{11}$

Similarly, figure 5.15 shows the error and norm of error on the first edge of leg $_{21}$ and 5.16 shows the error and norm of error on the second edge. It can be noted that in both cases, the desired leg orientation was achieved and there was an exponential decay of error, as desired.

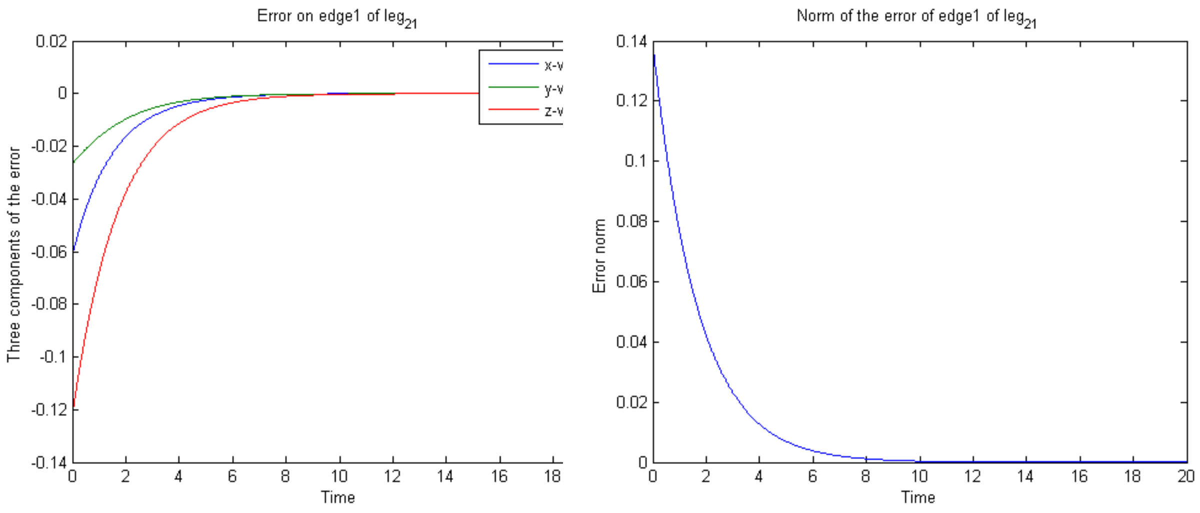


Figure 5.15: Error on first edge of Leg$_{21}$

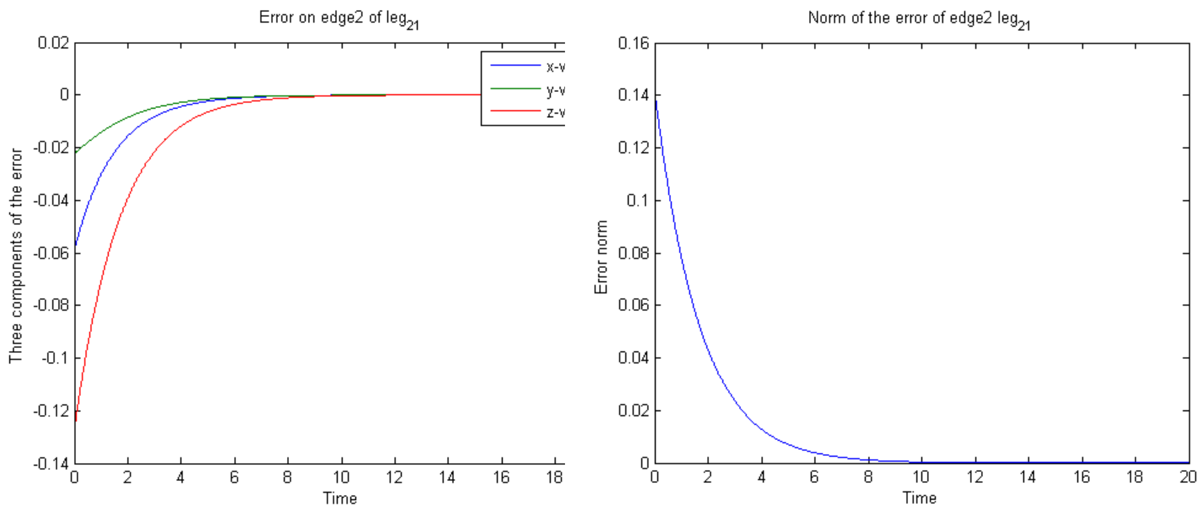


Figure 5.16: Error on second edge of Leg$_{21}$

Secondly, I changed the desired leg edges and see if these new values could be attained. Now, the desired edges of each leg are as below:
The desired orientations of leg$_{11}$ are as shown below:

$$\underline{n}_{11}^{1*} = \begin{bmatrix} 0.8946 \\ -0.4228 \\ 0.145 \end{bmatrix} \qquad \underline{n}_{11}^{2*} = \begin{bmatrix} -0.8793 \\ 0.434 \\ -0.1959 \end{bmatrix}$$

Similarly, for leg$_{21}$, the desired values for edges were:

$$\underline{n}_{21}^{1*} = \begin{bmatrix} 0.5263 \\ 0.7424 \\ -0.4146 \end{bmatrix} \qquad \underline{n}_{21}^{2*} = \begin{bmatrix} -0.549 \\ -0.7475 \\ 0.374 \end{bmatrix}$$

These desired leg edges correspond to the end effector pose of:

$$X = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad = \begin{bmatrix} 86.88 \\ 469.1 \\ 1000 \end{bmatrix}$$

Figures 5.17 and 5.18 demonstrates the results obtained for first and second edges of leg$_{11}$ respectively.

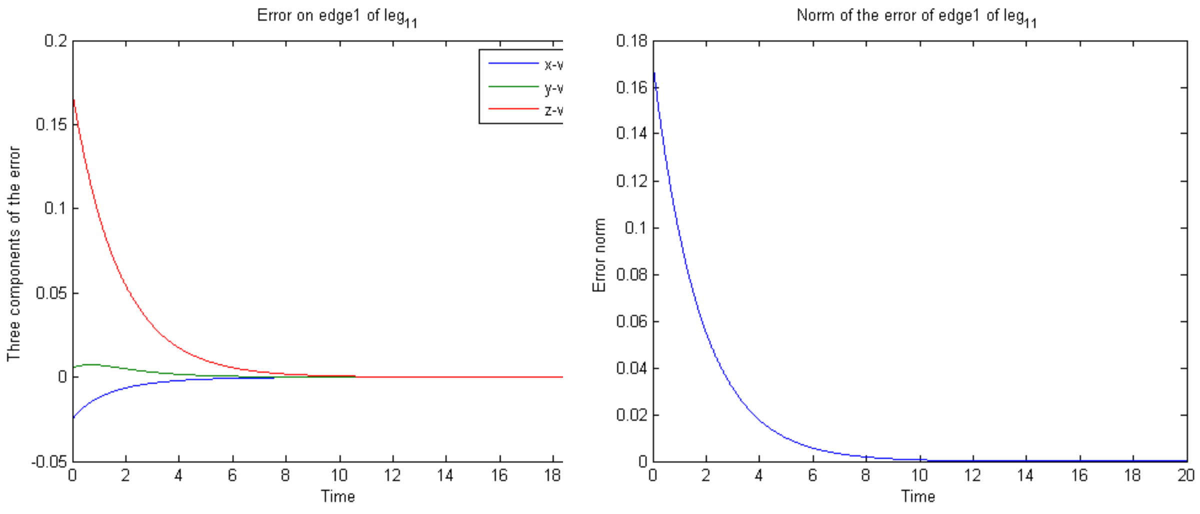


Figure 5.17: Error on first edge of Leg$_{11}$

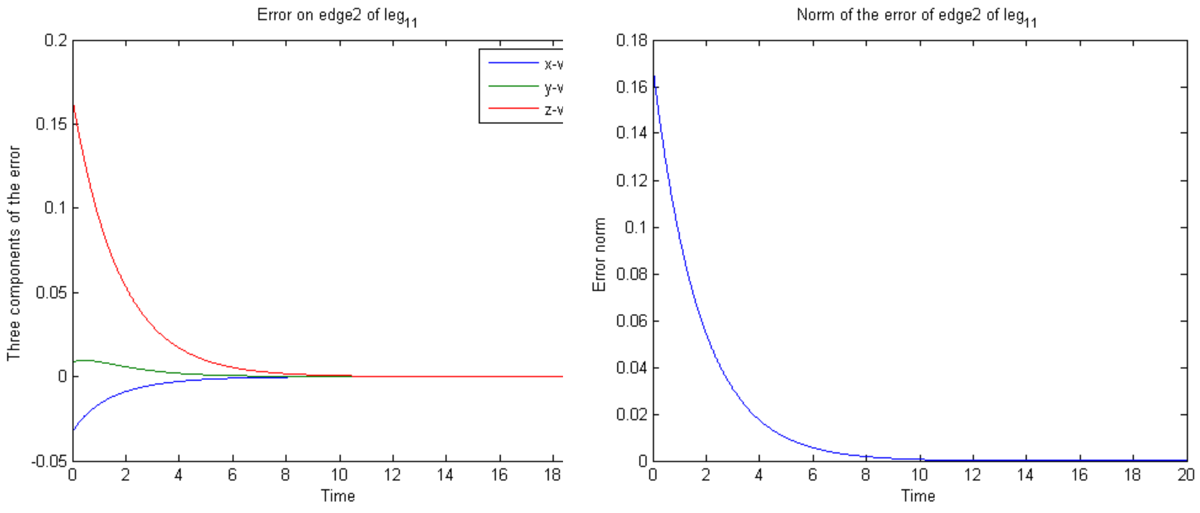


Figure 5.18: Error on second edge of Leg$_{11}$

Similarly, figures 5.19 and 5.20 show the error and norm of error on first and second edges of leg $_{21}$ respectively. It can be noted that in both cases, the desired leg orientation was achieved and there was an exponential decay of error, as desired.

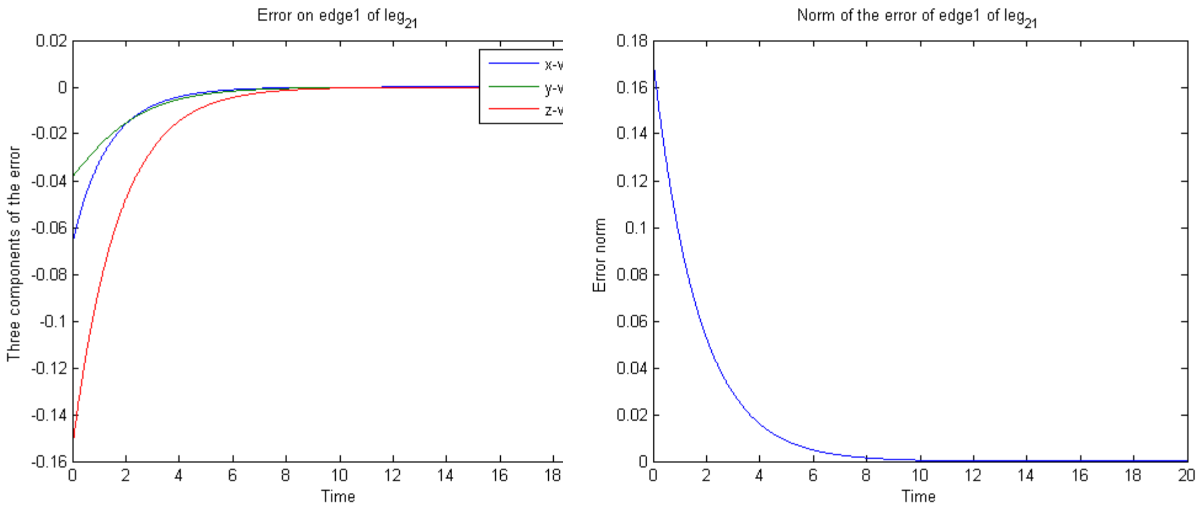


Figure 5.19: Error on first edge of Leg_{21}

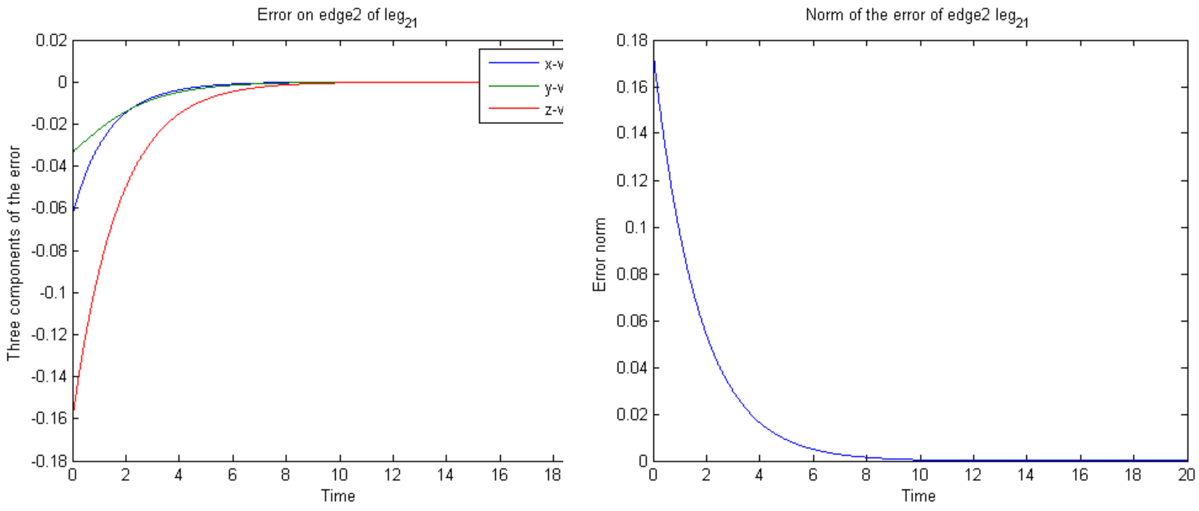


Figure 5.20: Error on second edge of Leg_{21}

5.4.6 Results with noise

In this case also, additive white gaussian noise was added with a variance of 0.001.
The initial values of the edges of leg_{11} were:

$$\underline{n}_{11}^{1} = \begin{bmatrix} 0.8091 \\ -0.5715 \\ 0.137 \end{bmatrix} \quad \underline{n}_{11}^{1} = \begin{bmatrix} -0.7931 \\ 0.5799 \\ -0.1865 \end{bmatrix}$$

Similarly, for leg$_{21}$, the initial values for edges were:

$$\underline{n}_{21}^{1} = \begin{bmatrix} 0.3884 \\ 0.8401 \\ -0.3788 \end{bmatrix} \quad \underline{n}_{21}^{1} = \begin{bmatrix} -0.4082 \\ -0.8477 \\ 0.3388 \end{bmatrix}$$

These initial values for leg edges correspond to the initial end effector pose of:

$$X = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 100 \\ 342.9 \\ 1000 \end{bmatrix}$$

The desired edges of leg$_{11}$ are as shown below:

$$\underline{n}_{11}^{1*} = \begin{bmatrix} 0.4973 \\ 0.7611 \\ -0.4165 \end{bmatrix} \quad \underline{n}_{11}^{2*} = \begin{bmatrix} -0.5195 \\ -0.767 \\ 0.3765 \end{bmatrix}$$

Similarly, for leg$_{21}$, the desired values for edges were:

$$\underline{n}_{21}^{1*} = \begin{bmatrix} 0.3884 \\ 0.8401 \\ -0.3788 \end{bmatrix} \quad \underline{n}_{21}^{2*} = \begin{bmatrix} -0.4082 \\ -0.8477 \\ 0.3388 \end{bmatrix}$$

These desired leg edges values correspond to the end effector pose of:

$$X = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 88.1 \\ 413.1 \\ 1000 \end{bmatrix}$$

It should be noted that all the vectors described here are in camera frame.Figure 5.21 demonstrates the results obtained for first edge of leg$_{11}$ and figure 5.22 demonstrates the results obtained for second edge.

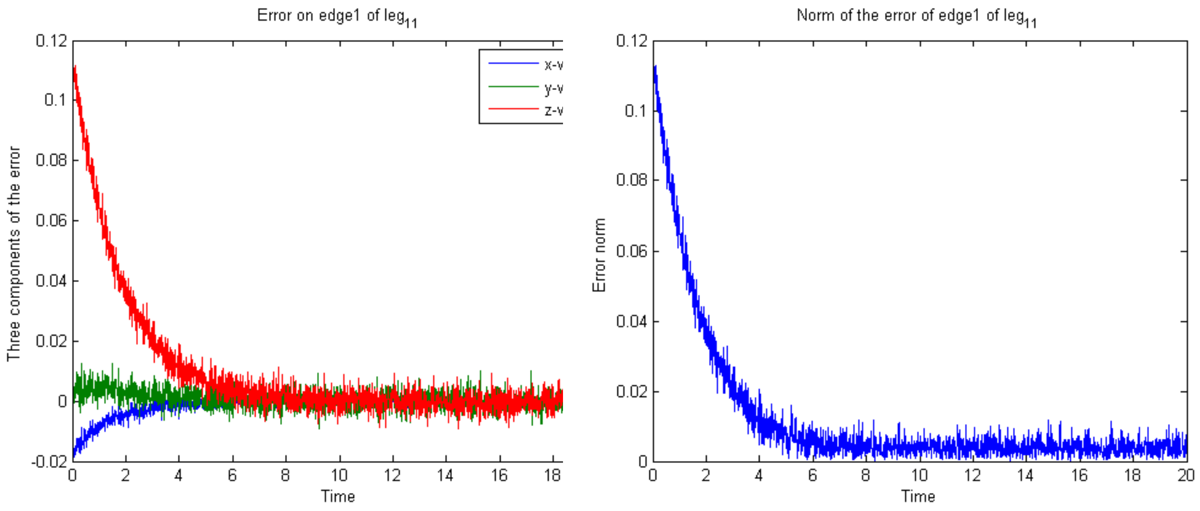


Figure 5.21: Error on edge1 of Leg$_{11}$

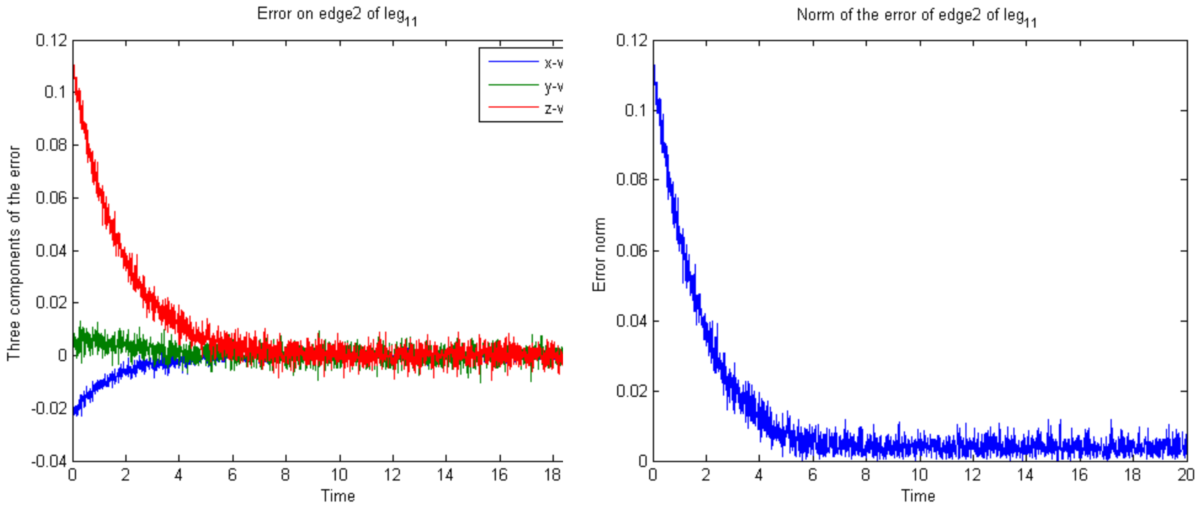


Figure 5.22: Error on edge2 of Leg$_{11}$

Similarly, figure 5.23 shows the error and norm of error on the first edge of leg $_{21}$ and 5.24 shows the error and norm of error on the second edge. It can be noted that in both cases, the desired leg orientation was achieved and there was an exponential decay of error, as desired.

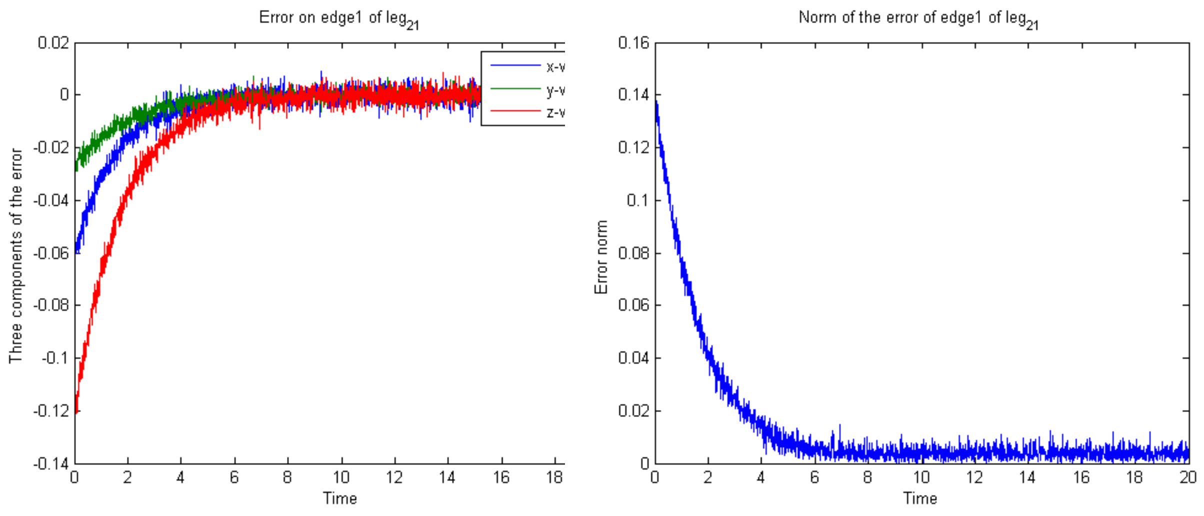


Figure 5.23: Error on first edge of Leg$_{21}$

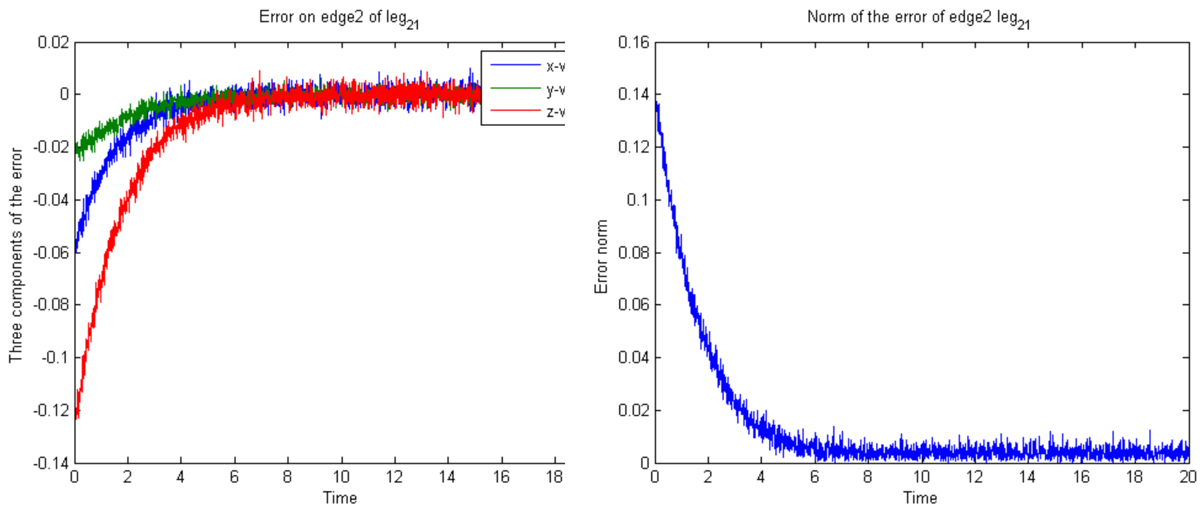


Figure 5.24: Error on second edge of Leg$_{21}$

The pose reached by end-effector in this case was:

$$X = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad = \begin{bmatrix} 88.12 \\ 413.1 \\ 1000 \end{bmatrix}$$

The desired pose was :

$$X = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 88.1 \\ 413.1 \\ 1000 \end{bmatrix}$$

The figure 5.25 shows that despite the noise introduced in the measurement of orientation vectors, the desired end-effector pose was reached with error even less as compared to the case of controlling the leg orientations.

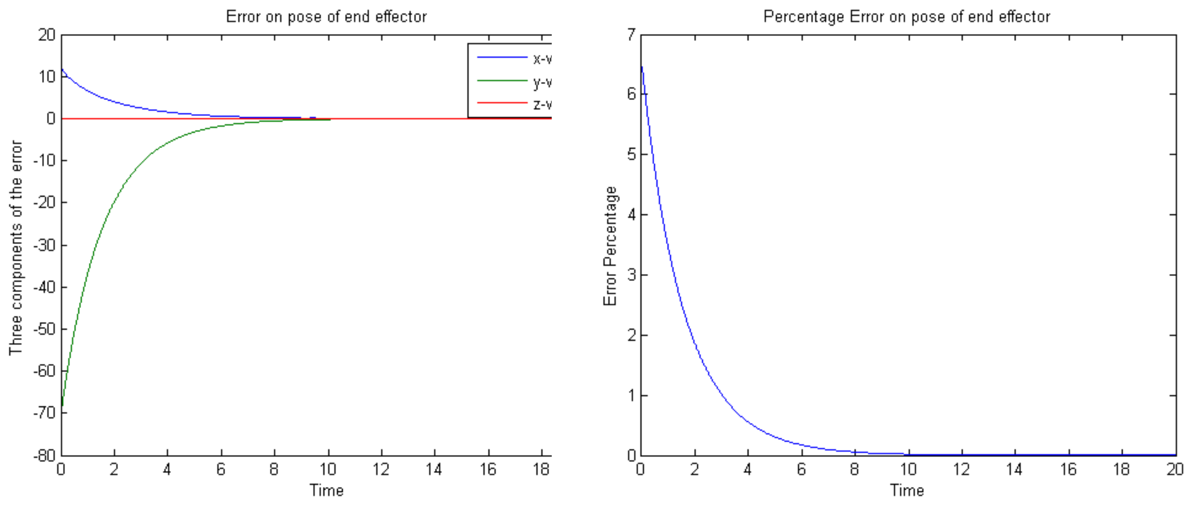


Figure 5.25: Error on pose of end-effector

We can see that the percentage error on the end-effector pose is almost equal to zero in the case of controlling the edges rather than the orientation.

5.5 Practical Edge detection on robot

5.5.1 Practical setup

For testing the algorithms developed, we have a practical scenario. The robot is IRSBot-2 available in IRCCyN. The camera used was monochrome Mikrotron 4CXP of type MC4082. THe lens is of 12mm focal length. THe maximum frame rate of this camera is 563 fps. The camera was in eye-to-hand configuration. A set of images were captured using this camera and then the edge detection algorithms were run to find the edges in those images. Once we have edges, we can test our algorithms. Since we need the camera to be calibrated in order to use our algorithms, so next section describes the steps followed for calibration and the matrices obtained.

5.5.2 Camera calibration

For calibration purposes, the camera calibrator application of MATLAB was used. A checkerboard was used and about 20 images of the checkerboard were taken by the camera. The images were loaded in the camera calibrator app of the MATLAB. Then it was provided with the size of the square on checkerboard. After that MATLAB will reject some images that were not good (either too blurry, checkerboard at bad angle etc). Then, MATLAB will return the detected corners of the checkerboard pattern and i went through each image in order to make sure that corners were correctly detected. Then there is option of selecting 2 coefficients or 3 coefficients for noise. I chose 2 coefficients and clicked on 'Calibrate'. The figure 5.26 shows some of the images used during the camera calibration.

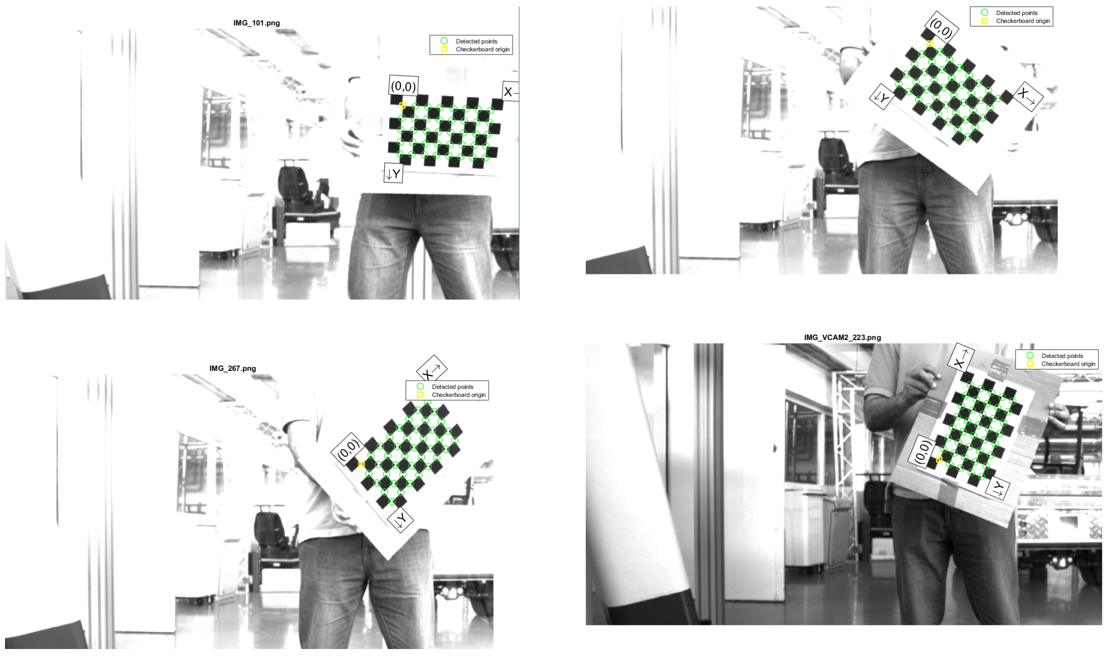


Figure 5.26: Process of Camera Calibration

After the calibration process is done, matlab gives us the camera parameters and provides information about the reprojection errors. In the figure 5.27 we can see the reprojection error (how well the detected points measured to the actual points in pixels). These reprojection errors are quite low, which means the calibration was good.

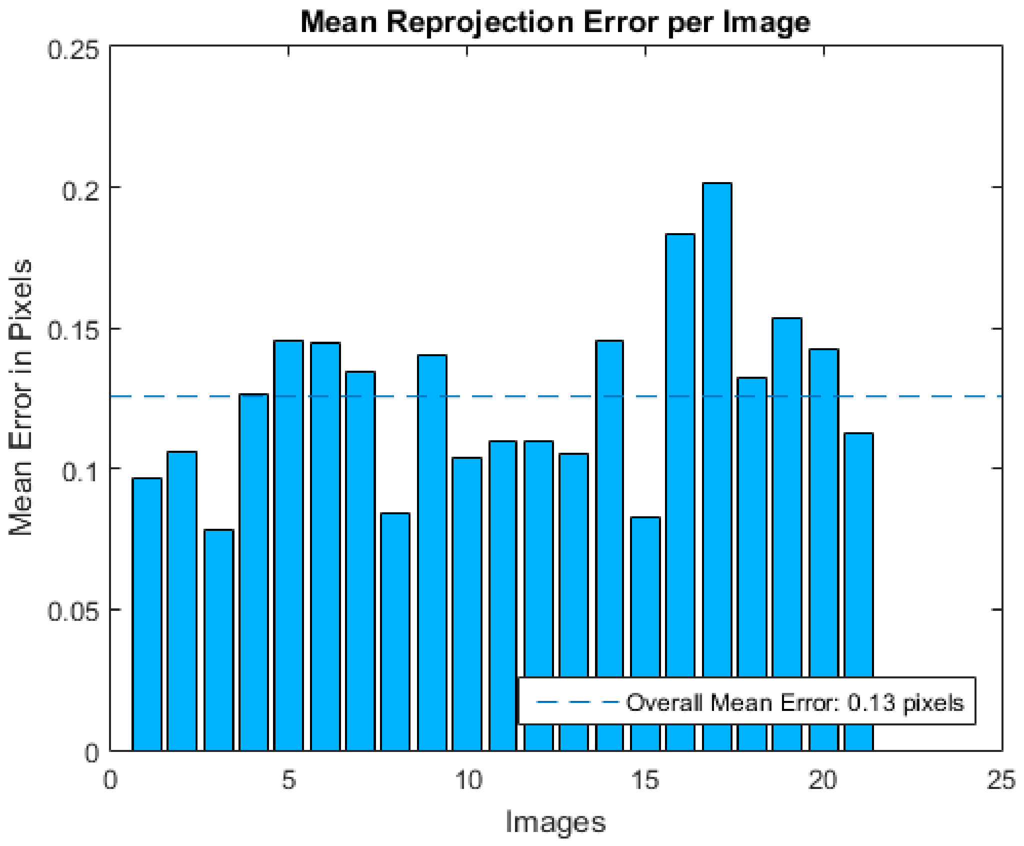


Figure 5.27: Reprojection error in pixels

The figure 5.28 shows a 3D visualization of the extrinsic relation between the positioning of camera and different images of the checker board.

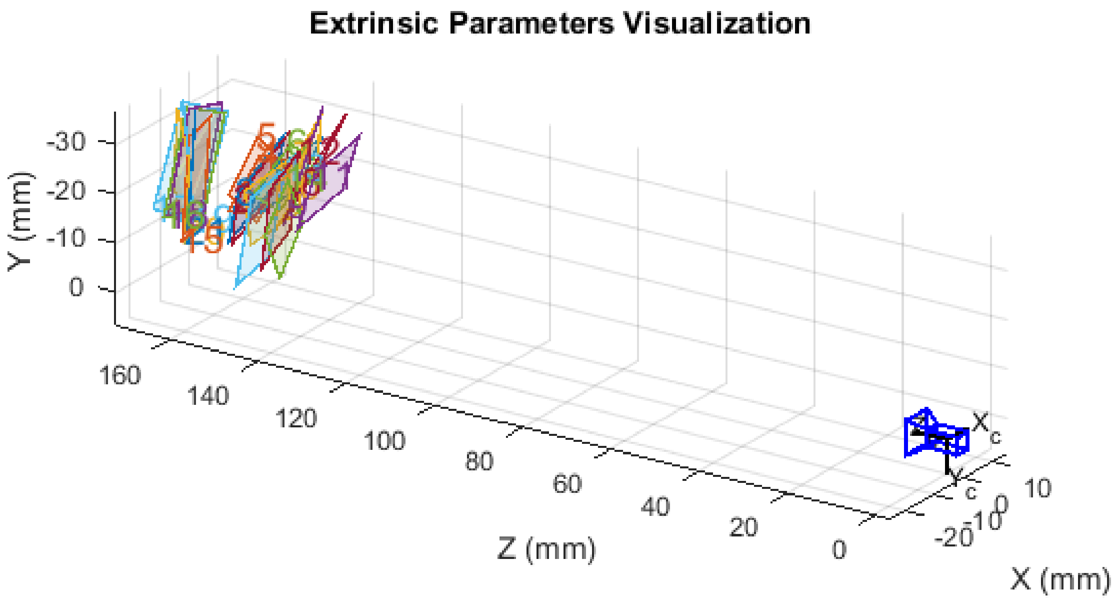


Figure 5.28: Extrinsic visualization

Next, we click on "export camera parameters" (this will be imported into MATLAB workspace). Then, we can go to MATLAB workspace and find cameraParameters file. Here, we can find the intrinsic parameters and the radial distortions needed to undistort an image. The values returned by MATLAB were:

$$K = \begin{bmatrix} f_u & \gamma & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1812.3 & 0 & 1231.4 \\ 0 & 1817.5 & 497.42 \\ 0 & 0 & 1 \end{bmatrix}$$

The radial distortion parameters determined were:

$$\text{Radial distortion} = \begin{bmatrix} k_1 \\ k_2 \\ k_3 \end{bmatrix} = \begin{bmatrix} -0.0801 \\ 0.1442 \\ 0 \end{bmatrix}$$

5.5.3 Technique used for edge detection

Since the full image was quite big (in terms of data) so first it was cropped to obtain two regions of interest. Each region of interest contained one leg, as shown in figure 5.30.

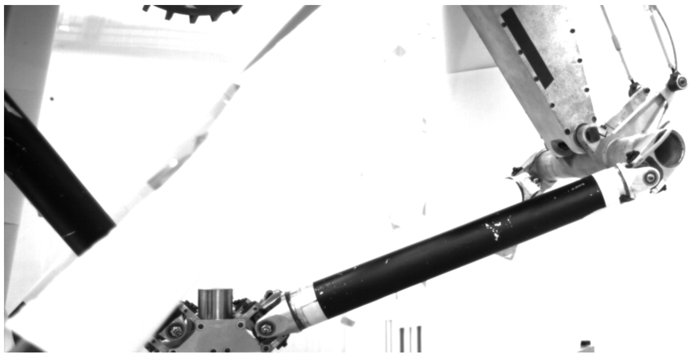


Figure 5.29: Original Image from Camera

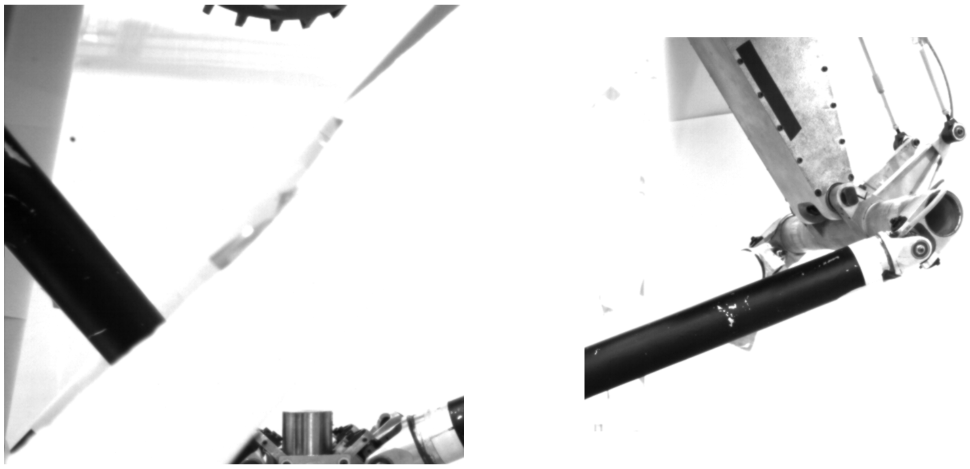


Figure 5.30: Sub Images from the original

Then on each image, the edge detection was applied simultaneously. Since we had a MATLAB model of the robot, so Computer vision toolbox of MATLAB was used to do image processing. For the detection of edges, an edge detection block was used, which is available in computer vision toolbox. Initially, the technique selected was using the Sobel filter, although this could be changed later. The basic idea in Sobel filter is to determine the approximations of derivatives in horizontal and vertical direction. For doing so, it uses a 3×3 Kernel and convolves it with a given image. This can be mathematically represented as follows:

$$M_1 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad M_2 = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

If we define I as the source image, and Gx and Gy as two images which at each point contain the horizontal and vertical derivative approximations respectively, the computations can be represented as follows:

$$G_x = M_1 * I, \quad G_y = M_2 * I$$

Here '*' operator denotes the 2-dimensional signal processing convolution operation.At each point in the image, the resulting gradient approximations can be combined to give the gradient magnitude, using:

$$G = \sqrt{G_x^2 + G_y^2}$$

Using this information, the gradient's direction is calculated as:

$$\theta = \arctan \frac{G_y}{G_x}$$

The images after the edge detection are shown in figure 5.31.

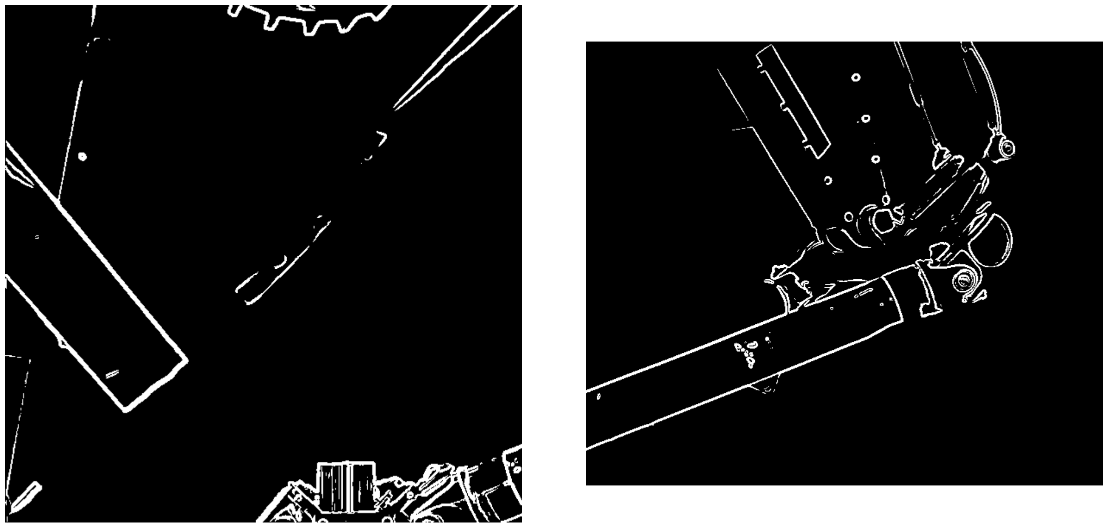


Figure 5.31: Edge detection

After the edge detection, there were more edges detected than we needed. So in order to extract the lines of our interest, some algorithm needs to be used. In this case, a hough transform block was used. The Hough Transform block computes the Hough matrix by transforming the input image into the rho-theta parameter space. The block also outputs the rho and theta values associated with the Hough matrix. The block parameters 'rho resolution' and 'theta resolution' were tuned in order to get the desired lines.
Then, The Find Local Maxima block was used. This block finds the location of the maximum value in the Hough matrix. The Maximum number of local maxima was set equal to 2 , since we needed to find two lines in a single image. A threshold value was also specified, which had to be tuned in order to get the desired results. Once the correct lines detected, a Hough lines block was used. The Hough Lines block determines where the given line intersects the edges of the original image. So this provides us with cartesian coordinates of the points where the intersection occurs. Using this information, we can draw the obtained lines on the original image, for viewing.This process was done for both the sub-images obtained from one single image by cropping it out, as described previously. This is shown in figure 5.32.

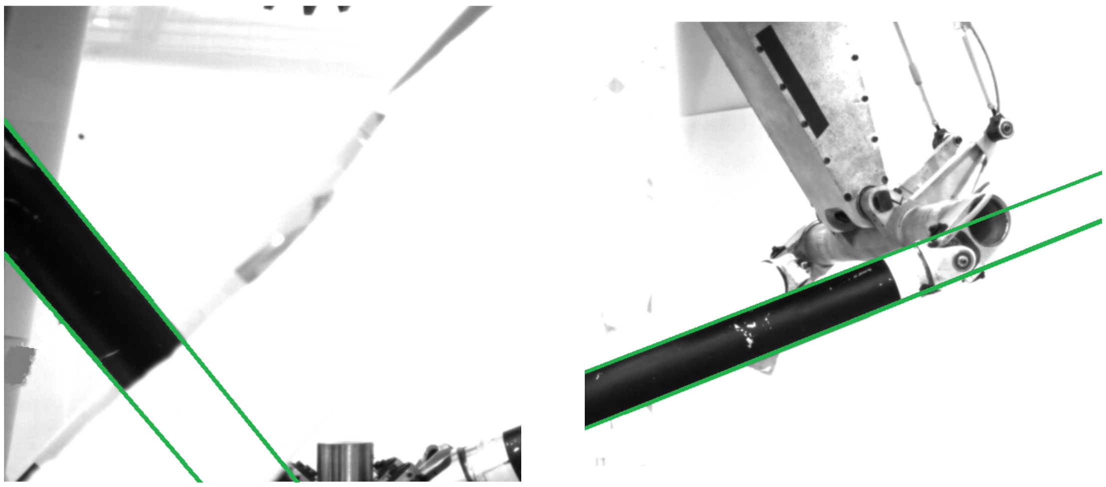


Figure 5.32: Sobel Edge detection

Finally the projection line vectors for each leg can be found from edges, as shown in figure 5.33.

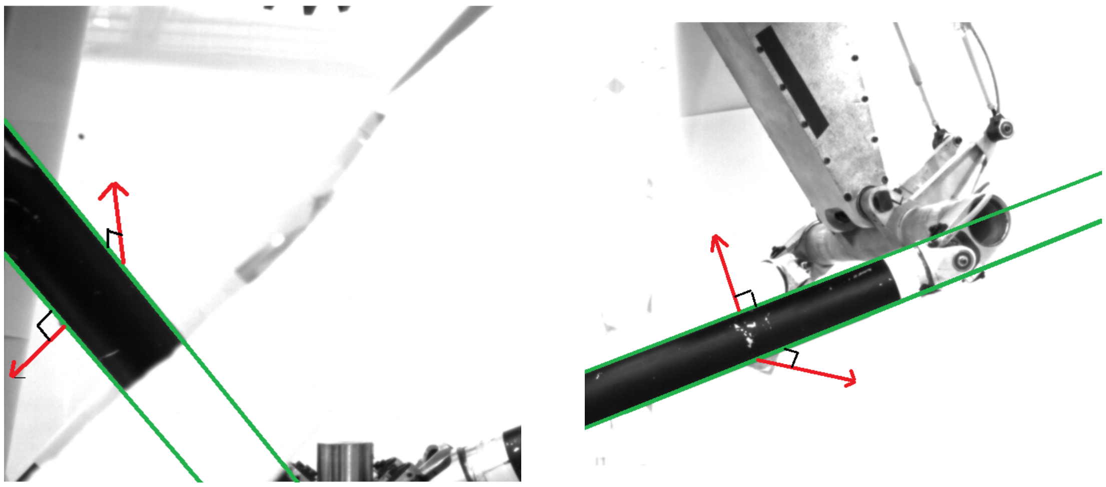


Figure 5.33: Edges (green) and projection-line vectors (red)

Part 3

Chapter 6

Conclusion and Future work proposal

In the previous chapters, state of the art in robotics,vision and control has been briefly covered. Different techniques to estimate the pose and velocity for parallel robots have been analysed and different control algorithms in the literature were covered.
Then the work done in the context of this thesis has been presented in detail. It has been shown that it is possible to control this robot using 2 out of the 4 legs of the robot. The methods using the leg orientations and leg edges have been investigated and validated in simulations.
Secondly, the way has been paved for practically testing the algorithms on robot. The edge detection has been performed on images obtained through a calibrated camera.

Future Recommendations

For building the future work based on this thesis, following points could be investigated:

- Practically testing the algorithms on the robot
- Improving the control algorithms by introducing the dynamic control while taking into account the high speed of robot

Chapter 7

Bibliography

[1] https://en.wikipedia.org/wiki/Radar

[2] Flavien Paccot, Philippe Lemoine, Nicolas Andreff, Damien Chablat and Philippe Martinet. A vision-based computed torque control for parallel kinematic machines. In IEEE International Conference on Robotics and Automation, 2008.

[3] Redwan Dahmouche, Nicolas Andreff, Youcef Mezouar, Philippe Martinet, LASMEA, CNRS, Université Blaise Pascal. 3d pose and velocity visual tracking based on sequential region of interest acquisition. In The 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems October 11-15, 2009 St. Louis, USA. IEEE, 2009.

[4] Optimal Kinematic Design of Robots by Mr Philippe Wenger,EMARO II Course, Ecole Centrale de Nantes

[5] Victor Rosenzvig, Sensor Based design and control of high-speed manipulators, PhD Thesis 2015, IRCCyN,Ecole Centrale de Nantes,Nantes

[6] P.Srinivas,K.Vijaya Lakshmi,V.Naveen Kumar.A comparison of PID Controller Tuning Methods for three tank level process. *International Journal of Advanced Research in Electrical,Electronics and Instrumentation Engineering,January 2014*

[7] Modelling and Control of Manipulator Robots by Professor Philippe Martinet,EMARO-ARIA Master 1,Ecole Centrale de

Nantes. Slides available at http://www.irccyn.ec-nantes.fr/~martinet/MoCom.html

[8] Flavien Paccot ,Nicolas Andreff, Philippe Martinet.A review on dynamic control of parallel kinematic machines:Theory and experiments. The International Journal of Robotics Research

[9] Chaumette, F. and Hutchinson, S. (2008). Handbook of Robotics - Visual Servoing and Visual Tracking pages 563-583

[10] Slides of Computer Vision course by Professor Philippe Martinet . Slides can be found at http://www.irccyn.ec-nantes.fr/~martinet/

[11] Daniel F. DeMenthon and Larry S. Davis. Model-based object pose in 25 lines of code. International Journal of Computer Vision, 15:123{141, 1995

[12] J.A. Gangloff and M.F. de Mathelin. High speed visual servoing of a 6 DOF manipulator using multivariable predictive control,October 7,2003

[13] Seth Hutchinson, Gregory D Hager, Peter I. Corke. A tutorial on visual servo control.IEEE transactions on Robotics and Automation,Volume 12, No.5, October 1996

[14] D. Stewart. A platform with six degrees of freedom. Proceedings of the Institution of Mechanical Engineers

[15] N. Andreff, A. Marchadier, and P. Martinet. Vision-based control of a Gough-Stewart parallel mechanism using legs observation. In Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on, pages 2535{2540, april 2005. doi: 10.1109/ROBOT.2005.1570494

[16] Nicolas Andref, Tej Dallej, Philippe Martinet. Image-based Visual Servoing of a Gough- Stewart Parallel Manipulator using Leg Observations. International Journal of Robotics Re- search. Special Issue on Vision and Robotics - Joint with the International Journal of Computer Vision, 2007, 26 (7), pp.677-687. <hal-00520165>

[17] Slides of Vision based control course by Professor Philippe Martinet . Slides can be found at http://www.irccyn.ec-nantes.fr/~martinet/VisionBasedControl.html

[18] S. Briot and P. Martinet. "Minimal Representation for the Control of Gough-Stewart Platforms via Leg Observation Considering a Hidden Robot Model", Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA 2013), May 6-10, 2013, Karlsruhe, Germany

[19] Redwan Dahmouche, Nicolas Andreff, Youcef Mezouar and Philippe Martinet.Efficient High-speed Vision-based Computed Torque Control of the Orthoglide Parallel Robot. 2010 IEEE International Conference on Robotics and Automation Anchorage Convention District May 3-8, 2010, Anchorage, Alaska, USA

[20] Tej Dallej, Nicolas Andreff and Philippe Martinet. Image-Based Visual Servoing of the I4R parallel robot without Proprioceptive Sensors. 2007 IEEE International Conference on Robotics and Automation Roma, Italy, $10 - 14$ April 2007

[21] N. Andreff, B. Espiau, and R. Horaud. Visual servoing from lines. International Journal of Robotics Research, 21(8): 679–700, 2002

[22] J. Plücker. On a new geometry of space. Philosophical Transactions of the Royal Society of London, 155:725–791, 1865.

[23] J.P. Merlet. Parallel Robots. Springer, 2nd edition, 2006.

[24] Coralie Germain. Conception d'un robot paralléle à deux degrés de liberté pour des opérations de prise et de dépose. Automatique / Robotique. Ecole Centrale de Nantes, 2013. Français. <tel-01108739>

[25] Erol Ozgur. From lines to dynamics of parallel robots. Other. Université Blaise Pascal, Clermont-Ferrand II, 2012

[26] Alessia Vignolo, Master Thesis(2014). Visual servoing of the Monash Epicyclic-parallel manipulator,Ecole Centrale de Nantes.

[27] Giovanni Claudio, Master Thesis(2013). Pose and velocity estimation for high speed robot control, Ecole Centrale de Nantes.

[28] Redwan Dahmouche, Thése de doctorat. Contributions à l'estimation de mouvement 3D et à la commande par vision rapide, Université Blaise Pascal - Clermont II.

[29] Tej Dallej, Thése de doctorat. Contributions à un modéle générique pour l'asservissement visuel des robots paralléles par observation des élements cinématiques,Université Blaise Pascal - Clermont II.

[30] Flavien Paccot, Thése de doctorat. Contributions à la commande dynamique référencée capteur de robots paralléles, Université Blaise Pascal - Clermont II.