ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/325260165

Cable-Driven Parallel Robot Simulation Using Gazebo and ROS

Chapter · January 2019

DOI: 10.1007/978-3-319-78963-7_37

CITATIONS
12

READS
9,358

4 authors:

Franklin Okoli
École Centrale de Nantes
20 PUBLICATIONS **24** CITATIONS
SEE PROFILE

Yuchuan Lang
2 PUBLICATIONS **13** CITATIONS
SEE PROFILE

Olivier Kermorgant
École Centrale de Nantes
42 PUBLICATIONS **580** CITATIONS
SEE PROFILE

Stéphane Caro
CNRS - Laboratoire des Sciences du Numérique de Nantes (UMR 6004)
367 PUBLICATIONS **5,669** CITATIONS
SEE PROFILE

All content following this page was uploaded by Franklin Okoli on 28 June 2018.

The user has requested enhancement of the downloaded file.

Ecole Centrale de Nantes

Development of a Dynamic Simulator for Cable-driven Parallel Robots

Master Thesis Final Report
Master Automatique, Robotique et Informatique Appliqueé

Presented by:
Franklin OKOLI

Supervisors:
Stephane Caro
Olivier Kermorgant

July 26, 2016

Abstract

This report aims to describe the steps taken and the tools used in the development of a simulator for cable driven parallel robots. We make a recall on cable driven parallel robots, their static and dynamic properties. We also make a further study of all robot simulation software that are available today, in particular, Gazebo and how it can be integrated with ROS to achieve our objectives of developing a simulator for cable-driven parallel robots that can also work with vision and other sensors. Then we make a comparison of the advantages we hope to gain from developing a simulator in gazebo to other existing software. In particular, we show how the simulated cable driven parallel robot is obtained in gazebo, and how it is controlled within the ROS architecture using the different tension distribution algorithms. We will also show some results obtained using this simulator and proffer some ideas for future development of this subject.

Contents

List of Figures

Abbreviations

ROS - Robot Operating System

CDPM - Cable-driven Parallel Manipulator

CDPR - Cable-driven Parallel Robot

MATLAB - Matrix Laboratory

PID - Proportional-Derivative-Integral Controller

3D - Three Dimensional Space

2D - Two Dimensional Space

OSRF - Open Source Robotics Foundation

GUI - Graphical User Interface

SDF - Simulation Description Format

XML - Extensible Markup Language

URDF - Universal Robot Description Format

LTS - Long Time Support

RAM - Random Access Memory

CPU - Central Processing Unit

EU -FP7 - European Union Seventh Framework Program

PID - Proportional-Derivative-Integral Controller

FC - Force Closure Workspace

WFW - Wrench Feasible Workspace

WCW - Wrench Closure Workspace

IFCOW - Interference Free Constant Orientation Workspace

DARPA - Defence Advanced Research Projects Agency

YAML - YAML Ain't Markup Language

GPS - Global Positioning System

ODE - Open Dynamics Engine

Chapter 1

Introduction

The study of parallel robots has been one of the active fields of robotics research for several decade. [10].Parallel Robots present several advantages such as high accuracy and high accelerations and a large payload-to-weight. They usually consist of a number of links fixed to a base and with the other end attached to a common end effector.

Cable driven parallel robots are an extension of the idea of parallel robots by a combination of the principles of parallel robots with the properties of cables. In other words, the rigid links of the parallel robot has been replaced with a cable under tension to form a cable driven parallel robot. CDPMs come with the advantage of having negligible inertia, lighter and fewer mechanical components to give a lighter assembly, larger workspace than conventional parallel robots.

An important stage in the process design of cable driven parallel robots is the simulation phase. Simulators allow us to quickly test a new idea and see how this implementation fits our design. Design of robots make use of simulators test how the robot will operate under certain conditions before we begin the expensive journey of building an actual robot. Also it can be quite dangerous for robots if the designer decides to deploy a new algorithm which has not been tested in simulation. The robot can be damaged if something goes wrong or if some parameters had not been taken into account.

Having a good visual representation of the robot's workspace is also quite interesting to show the capabilities of a robot to an audience who could be interested in investing in such a project or to an audience who want to learn about these robots. Simulating interactions between this robot and the environment or between this robot and objects also help us to study these robots and really test them to the limits.

Our focus on this study is to look at existing work that has been done to implement simulators for cable driven parallel robots and how we can use Gazebo to implement a simulator for a Cable driven parallel robot. Gazebo presents powerful tools that we can use to achieve this. But currently, there is no existing cable driven parallel robot simulator that has been implemented in Gazebo at the time of this publication to the best of our knowledge. Thus this gap presents an opportunity for us to define a simulator that can be used in the teaching and research of cable driven parallel robots by the community.

1.1 Literature Review

The earliest built CDPM was NIST ROBOCRANE [11] Another early application is seen in SKYCAM [12], where cables are used to move a camera over a football field. The advantage here is the large workspace covered by the CDPM and that there is no interference of the view of the spectators by the cable due to their small size.

CDPMs have also been employed in construction[13] and medical rehabilitation [14]. A system for teleoperation [15] has been proposed based on cable driven parallel robots and other for cable system for space-craft application is seen in [16].

Dynamic Workspace has been defined by the authors in [17] while Force Closure Workspace (IFCOW) has been defined by [18]. The set of wrenches that the CDPM is able to apply on the environment while maintaining a given pose given force limits imposed on actuator for a given actuator was studied in [19] and [20].

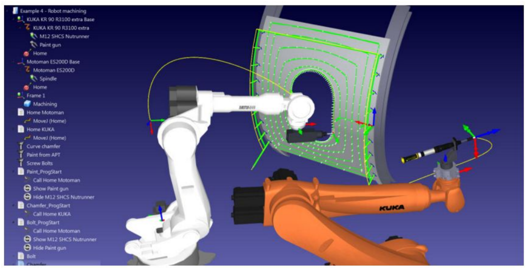
Wrench Closure Workspace WCW is derived from stiemke's theorem in [21] and implemented numerically by interval analysis in [22] [23] [24].

Static Equilibrium conditions for CDPMs are presented in [25] Here we see that an over constrained configuration is important especially in applications where there exists no gravity. This system having more cables than the number of DOF of the moving platform is solved by linear programming as shown in [26] and [27].

Control of CDPMs might be complex due to actuation redundancy. Methods to measure the tension in the cables were proposed in [28] and a controller based on the cable tension was proposed in [29]. Sliding mode control has been applied to CDPMs in [30] .

Visual Based servoing has been used to control a CDPR in [31]. Here a CDPM called Marionet-Assist, a mobility assist device which has 3-DOF translation with a camera at its end effector is controlled to grasp a target and move it from one location to another. This control scheme uses photometric moments to ensure a robust control.

Several open-source and licensed Robot simulators exist [32] [1] including Stage, Gazebo, V-Rep, MS-Robotics studio, Robologix, Labview,MATLAB to name a few which are able to recreate the behavior of a robot in software.


(a) Industrial Machine Simulation using Robo DK software

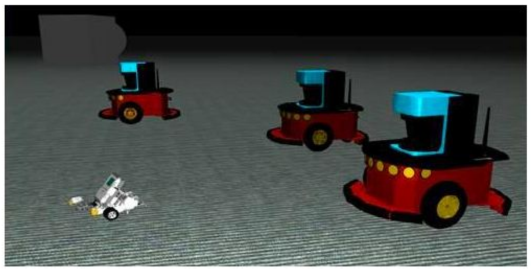

(b) Mobile Robots Simulated in Microsoft Robot Development Studio

Figure 1.1: Examples of Robot Simulation in Software [1]

The author in [33] has studied the use of a CDPR for the use in Algae Harvesting by performing a simulation to choose the design parameters for such a system. This approach intends to solve the workspace problem of large scale commercial farming of and he performs the simulation of such a system using MATLAB.

Another simulator which has been developed for cable driven parallel robots is *ARACHNIS* [5]. Here, the authors have developed a CDPM simulator using MATLAB for the Analysis of CDPMs. This software is able to generate the WFW Wrench Feasible Workspace and Interference Free Constant Orientation Workspace (IFCOW) of CDPMs for both spatial and planar mechanisms. This software is able to greatly reduce the computation time for the WFW and IFCOW of the cable driven parallel robot which can be quite computationally expensive. But this simulator does not animate the motion of such a mechanism within this workspace or show the structure of the mechanism in an indoor or outdoor environment or take into account the ability to extend the simulation using sensors and interaction with real world objects. A dynamic simulator for autonomous underwater vehicles has been developed based on gazebo, integrated with ROS and rendered using underwater image rendering software called UWSim [34]. The authors have developed plugins that implements the model of the robot, a model for the environment which is underwater and a model for the PID controller. There exists some implementation of CDPMs simulators [35] [2] which are developed using MATLAB/Simulink.

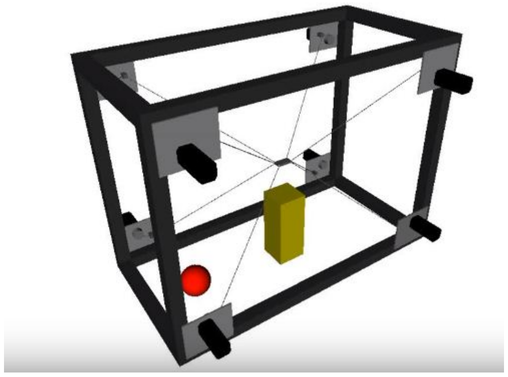


Figure 1.2: Cable Driven Parallel Robot 3D Simulator developed by means of MATLAB/SIMULINK [2]

The authors in [36] have made a study among researchers to determine the most preferred choice of software for dynamic simulation and reasons for this choice. From the results of this study, we see that Gazebo is the most preferred software for simulation amongst researchers.

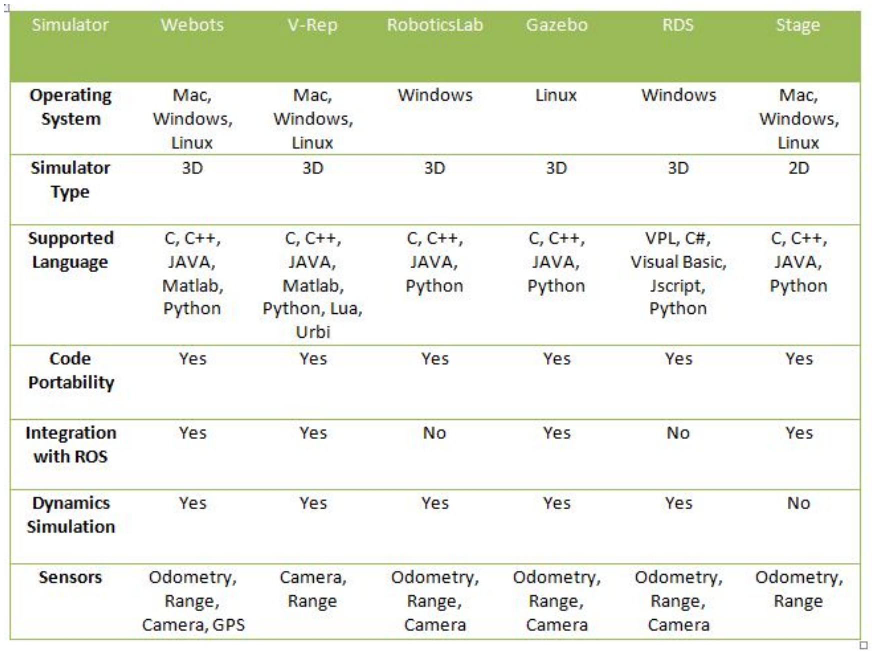
Some of the requirements for a robotic simulator are:

1. The supported operating system on where the simulator can run.
2. Ability to perform 2D or 3D simulation.
3. Supported Programming languages.
4. Code Portability where we look at how easily we can deploy such a code directly on a real robot.
5. Sensor and sensor support.

6. Ability to perform dynamic simulations using rigid body dynamics collisions, obstacles etc.

7. Integration with ROS.

Below we can see a comparison of the capabailities of these software



Simulator	Webots	V-Rep	RoboticsLab	Gazebo	RDS	Stage
Operating System	Mac, Windows, Linux	Mac, Windows, Linux	Windows	Linux	Windows	Mac, Windows, Linux
Simulator Type	3D	3D	3D	3D	3D	2D
Supported Language	C, C++, JAVA, Matlab, Python	C, C++, JAVA, Matlab, Python, Lua, Urbi	C, C++, JAVA, Python	C, C++, JAVA, Python	VPL, C#, Visual Basic, Jscript, Python	C, C++, JAVA, Python
Code Portability	Yes	Yes	Yes	Yes	Yes	Yes
Integration with ROS	Yes	Yes	No	Yes	No	Yes
Dynamics Simulation	Yes	Yes	Yes	Yes	Yes	No
Sensors	Odometry, Range, Camera, GPS	Camera, Range	Odometry, Range, Camera	Odometry, Range, Camera	Odometry, Range, Camera	Odometry, Range

Figure 1.3: Comparison of existing Robot Software

These come from the desired criteria such as: a simulation very close to reality, open-source software, ability to use same code for real and simulated robot, light and fast simulator. This study has indeed confirmed our choice of Gazebo to perform this simulation.

A simulator for CoGiRo, a CDPM, was developed in [35] as part of the EU-FP7 CableBot project. The authors make use of a 3D CAD software to model the component parts of the CDPR such as winches, pulleys, cables and cable fastening, then the parts are imported to another software called XDE where the simulation is to be performed and assembled as a complete CDPR. This robot assembly is then controlled using Matlab/Simulink interface that implements motion generation, inverse kinematics and PID control of the robot's cartesian platform position.

This thesis report will study:

- The Modeling of Cable Driven Parallel robots.
- The use of Gazebo in performing Simulation.
- Interfacing our Gazebo simulated robot with ROS.
- Interfacing our Gazebo simulated robot with Matlab.
- The dynamic control and tension distribution of cable driven parallel robots.

Chapter 2

Modeling of Cable Driven Parallel Robots

In this chapter we discuss the current approach used to carry out the modeling a cable driven parallel robot which has been dealt with in several works of study such as [10] [3] [37], [25], [17].

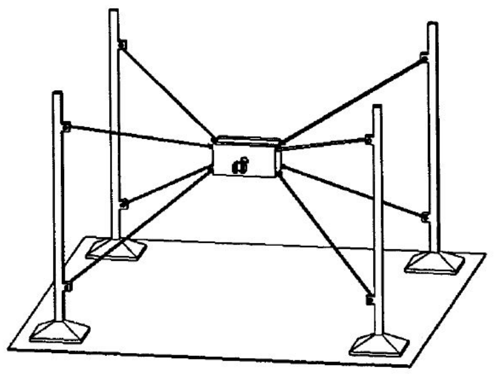
A cable driven parallel manipulator usually consists of a number of cables, which are connected to fixed points and the other end point of these cables are attached to the common end effector. The position and orientation of the end effector is controlled by increase or decrease of the cable lengths achieved by rolling of the spool that holds the total cable length.

If we denote the number of cables that make up the CDPM as m and the number of degrees of freedom of the end-effector as n, then three cases arise [38]:

Case 1 : Incompletely Restrained CDPM - In this case, at least one equation of constraint is still required to determine the pose of the end-effector in any configuration.

Case 2 : Completely Restrained CDPM - In this case $m = n + 1$, where the pose of the robot is fully determined by the unilateral kinematic constraints imposed by the cable.

Case 3 : Redundantly Restrained CDPM - In this case $m > n + 1$, where the pose of the robot has more unilateral kinematic constraints imposed on it by the cables than required.


(a) An Example of a Redundantly Constrained CDPM [25]

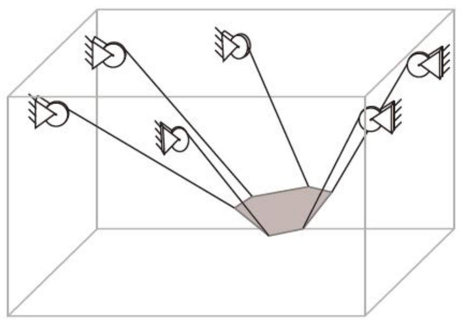

(b) An Example of a Compeletely Constrained CDPM [10]

Figure 2.1: Examples of Cable Driven Parallel Manipulators

From the following, we see that we based on the difference between the number of wires m and number of the degrees of freedom of the end-effector n, the CDPM will have a redundancy given as $r = m - n$.

We note that in both cases, there is an effect on the static and dynamic equilibrium of the CDPM depending on the number of cables used to define the configuration.

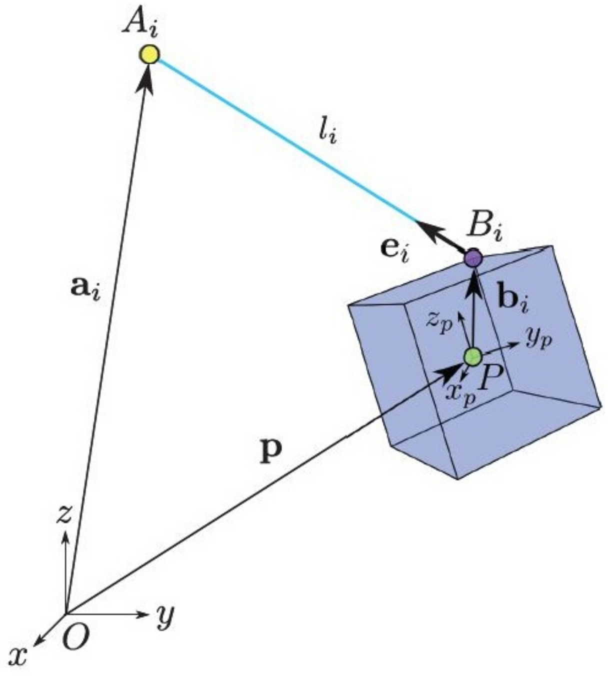


Figure 2.2: Geometric Description of a single cable from Base to End effector of a CDPM [3]

2.1 Geometric Description of CDPMs

A single connection of a CPDM from the fixed base to the end-effector point is described geometrically in 2.2.

From the figure point A_i defines the point on the base where cable i is attached while point B_i defines the point on the end effector where the cable i is attached. From this, we also define the co-ordinates of point A_i to be a vector $\mathbf{a}_i$ which is defined in the absolute/reference frame $\mathfrak{R}_\mathfrak{a} = \{O, x, y, z\}$.

Usually the points of attachment on the base of the CDPM are known points in the room, thus the absolute frame will be the frame of the room. We also denote the co-ordinates of point B_i using the vector $\mathbf{b}_i$ which is defined in the local frame of the end-effector $\mathfrak{R}_\mathfrak{p} = \{P, x_p, y_p, z_p\}$. From this, we can see that the instantaneous length of the cable i will be given by the relation:

The position and orientation of the end-effector at point p in the global frame can be described by a pose vector $\mathbf{x} = [x, y, z, \phi, \theta, \psi]^T$ for a spatial robot or $\mathbf{x} = [x, y, \theta]^T$ for a planar robot.

The Homogeneous Transformation matrix between the Global frame and the End-effector frames in the spatial case is given as:

$$\mathbf{T} = \begin{bmatrix} c\theta c\psi & -c\theta s\psi & s\theta & x \\ c\phi s\psi + s\phi s\theta c\psi & c\phi c\psi - s\phi s\theta s\psi & -s\phi c\theta & y \\ s\phi s\psi - c\phi s\theta c\psi & s\phi c\psi + c\phi s\theta s\psi & c\phi c\theta & z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_{3x3} & \mathbf{c}_{3x1} \\ \mathbf{0}_{1x3} & 1 \end{bmatrix} \tag{2.1}$$

whereas in the planar case:

$$\mathbf{T} = \begin{bmatrix} c\theta & -s\theta & x \\ s\theta & c\theta & y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_{2X2} & \mathbf{c}_{2x1} \\ \mathbf{0}_{1x2} & 1 \end{bmatrix} \tag{2.2}$$

In both cases, matrix, $\mathbf{Q}$ defines the rotation matrix and vector $\mathbf{c}$ is a vector of the position of the center of the end-effector.

The orientation in spatial case is represented by ϕ, θ, ψ given by the Euler angles obtained first by a rotation ϕ about the z-axis, followed by a rotation θ about the $x-axis$ and finally a rotation of ψ around the $y-axis$.

In the planar case, the angle θ is given by the angle between the $x-axis$ of the robot frame and the $x-axis$ of the frame attached to the end effector

2.2 Inverse Kinematics

From the geometric description that we have given in the previous section, we see that the length of the cable at any instant is given as

$$\mathbf{l}_i = \mathbf{a}_i - \mathbf{c} - \mathbf{Q}\mathbf{b}_i \tag{2.3}$$

The length of the cable i is the norm of $\mathbf{l}_i$

$$l_i = \|\mathbf{l}_i\|_2 \quad 1 \leq i \leq m \tag{2.4}$$

The vector $\mathbf{l}$ is a vector of cable lengths:

$$\mathbf{l} = [l_1 \ l_2 \ ... \ l_m]^T \tag{2.5}$$

The Inverse kinematics of a cable-driven parallel manipulator gives the relationship between the speed of the cables $\dot{\mathbf{l}}$ and the derivatives of the platform co-ordinates $\dot{\mathbf{x}}$.

$$\dot{\mathbf{l}} = \mathbf{J}\dot{\mathbf{x}} \tag{2.6}$$

This relationship is called the Jacobian matrix of the robot. where

$$\mathbf{J}_{mxn} = \begin{bmatrix} -\mathbf{d}_1^T & (\mathbf{d}_1 \text{ x } \mathbf{Q}\mathbf{b}_i)^T \\ -\mathbf{d}_2^T & (\mathbf{d}_2 \text{ x } \mathbf{Q}\mathbf{b}_2)^T \\ \vdots & \vdots \\ -\mathbf{d}_n^T & (\mathbf{d}_n \text{ x } \mathbf{Q}\mathbf{b}_n) \end{bmatrix} \tag{2.7}$$

$$\dot{l} = [\dot{l_1}\ \dot{l_2}\ ...\ \dot{l_n}]^T \tag{2.8}$$

where : $\mathbf{d}_i$ is the normalized vector of the cable length which is given as

$$\mathbf{d}_i = \frac{\mathbf{l}_i}{\|\mathbf{l}_i\|_2} \tag{2.9}$$

2.3 Static Equilibrium

The static equilibrium of a cable-driven parallel manipulator is closely related to its inverse kinematics [25]. This relationship makes use of the transpose of its Kinematic Jacobian to write:

$$\mathbf{Wt} + \mathbf{f}_p = \mathbf{0} \tag{2.10}$$

The proof of this can be derived from the principle of virtual work as shown in [3]. Where: $\mathbf{t}$ represents the mx1 vector of positive tension in the cables of the parallel robot.

$$\mathbf{t} = \begin{bmatrix} t_1 & t_2 & \cdots & t_m \end{bmatrix}^T \tag{2.11}$$

This value is usually limited within a mechanical range conforming to the maximum load of the cable and the elastic limit beyond which the cable can no longer hold.

$$t_{min} \leq t_i \leq t_{max} \tag{2.12}$$

Another condition is that that the vector of cable tensions only admits positive values and must be non-negative

$$t_i > 0 \tag{2.13}$$

$\mathbf{f}_p$ represents the represents the nx1 vector forces and moments acting on the end-effector of the CDPM.

$$\mathbf{f}_p = [\mathbf{f}^T\ \mathbf{m}^T]^T \tag{2.14}$$

This vector could be composed of gravitational forces which is acting on the robot end effector or other external contact forces that act on the end effector.

$$\mathbf{W} = -\mathbf{J}^T \tag{2.15}$$

This matrix is used to study the static equilibrium of the robot since by physical relations[25].

The meaning of (2.10) is that the sum of all wrenches acting on the CDPM must be equal to zero to achieve static equilibrium configuration. We will see that there exists some desired pose in the workspace for which the CDPM cannot achieve static equilibrium and this is major due to the conditioning of the wrench matrix at this pose.

The matrix $\mathbf{W}, nxm$ is popularly called the *wrench matrix* and is dependent on the pose of the cable- driven parallel robot. In a completely constrained cable-driven parallel manipulator, the wrench matrix is not square and the inverse solution is usually obtained by a generalized inverse written as:

$$\mathbf{W}^{+} = \mathbf{W}^{T}(\mathbf{W}\mathbf{W}^{T})^{-1} \tag{2.16}$$

or by minimization for the appropriate cable tensions $\mathbf{t}$ to hold the robot in the static configuration at a given pose while satisfying all the cable tension conditions [17] [25].

$$min \;\; \mathbf{t}^{T}\mathbf{t} \begin{cases} \mathbf{W}\mathbf{t} = \mathbf{f}_p & (2.10) \\ \mathbf{t}_i > 0 & (2.13) \end{cases}$$

2.4 Dynamics of Cable Driven Parallel Robot

We will consider the dynamic modeling of a CDPM whose cables are assumed to have negligible mass and are not elastic. With this assumption, the dynamics is reduced to only that of the end effector and is given as

$$\mathbf{M}\ddot{\mathbf{x}} + \mathbf{C}\dot{\mathbf{x}} + \mathbf{f}_g = \mathbf{W}\mathbf{t} \tag{2.17}$$

where : $\mathbf{M}$ is the Inertia matrix of the moving platform, $\mathbf{f}_g$ is a vector of the gravity terms,$\mathbf{C}\dot{\mathbf{x}}$ gives the Coriolis and centrifugal terms.

$$M = \begin{bmatrix} mI_p & \mathbf{0}_{pxq} \\ \mathbf{0}_{qxp} & \mathbf{K} \end{bmatrix} \tag{2.18}$$

where p = No of Translational DOF in the pose vector and q = No of rotational DOF of the pose vector, m is the mass of the moving platform, I_p denotes a p x p identity matrix. $\mathbf{f}_g = [0\ 0\ \ -mg\ 0\ 0\ 0]^T$

$$\mathbf{C}\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{0}_{n\text{x}1} \\ \omega\text{x}\mathbf{K}\omega \end{bmatrix} \tag{2.19}$$

The actuator dynamics is given as:

$$\mathbf{I}_m\ddot{\mathbf{q}} + \mathbf{f}_m(\dot{\mathbf{q}}) + \mathbf{R}\mathbf{t} = \tau_m \tag{2.20}$$

where $\mathbf{q}$: joint angles of the motor shaft, $\mathbf{I}_m$ m x m Diagonal Positive Inertia Matrix of the Actuator, $\mathbf{R} = diag[r_1,\ r_2,\ .\ .\ .,\ r_m]$ given that r is the radius of the pulley, $\mathbf{f}_a$ is the vector of cable tensions and τ_m is the m x 1 vector of motor torques, $\mathbf{f}_m$ is the m x 1 vector of friction for each of the motor. $\mathbf{K} = \mathbf{Q}\mathbf{I}_p\mathbf{Q}^T$ is the inertia matrix of the end-effector expressed at the reference point, P in the global frame with I_p being the inertia matrix of the end-effector expressed in the local end-effector frame.

$$\dot{\mathbf{q}} = \mathbf{R}^{-1}\dot{\mathbf{l}} \tag{2.21}$$

$$\dot{\mathbf{l}} = \mathbf{J}\dot{\mathbf{x}} \tag{2.22}$$

we can obtain the relationship between the torques in the motor and the end effector as below by substituting for the motor shaft velocity and differentiating with respect to time to obtain the motor joint acceleration.

$$\ddot{\mathbf{q}} = \mathbf{R}^{-1}\mathbf{J}\ddot{\mathbf{x}} + R^{-1}\dot{\mathbf{J}}\dot{\mathbf{x}} \tag{2.23}$$

substituting for $\mathbf{f}_a$ we finally obtain the full dynamic model as:

$$\mathbf{M}\ddot{\mathbf{x}} + \mathbf{C}\dot{\mathbf{x}} - \mathbf{f}_g = \mathbf{W}\mathbf{R}^{-1}(\tau_{\mathbf{m}} - (\mathbf{I}_m\ddot{\mathbf{q}} + \mathbf{f}_m(\dot{\mathbf{q}}))) \tag{2.24}$$

2.5 Stiffness of a Cable-Driven Parallel Robot.

A goal in the design of CDPM is to have a high stiffness such that the robot has a small displacement to an external wrench. The cable tensions and the cable stiffness contribute to the total stiffness of the CDPM and enable the robot to resist an external wrench applied on it.The stiffness Matrix of a CDPM provides a relationship between an external wrench acting on the robot and infinitesimal displacement of this robot [39].

In static equilibrium, an external wrench $\mathbf{f}_p$ acting instantaneously at the end effector, the instantaneous change in the pose d$\mathbf{p}$. It can be obtained by taking a derivative of $-\mathbf{f}_p$ with respect to the pose $\mathbf{p}$ and substituting the relationships (2.10) and (2.15).

$$\mathbf{K} = -\frac{d\mathbf{f}_p}{d\mathbf{p}} = \mathbf{J}^T\frac{d\mathbf{f}_a}{d\mathbf{p}} + \mathbf{f}_a\frac{d\mathbf{J}^T}{d\mathbf{p}} \tag{2.25}$$

From (2.22)

$$\frac{d\mathbf{f}_a}{d\mathbf{p}} = (\frac{d\mathbf{f}_a}{d\mathbf{l}})\frac{d\mathbf{l}}{d\mathbf{p}} = \mathbf{J}\frac{d\mathbf{f}_a}{d\mathbf{l}} \tag{2.26}$$

$$\mathbf{K} = \mathbf{J}^T\Omega\mathbf{J} + \mathbf{f}_a\frac{d\mathbf{J}^T}{d\mathbf{p}} \tag{2.27}$$

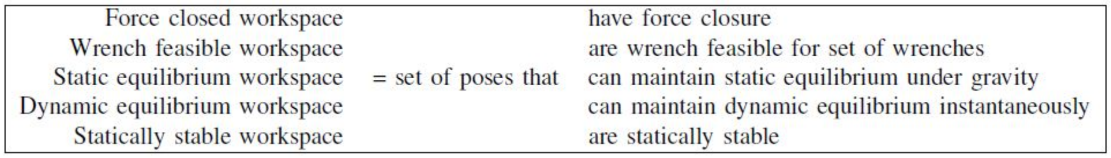
$$\Omega = \frac{d\mathbf{f}_a}{d\mathbf{l}} = diag(\ k_1,\ k_2, .\ .\ .\ k_m) \tag{2.28}$$

where k_i represents the individual stiffness of the cable i, it is a function of the elasticity of the cable

2.6 Workspace of CDPRs

The workspace of a cable driven parallel robot has been a subject of research from several authors [40] [23] [17] [4]. Cable-driven parallel robots generally have a large workspace since the length of the cables are limited by the dimensions of the room in theory. The application that we will use the robot will also require that we exert specific forces and moments at different parts of the workspace. The important workspaces of a cable-driven parallel robot are:

1. : The Wrench Feasible Workspace (WFW).
2. : The interference-free constant orientation workspace (IFCOW).
3. : Force Closure Workspace



Force closed workspace		have force closure
Wrench feasible workspace		are wrench feasible for set of wrenches
Static equilibrium workspace	= set of poses that	can maintain static equilibrium under gravity
Dynamic equilibrium workspace		can maintain dynamic equilibrium instantaneously
Statically stable workspace		are statically stable

Figure 2.3: Definitions of the different types of Workspaces of a CDPR [4]

2.6.1 The Wrench Feasible Workspace (WFW)

The wrench feasible workspace is a set of poses for a CDPR where a wrench or a set of required wrenches can be applied to the end-effector can be counteracted by tension forces in the cables. We note that the wrenches are always non-negative since the cables can apply forces to the end-effector in one direction and these wrenches lie within an admissible range since the cables have a stress limit.

Mathematically, WFW corresponds to every pose of the robot where (2.10) can be satisfied and thus is dependent on the wrench matrix W. This determines the types of wrenches that the robot applies to the environment and therefore determines the application for which the robot can be used.

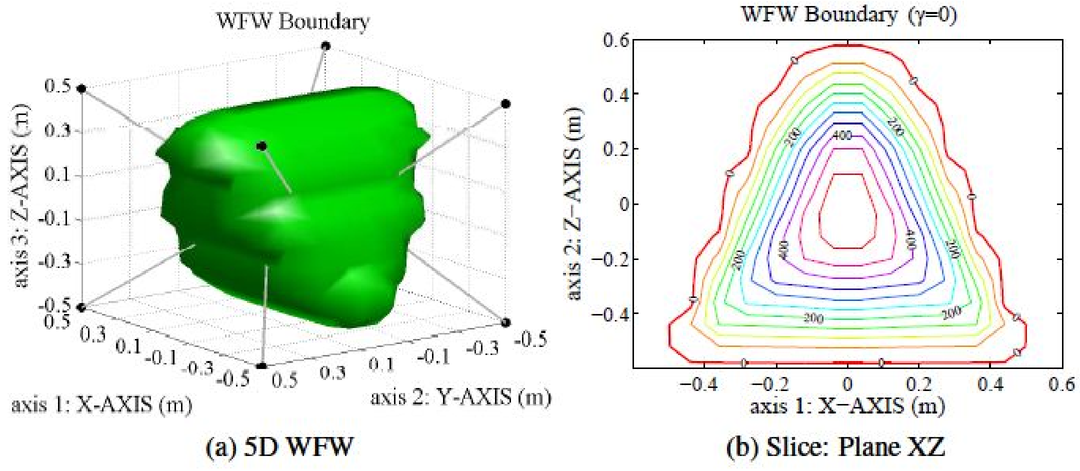


Figure 2.4: Wrench Feasible Workspace of a spatial CDPR as obtained from *ARACHNIS* [5]

WFW can be observed in two cases :

1. : Static Equilibrum Workspace: This is defined as the set of poses for which the end-effector can be balanced by non-negative cable tensions with no external forces on the end-effector except gravity. Here the only forces acting on the robot come from the weight of the cable and the weight of the end-effector

2. : Dynamic Equilibrum Workspace: This is defined as the set of poses for which any instantaneous load attached to the moving platform can be balanced by the non-negative cable tensions.

2.6.2 The interference-free constant orientation workspace (IF-COW)

This is characterized by the set of poses where there is no interference between the cable to another cables or between the cables and the end-effector with a fixed-orientation.

2.6.3 Force Closure Workspace

This is a pose of the cable-driven parallel robot where any arbitrary external wrench applied to the end-effector can be counter balanced by non-negative cable tensions.

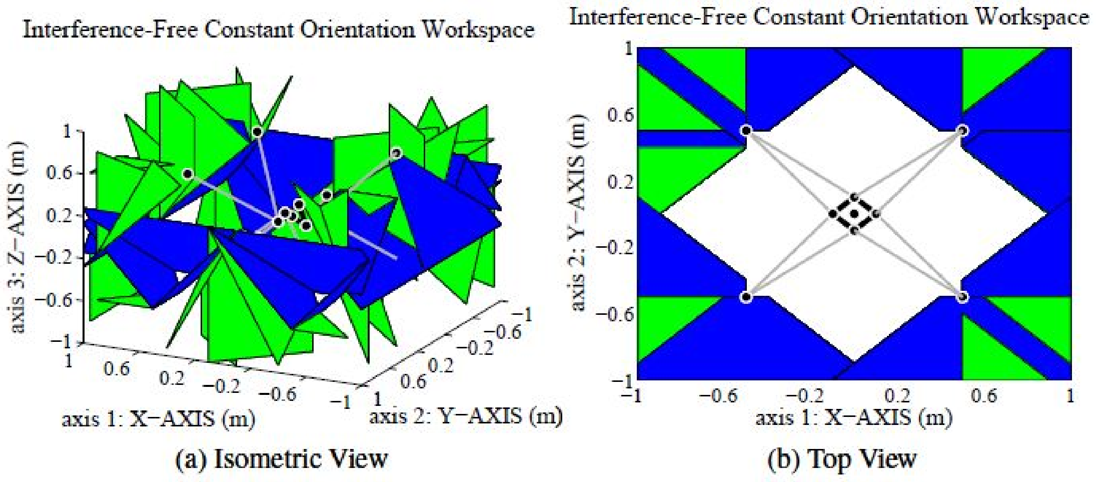


Figure 2.5: interference Free Constant Orientation Workspace of a planar CDPR as obtained from *ARACHNIS* [5]

Several methodologies have been proposed to obtain the different workspaces of a cable-driven parallel robot:

[23] proposes the use of interval arithmetic to analyze a box of poses to determine conditions in which the box of pose could lie either fully inside the WFW or fully outside the WFW.

WFW can also be determined by a discretization to obtain a grid of end-effector poses and testing each pose for wrench-feasibility using a search or 'brute-force' [41] or analytically for point, spatial and planar cable driven parallel robots as has been done in [40].

[42] proposes an algorithm to calculate an interference between a cable of a CDPR and another cable of the same robot and also another algorithm to calculate an interference between the cable

But more recently in [5], a software called *ARACHNIS* has been developed which is able to perform the fast computation of the wrench-feasible workspace and Interference-Free Constant Orientation workspace for a particular CDPR when we provide the parameters describing the robot. This software will be very useful in design.

Chapter 3

The Gazebo Simulator

What is Gazebo?

Gazebo is a 3D dynamics simulator used in developing and testing of robotic applications and scenarios [6]. This enables researchers and developers test a robot or several robots quickly and to evaluate robot performance in more realistic conditions without damaging the actual robot. This is a free and open-source software giving it a huge cost advantage for use in research and development.
Gazebo development began in the fall of 2002 at the University of Southern California. Originally, Gazebo was developed to be used to test robot algorithms but today, it has evolved adding further functionalities such as:

- Dynamics Simulation
- Visualization of 3D Graphics
- Integration of Sensors and Noise to the Robot
- Ability to add plug-ins to test your model
- Some readily available robot models
- Cloud Simulation and Command Line tools.

The current version of gazebo at the time of this publication is *version* 6.0.0. It is free, open-source and very popular within the robotics community. Gazebo is currently funded by DARPA and maintained by OSRF Foundation.

3.1 Physics Engine

The physics Engine is used by gazebo to simulate the dynamics and kinematics of rigid bodies, tests for collisions. Gazebo is able to use different physics engines to perform the simulation.ODE [43] is the default physics engine used, but there exist other physics engines that can be integrated to gazebo based on the users choice such as Bullet, Simbody, DART.

3.2 OpenGL

OpenGL/GLUT work together in gazebo to provide the user interface and visualization for gazebo [44]. OpenGL is a platform independent library for rendering 2D and 3D scenes the rendering of the simulation at the front-end. This used to generate Images of the Simulation for the Graphical User Interface (GUI) or for camera sensors within the simulation. Scenes that are created in OpenGL are displayed on the screen by GLUT.

3.3 Architecture

The gazebo program runs in a client-server architecture which can be depicted in the figure **??**. The server then receives data and control commands from the client through a shared memory interface.

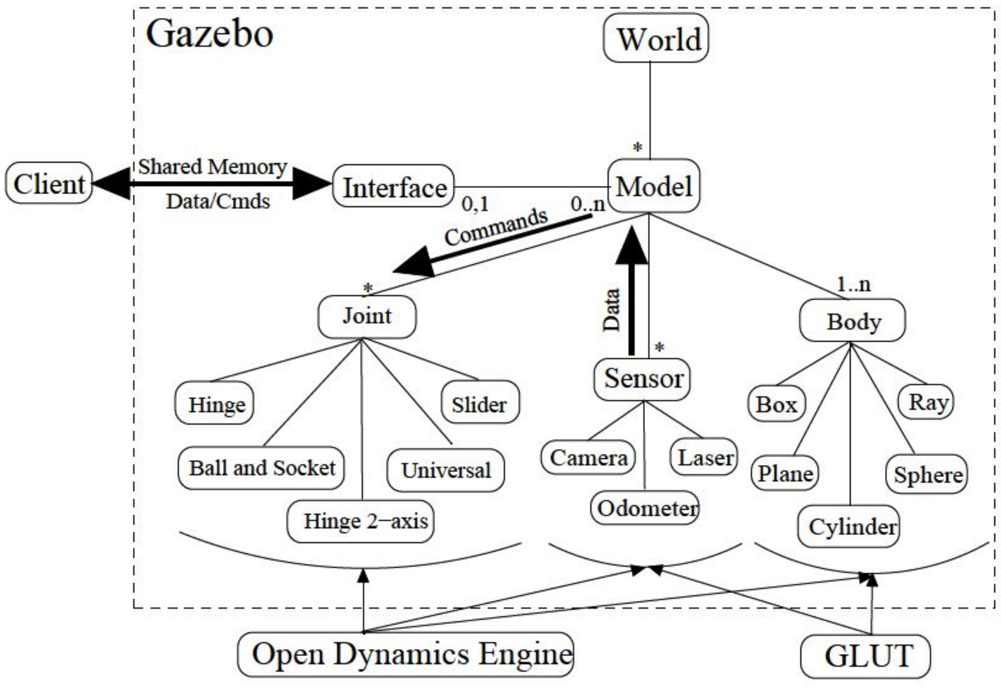


Figure 3.1: Architecture of Gazebo [6]

3.3.1 Models

The world contains all the sensors, joints and links that can be put together to construct a real device or robot. These include basic physical shapes such as a box, cylinder, sphere or other physical aspects such as ray of light, hill etc. These models are defined using a Simulation Definition format (SDF)

A ”link” is a rigid body that moves as a whole. A link contains three main sets of properties:

- Inertial Parameters : These defines the mass and inertia used to calculate the dynamics of the object.
- Collision: Defines the physical space occupied by the object and checks when the object collides with another object or the environment
- Visualization: Used to show the object on the screen or to make it possible to capture the object in a vision sensor

A "joint" holds together two links, constraining their motion. A joint may have limits. We can choose from different types of joints available e.g. universal joint, spherical joint etc to include in our model of a cable driven parallel robot.

The sensors could the different types of sensors available such as a camera, laser range finder etc.

Objects in gazebo can be static or dynamic. Static Objects are those that cannot move but can only collide and are defined by setting the tag in the sdf file as $< static > true < /static >$

Dynamic Objects are those that can move, have an inertia and can collide, they are set by removing the $< static >$ tag.

3.4 SDF

Simulation Description Format, SDF, is an XML file that describes objects and environments for robot simulators, visualization, and controllers [45]. It is an extension of the URDF format that is used in ROS with an improvement of the ability to define lighting, outdoor environments, obstacles etc. A big advantage of Gazebo over ROS is that Gazebo uses SDF which supports closed kinematic chains meaning that we can use it for parallel robots, ROS uses URDF which does not support closed kinematic chains.[46]

3.5 Using Gazebo with ROS

ROS (Robot Operating System) [47] is an open-source middle-ware that contains tools for building and testing robotics applications. It has a lot of plug-ins, libraries and source code that we can easily use from the repository to integrate with the gazebo simulator.

Integration of our Gazebo simulator with ROS gives us access to a wide range of user contributed algorithms making development faster and more focused.

Older versions of Gazebo were included inside of ROS as one of its packages but today Gazebo operates as a standalone software and has no direct dependencies on ROS. The interest in ROS comes from the fact that we can use already defined robot models and robot algorithms that exist in ROS through meta-package called *gazebo_ros_pkgs*.

3.5.1 ROS nodes

These are a collection of largely independent but interacting software processes which are able to communicate with each other using predefined message types. These nodes can be added or removed as long as they follow specific rules to communicate their information. They could be written in C/C++, Python, LISP, MATLAB or Java. The two central process in the ROS frame work is the **roscore** which manages all processes and communication between processes. The parameter server is used to holds parameters centrally to ensure that when two different software processes share data, get the same information while providing a means to store and retrieve software parameters without having to change code.

ROS Topics and Services

Nodes in ROS can run and produce information, this information or data is advertised over a **rostopic** such that another process or software willing to use such information could subscribe to this topic and receive this data at each cycle.

Within ROS, two different software processes, e.g a process that requests for an inverse kinematic solution and another process that performs the inverse Kinematics solver will communicate using using a **rosservice**. ROS topics and services are the two means of communication between ROS nodes.

ROS Messages

ROS defines message types which are structures containing data of different types such as integers, strings, time etc, that can be passed between nodes but we are also free to create out own message type depending on our needs. For example, we can use a PID controller to compute Joint torques but we will compose this torque into a Message vector including a time stamp and this ros node will send this torque as a message to a robot simulation model in gazebo.

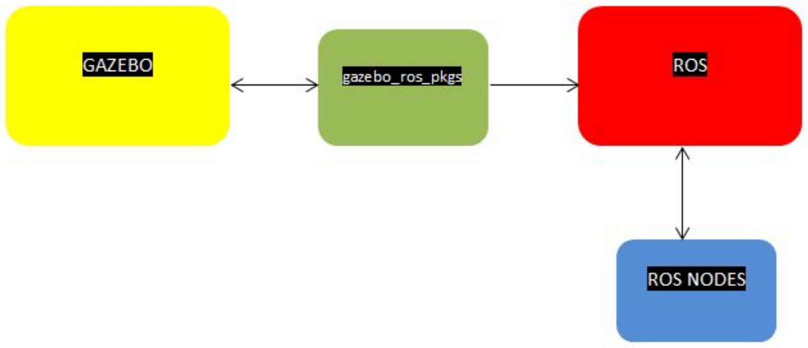


Figure 3.2: Figures Showing the Relationship between Gazebo and ROS [7]

ROS has a very active and collaborative user community and is at the frontier of Robotics software. There are also tutorials, wiki pages, support forums and periodical conferences organized by OSRF. ROS was initially developed at Stanford Artificial Intelligence Laboratory, today ROS is maintained by OSRF Foundation.

3.6 Advantages of using Gazebo

- Ability to define completely our robot in 3D software.
- Simulation of commonly used sensors such as laser ranger finders, GPS, Pan-Tilt-Zoom cameras etc.
- It also has some predefined robot models such as Pioneer2DX, PR2, Care-O-Bot Robot, Kobuki Robot, Nao Humanoid, robot.
- This is also a free software and thus helps us to reduce the development costs.
- We can include several robots in our simulation, add, remove or change our environment and models.
- We can also use the physics engine that simulates the behavior of materials.
- It runs on Linux and can integrate plug ins written in C++/C/Python programming language.
- We can take advantage of the resources such as models, source code and plugins for devices that exist within ROS using the gazebo_ros_pkgs bridge.

3.7 Performance Issues in Gazebo

The Dynamics accuracy is affected by the parameters used in the physics engine and the accuracy of the robot models.

It is therefore important that we note the following points [48] :

1. Choice for the correct step size that will be used to solve the dynamics.
2. Choice for the correct number of iterations in order to obtain a bounded system.
3. Ability to represent as close as possible the real robot in software. The effect of this could be a trade off between having a good visualization and having a fast simulation.
4. Such things as improper joint placement, wrong inertial values or wrong initial positioning of the robot can affect the simulation and should be defined with care.
5. We could have a robot model disappear within the simulation environment due to large accelerations. Such a scenario appears when the force applied to the robot is much greater than it's mass.
6. We could also have a Jittery model if there is a very large ratio between the mass of two connected links. This comes from the stiffness naturally present in such a configuration.

Chapter 4

System Setup

4.1 Operating System

The operating system used in the simulator development for this project is the Ubuntu 14.04.4 LTS (Trusty Tahr 64-bit). This is a long-time support version of Ubuntu was chosen due to its compatibility with most of the software tools and packages used for this project. It is also a free and open source software that is freely available on the Ubuntu website. The installation of Ubuntu for users who may not have it already installed can be found on this reference. [49]

Currently, only the ubuntu distribution is supported on SDF library, thus it is adviceable to perform this simulation development on buntu platform. This covers installation for Gazebo $version$5.0.0 that has been made on ubuntu 14.04 LTS version operating system with the computer connected to the Internet. For installation on older versions of ubuntu, see installation guide at [7].

To have a good simulator environment for 3D simulation in Gazebo, fast communication in ROS and computation in Matlab, we would recommend using a 64-bit computer equipped with a high performance graphics card such as NIVIDA GEFORCE, at least 2 GHz CPU speed and 8G of RAM.

4.2 ROS Intallation

This project uses ROS jade which is the version of ROS that is compatible with our Gazebo libraries. The installation is detailed below and can be found on the ROS web [50]

1. On the linux terminal we type or paste the following lines one after the other:list

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/
    sources.list.d/ros-latest.list'
sudo apt-key adv --keyserver hkp://pool.sks-keyservers.net:80 --recv-key 0xB01FA116
sudo apt-get update; sudo apt-get install ros-jade-desktop-full
sudo rosdep init && rosdep update
echo "source /opt/ros/jade/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

4.3 Gazebo Installation

For this project, we make use of Gazebo 5 which is the version of Gazebo that is comptible with ROS jade [51]. It is important to check the compatibility of a chosen version of gazebo and that of ROS during a development project in order to avoid compile time or run-time errors.

1. On the linux terminal we type or paste the following lines one after the other:

```
wget -O /tmp/installation.sh http://osrf-distributions.s3.amazonaws.com/gazebo/
    gazebo5_install.sh; sudo sh /tmp/gazebo5_install.sh
export GAZEBO_MODEL_PATH=~/.gazebo/models:$GAZEBO_MODEL_PATH
export GAZEBO_RESOURCE_PATH=/usr/share/gazebo-2.2:$GAZEBO_MODEL_PATH
source ~/.bashrc
sudo apt \_ get install ros-indigo-gazebo-ros-pkgs
```

4.4 Developing Models

A model is dynamically loaded into gazebo during simulation by programming or by selecting it from the GUI. Gazebo has an online database where models contributed by a community of designers are stored and can be accessed. This can be cloned from a linux terminal by:

```
hg clone https://bitbucket.org/osrf/gazebo_models
```

Every Model is created to conform to a specific file structure as shown below:

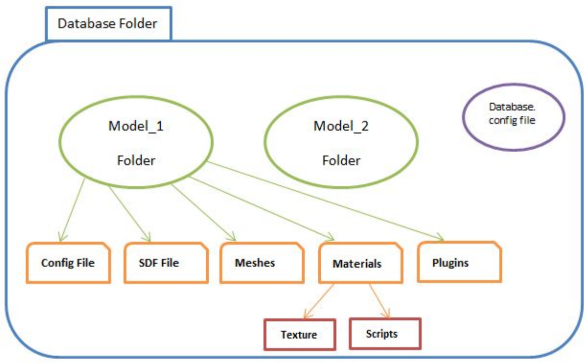


Figure 4.1: Database Structure for a Gazebo Model

1. The database.config contains the metadata about the database and is only required if we want to store our model in the online repository.

2. model.config is an xml file that contains the metadata about the model itself such as the name of the model and the author.

3. model.sdf contains the Simulator description format of the model.

4. meshes is a directory where we put .dae or stl files of 3D models.
5. plugins is a directory where we keep source files and header files for plugins.
6. The materials directory contains the texture folder which holds images of the model (jpg, png)and scripts folder where we can find material scripts.

Creating the World

We will create an empty world with with ground plane, light source.

1. Inside gazebo_ros_pkgs folder create a tutorial.world file and paste the following inside:

```
?xml version="1.0" ?>
<sdf version="1.4">
<world name="createWorld">
<include>
<uri>model://sun</uri>
</include>
<include>
<uri>model://ground_plane</uri>
</include>

```

Add Obstacles

We will add some obstacles to the empty world file.

1. Inside the tutorial.world file and paste the following inside:

```
 <model name="barrierA">
 <include>
 <uri>model://jersey_barrier</uri>
 <pose> 2 0 0 0 0 0</pose>
 <static>true</static>
 </include>
 </model>
 <model name="barrierB">
 <include>
 <uri>model://jersey_barrier</uri>
 <pose> 0 2 0 0 0 1.2</pose>
 <static>true</static>
 </include>
 </model>


```

Import a Created Model

For this demonstration, we will add a previously existing robot model in gazebo which closely represents a single cable.

1. Inside the tutorial.world file and paste the following inside:

```
<!-- Import a Fire hose -->
<include>
<uri>model://fire_hose_long</uri>
<pose>0 0 5 0 45 0</pose>
</include>

```

Adding a Sensor to a Model

We will add a hokuyo laser sensor to the cable that we have added previously, we define the pose of this sensor and attach it to the cable using a joint.

1. Inside the tutorial.world file and paste the following inside:

```
<include>
<uri>model://hokuyo</uri>
<pose>0 0 6 0 0 0</pose>
</include>
<joint name="hokuyo_joint" type="revolute">
<child>hokuyo::link</child>
<parent>fire_hose_long :: nozzle</parent>
 </joint>
  </world>
  </sdf>

```

Make sure the tutorial.world file amd then launch it on gazebo by typing on the terminal:

1. Inside the tutorial.world file and paste the following inside:

```
gazebo tutorial.world

```

We should see something similar to:

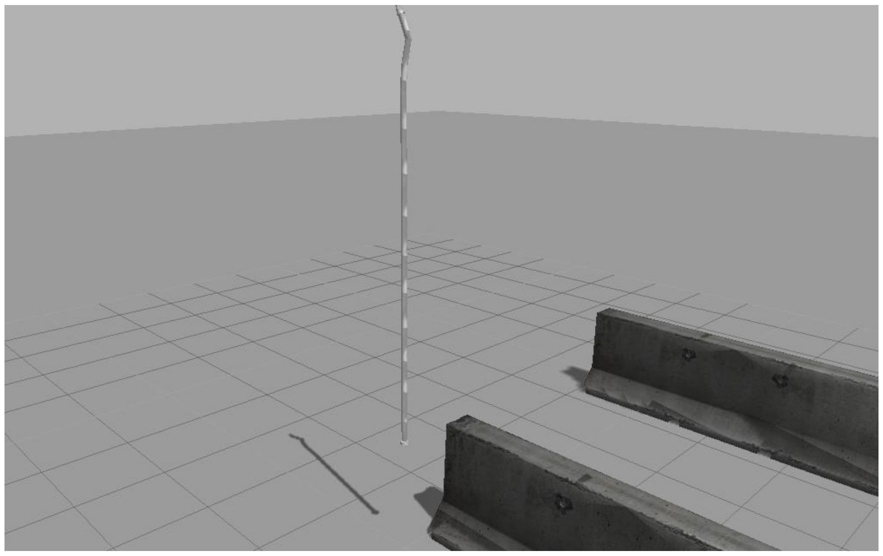


Figure 4.2: An example of Modelling of a cable in Gazebo

4.4.1 Adding a Plugin to a model

A plugin is a C++ code or algorithmn designed to perform a specific task. We can use such code to solve robot kinematics, implement a robot controller and send information from one robot to another. Plugins enable us to choose exactly what to include in our model in order to obtain a specific desired behaviour. They are very flexible since they enable us to alter the program in simulation, rather than being compiled statically. Using plugins in Gazebo also enables a quick sharing of code. There are 5 types of plugins in gazebo:

1. World plugin which is attached to the world
2. Model plugin which is attached to a particular model, e.g a robot
3. Sensor plugin which is attached to a sensor to define the behaviour of such a sensor
4. System plugin which is loaded during startup in the command line.
5. Visual plugins for the rendering

1. Here we show an example of where a plugin is added to a robot model. The plugin is identified by a plugin name and the file name which ends in **.so**. We can optionally add parameters if they are required by the plugin. we can refer to [8] for an example of how a plugin is created in C++

```
<model name="your_robot_model">
<plugin name="differential_drive_controller" filename="libdiffdrive_plugin.so">
... plugin parameters ...
</plugin>
</model>

```

4.4.2 Integrating with with ROS

Gaebo used to be a ROS package but now it runs as a stand-alone package with no dependencies on ROS. The main contact point between gazebo and ROS is through the plugins. We are able to create gazebo plugins through C++ classes that have been predefined in ROS. Then we an compile these plugins, add them to our world file and launch them as a ROS package from the catkin folder. The process flow is shown below:

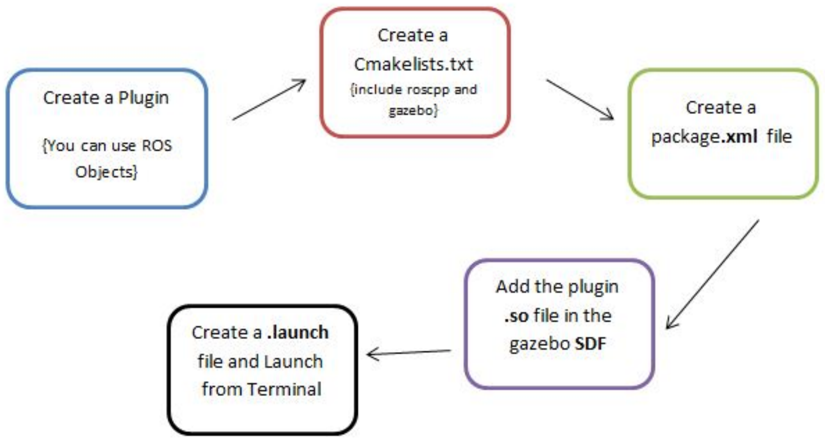

Create a Plugin
{You can use ROS Objects}
Create a Cmakelists.txt
{include roscpp and gazebo}
Create a package.xml file
Add the plugin .so file in the gazebo SDF
Create a .launch file and Launch from Terminal

Figure 4.3: Process flow of using Gazebo and ROS together [8]

4.5 The Simulator as a Catkin Package

The simulator is developed under the standards adopted by ROS for the creation and deployment of ROS packages. Therefore it has been developed as a catkin package [52]. The catkin package is a collection of source files used in this project and all their dependencies all grouped under a single workspace called **cdpr**.

This means that a user that who has a catkin workspace installed can copy the project folder into their catkin workspace, compile and launch the project very quickly.

Below we will detail the installation of catkin workspace for users who do not have this installed already and then we will proceed to describe the early stages of the project creation.

4.5.1 Catkin Installation

1. : We set-up the catkin workspace which is a folder in the top level of the home directory within which we will create all packages for the project.

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/src
catkin_init_workspace
cd ~/catkin_ws
catkin_make
```

```
echo "source /home/user_name/catkin_ws/devel/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

4.5.2 Creating the Catkin Package

The catkin package was created using the command line syntax for creating a package as shown below including all the dependencies on the terminal:

```
catkin-create-package cdpr  gazebo_ros roscpp roslib sensor_msgs
geometry_msgs  visp
```

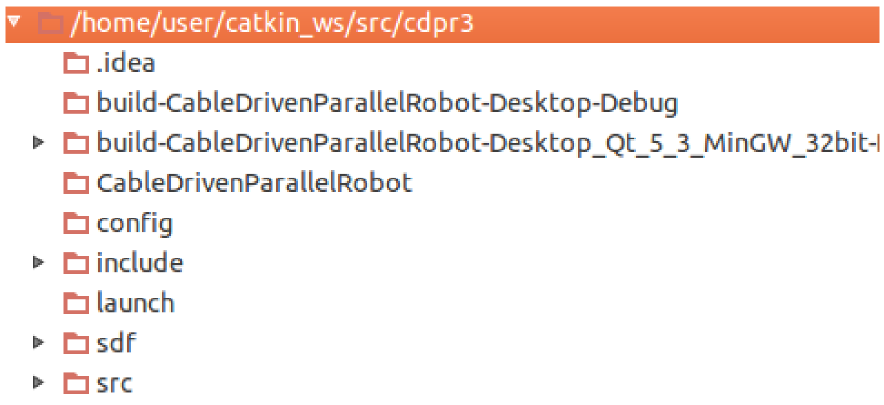
4.6 VISP Installation

As we can see from the catkin package creation step, we make use of some classes and messages available in ROS. But we also make use of some external libraries such as Visp Library [53] [54] which contains some useful objects and functions that we use to perform certain matrix and vector operations. Visp was installed by typing on the terminal:

```
sudo apt-get install ros-jade-visp
```

4.7 Folder Structure of the Workspace

The below image shows the folder structure of the workspace and we will give a little explanation on the contents of each important folder.



```
/home/user/catkin_ws/src/cdpr3
    .idea
    build-CableDrivenParallelRobot-Desktop-Debug
    build-CableDrivenParallelRobot-Desktop_Qt_5_3_MinGW_32bit-
    CableDrivenParallelRobot
    config
    include
    launch
    sdf
    src
```

Figure 4.4: File Structure of the workspace

1. The **Include** folder contains all the C++ header files and other dependencies for the robot controller and other plug-ins used for the project. It also contains the python source files whose use we will be explaining later in this report.

2. The **launch** folder contains the launch files used for launching the robot in Gazebo. We shall also be explaining the process of launching the robot into simulation and other options that are possible

3. The **sdf** folder contains the simulation description file which is an XML-based description of the robot that is then interpreted by the gazebo to generate the robot in simulation. The folder also includes the *.yaml* file that contains the parameters of the robot and changing this file enables us to generate a different robot.

4. The **src** folder contains C++ source files of the robot task-space dynamic controller and the trajectory generator used for this controller. It also contains the python source files for generating the User Interface as a plug-in and also the user interface *.ui* files.

4.8 Installation of Pycharm

The python code development is done with the help of Pycharm which is an Integrated development environment for python. It is suffice to say that any IDE that supports python development can be used for this part of the development but it is important to precise whatever the choice of IDE, this project uses the python 2.7.6 interpreter.

The Pycharm installation can be done by opening a terminal and using the following commands:

```
sudo add-apt-repository ppa:mystic-mirage/pycharm
sudo apt-get update
sudo apt-get install pycharm-community
```

Chapter 5

Development of the Robot SDF

5.1 Representation of the components of a cable robot in gazebo

The Simulation Description Format (SDF) is an XML format employed by Gazebo to describe models and their properties. SDF describes several properties of a cable driven parallel robot by coding information about the component parts such as the links, the moving platform, the fixed base and the joints. We precise here that the SDF file that describes the robot is developed with the help of a collection of python functions previous used in another Gazebo project for Autonomous Underwater Robotic simulator [34]. The functions available in this projected were adapted and used for a fast robot SDF generation. It also gives a good indentation which helps us to save a lot of time and gives a lot of accuracy when writing thousands of lines of indented XML code that describes the robot.

5.2 The Fixed base

This was modelled as a single parallel piped cuboid and it is on the vertices of this cuboid or any other known points of this cuboid that we attach the cables. For stability, this fixed based is assigned a heavy mass (100000kg in our case) which is more greater than the mass of any of the other components attached to it. Within its collision property in the SDF, it is set as to be able to collide which makes it to remain relatively fixed when other objects collide with it due to its heavy mass. This cuboid is built of with 12 segments whose lengths were calculated from the known co-ordinates and desired placement in the gazebo enviroment

5.3 The moving platform

The moving platform has been modelled with two possibilities

1. One possibility is to represent the moving platform as a point mass whereby in our Gazebo modelling we use a sphere object to denote this point mass. All the cables are attached to the same point on the end effector. Thus this can be used to model the following classes of Cable-driven parallel robots [55]

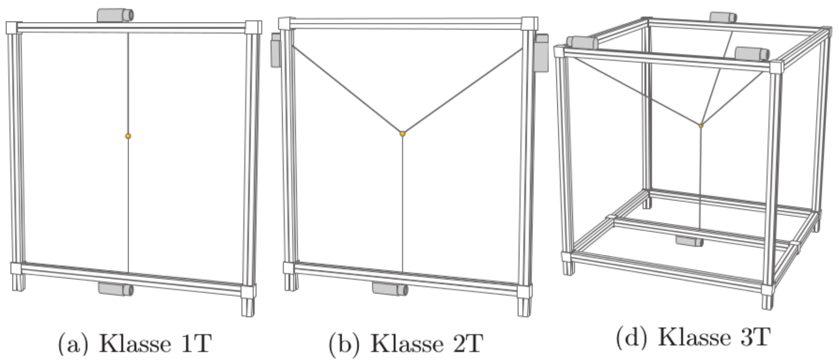


Figure 5.1: classes of cable driven parallel robots with point mass moving platform [9]

(a) 1T linear motion of a point

(b) 2T planar motion of a point

(c) 3T Spatial motion of a point

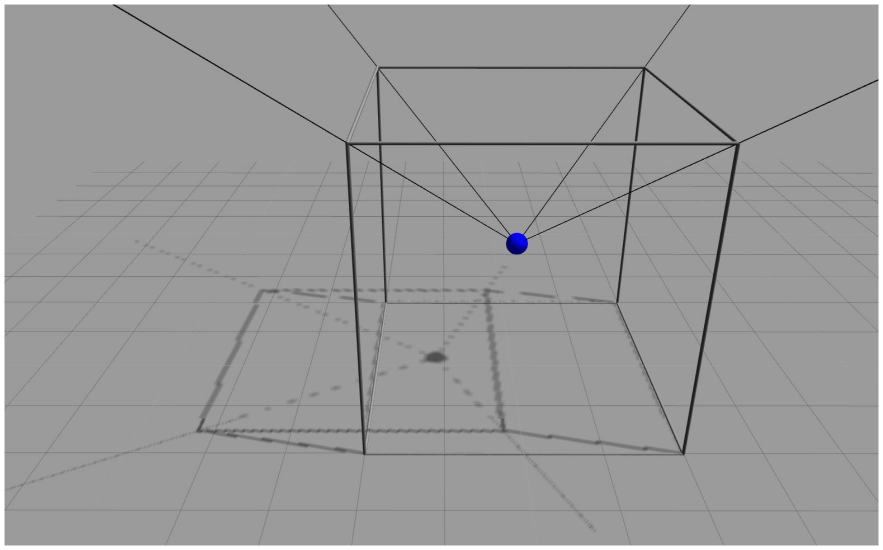


Figure 5.2: cable driven parallel robot with end-effector as point mass in gazebo

2. Another possibility is to represent the moving platform as a spatial body. In this case, we have used a parallel-pied (box in gazebo terms). The vertices of this box object are known or can be determined easily by geometric relations which makes it ideal for use in the representation of the moving platform. We also assume that the position of the center of mass of the moving platform coincides with the geometric center of the this box The cables can then be attached at the vertices of this box as Thus this can be used to model the following classes of Cable-driven parallel robots [55]

   (a) 1T linear motion of a point

   (b) 2T planar motion of a point

   (c) 3T Spatial motion of a point

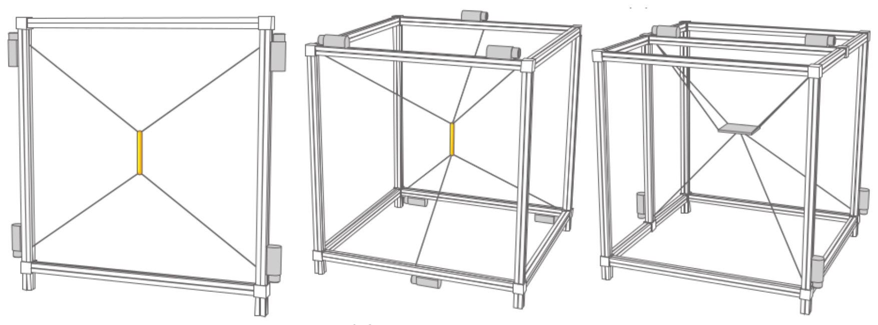

(c) Klasse 1R2T (e) Klasse 2R3T (f) Klasse 3R3T

Figure 5.3: classes of cable-driven parallel robot with box-shaped end effector

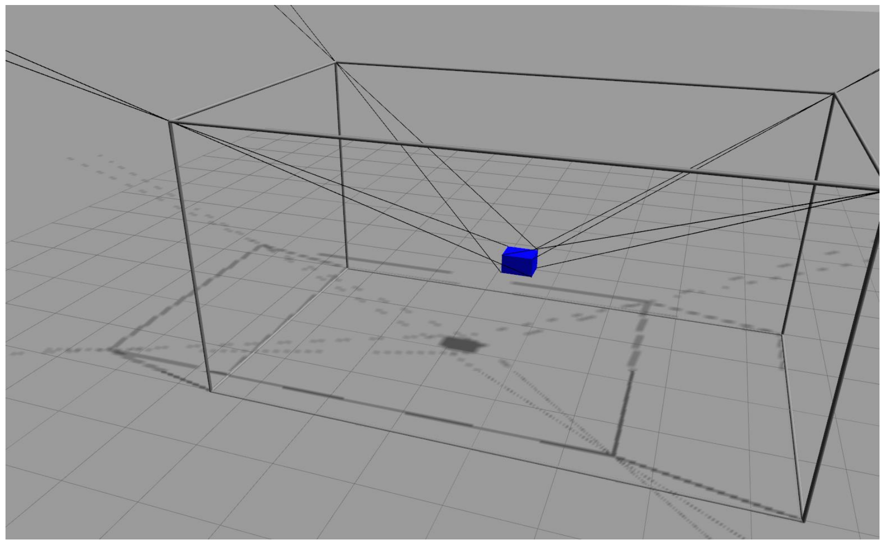


Figure 5.4: 8-cable Caroca cable robot in gazebo with box-shaped end effector

5.4 The Joints

A cable driven parallel robot link can be seen as a UPS architecture with the prismatic joint actuated [56]. This formalism is used in the modelling of the joints. The joints are used to connect the links that represent the cable to the link representing the moving platform and used to connect the link representing the fixed base to the other end of the cable for each cable.

By this method, we have actuated joints in the construction and passive joints. The prismatic joint of each cable is actuated and is directed linearly along the axis of the cable. Forces were applied on this joint to simulate the effect produced by a cable on a moving-platform produces the desired motion. The passive joints were placed both ends of the cable to which we do not apply any forces to these joints in simulation. They were created by using the formalism of axis-intersecting revolute joints which constrain a number of degrees of freedoms between two bodies. The spherical joints at the moving platform consists of three intersecting revolute joints and at the fixed base, we placed two intersecting revolute joints forming a universal joint.

5.5 The cables

The cables are modelled using the massless inextensible model [57]. This helped us to represent the cables as rigid cylinders under tension thereby simplifying the cable model for control using the gazebo objects are similar to this model which are straight cylinders that have a small mass (0.001kg). The cylinders which make up the cable have a thin radius to model a cable and the collision property of the cable is disabled allowing the cable to become attached to the fixed frame and prevent the cable from colliding with the ground plane.

The mass of this cylinder in gazebo cannot be set to zero in gazebo as this causes gazebo to crash since the physics engine cannot solve objects that have zero mass. We also find that the stability of the model depends on the ratio between the mass of the cable and the mass of the end-effector. Indeed the author in [57] shows this relationship experimentally therefore the total mass at the end-effector and the mass of the cable was carefully chosen to maintain the cable model as an almost massless and inextensible rigid object.

5.6 Positioning of the cables in gazebo

In this section we describe a calculation performed to generate the pose of the cable that is attached at the fixed frame and the end effector. We note that to set absolute position and orientation of an object in gazebo, it requires 3- co-ordinates for the position and 3 -elements to represent the orientation in the absolute frame.

Gazebo expects that the pose is at the midpoint of the cylinder object and therefore it was required that we calculate the initial position in absolute frame of the cable to correspond to the cylinder midpoint. We also calculated the orientation of the cable at the initial configuration of the robot its rest position. We denote the parameters used in this calculation

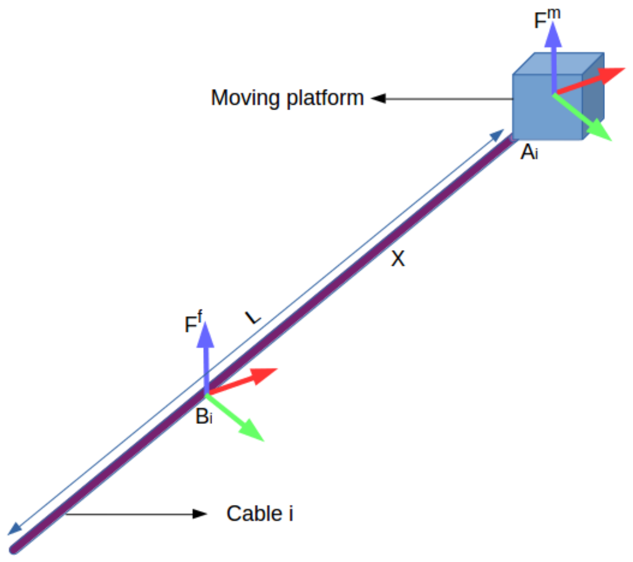


Figure 5.5: Figure showing the attachment and positioning of a cable in gazebo

as:

1. $A_i(x_a, y_a, z_a)$ = Co-ordinates of the Attachment points of the cable on the fixed frame given in the robot's absolute frame.

2. $B_i(x_b, y_b, z_b)$ = Co-ordinates of Attachment points on the moving platform given in the frame attached to the moving platform

3. $P(p_x, p_y, p_z)$ = Initial position of center of mass of the moving platform in the absolute frame = Geometric center of the box.

4. l_i = length of cable i with a unit vector u_i along this cable

To compute the cable orientation for each cable i:

1. So we compute the homogeneous transformation matrix between the fixed frame and the moving platform's frame. If we assume a pure translation and no rotation between both frames
$$^{f}\mathbf{T}_m = \begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_{3X3} & \mathbf{P}_{2x1} \\ \mathbf{0}_{1x3} & 1 \end{bmatrix} \tag{5.1}$$

2. Then we multiply the homogeneous co-ordinates of the moving platform attachment points by this transformation matrix in order to define these points in the absolute frame which is equivalent to:
$$^{f}A_i = QA_i + P \tag{5.2}$$

3. Then we compute the unit vector between along the fixed frame attachment point and the moving platform attachment point in the absolute frame (vector along the axis of cable):
$$\vec{u} = \frac{^{f}A_i - ^{f}B_i}{\left\| ^{f}A_i - ^{f}B_i \right\|_2} \tag{5.3}$$

4. Then we computed the rotation matrix between the z-axis of the absolute frame and the unit vector lying along the axis of the cable where both vectors are unit vectors:

   First we compute the angle between these two vectors
$$angle = atan2(norm(cross(z, u)), dot(z, u)) \tag{5.4}$$

   where the z-axis is defined as $\vec{z} = [0, 0, 1]^T$

   Then we find the orthogonal vector between the z-axis and the axis along the cable length
$$\vec{e} = \vec{u} \times \vec{z} \tag{5.5}$$

Then we can find the rotation matrix by using Rodrigues formular

$$R(\vec{e}, angle) = I_3 + \hat{e}\sin(angle) + \hat{e}^2(1 - \cos(angle) = \begin{bmatrix} s_x & n_x & a_x \\ s_y & n_y & a_y \\ s_z & n_z & a_z \end{bmatrix} \tag{5.6}$$

And then the roll, pitch and yaw angles for describing the orientation of the cable in gazebo can be recovered from the rotation matrix using the formula for the zyx convention

$$\phi = atan2(s_y, s_x) \tag{5.7}$$

$$\theta = atan2(-s_z, s_x \cos\phi + s_y \sin\phi) \tag{5.8}$$

$$\psi = atan2(a_x \sin\phi - a_y \cos\phi, -n_x \sin\phi + n_y \cos\phi) \tag{5.9}$$

To compute the cable position:

1. We note that we set the maximum length of each cable to be equal to the length of one diagonal of the fixed frame. Then these relations hold for each cable to be positioned accurately:

$$\vec{XA} = \alpha \vec{BA} \tag{5.10}$$

$$\|XA\|_2 = \frac{L}{2} \tag{5.11}$$

$$X =^f A_i - \frac{L}{2}\left(\frac{^f A_i -^f B_i}{\left\|^f A_i -^f B_i\right\|_2}\right) \tag{5.12}$$

The above calculation is performed in a python script that is used to generate the SDF of the cable-driven parallel robot.

5.7 The CDPR Plugin

As part of the development of the SDF, we have specified the name of the model plugin which is used as the operational space dynamic controller of the robot. The gazebo methodology for introducing a model plug-in and we can see as shown below:

```
<plugin filename="libcdpr3.so" name="freefloating_gazebo_control"/>
```

Above we have simply included in the sdf file describing the robot, the name of the compiled binary of the shared object that is used as the model plug-in.

5.8 The YAML file

YAML is a human friendly data serialization standard for all programming languages. It is able to hold complex nested data structures corresponding to variables that can describe the same object.

We use a YAML file to hold the values and keys of all parameters that are used to generate the robot SDF file such as masses, co-ordinates, damping co-efficient, tension limits.

They are accessed by the python function used in the robot generation and can be modified in order to generate a new robot. The contents of the YAML file can be shown in the example below:

```
model:
cable: {mass: 0.001, radius: 0.005}
filepath: /home/themarkofaspur/catkin_ws/src/cdpr3/sdf/cube.yaml
frame:
  color: Black
  lower: [−3.5, −2, 0]
  mass: 5000.0
  radius: 0.02
  type: box
```

5.9 Launching the Developed Robot

Roslaunch provides a tool for easily launching multiple ROS nodes locally and remotely via SSH, as well as setting parameters on the Parameter Server [58]. It enables us to launch multiple rosnodes and to remap the topics of these nodes to make them communicate within the ROS enviroment.

```
<launch>
    <arg name="paused" default="true"/>
    <arg name="model" default="cube"/>
    <!−− Launch Gazebo with empty world−−>
    <include file="$(find gazebo_ros)/launch/empty_world.launch">
        <arg name="paused" value="$(arg paused)"/>
</launch>
```

We have created a *.launch* file in the launch folder that enables us to launch the generated robot in gazebo. The gazebo is launched in an empty world in paused mode and we also specify as a parameter that is stored in the parameter server, the *.yaml* file as it contains parameters and co-ordinates that are accessed in the model plugin to be used in the control.

In order to launch the robot, we type on the terminal

```
roslaunch cdpr3 cube.launch
```

Chapter 6

The controller as a Model Plug-in

Here we will present the implementation of the control scheme that we adopted for this project. Generally a user of this project could decide to develop and use their own defined controller as a model plug-in. But we develop a controller in order to prove controllability of this robot.

We have chosen to implement the dynamic task space controller because gazebo as simulator is a dynamic simulator and we use a task space controller to simplify the approach since in this approach, there is no need to measure the cable length at each time instant and there exists functions in gazebo that enable us to obtain the position, velocity and acceleration of the moving platform.

6.1 The Dynamic Controller in Task Space

We recall that the Dynamic model of cable-driven parallel robots was shown in (2.17) considers only the platform dynamics assuming that massless and inextensible cables are used. If we consider that the position of the center of mass of the moving platform is the same as the geometric center of the moving platform and neglecting Coriolis force terms, the dynamic model becomes:

$$\mathbf{M}\ddot{\mathbf{x}} + \mathbf{f}_g = \mathbf{W}\mathbf{t} \tag{6.1}$$

where from (2.10), the sum of the external forces acting on the moving platform is:

$$\mathbf{f}_p = -(\mathbf{M}\ddot{\mathbf{x}} + \mathbf{f}_g) \tag{6.2}$$

Given a desired trajectory $[\mathbf{x_d}, \dot{\mathbf{x_d}}, \ddot{\mathbf{x_d}}]$ to which we want the system dynamics to track, the dynamic control in task space uses the feedback linearisation of the dynamic model to stabilise the system to desired trajectory [38][59] [3] [56] The system dynamics is given as:

$$\ddot{\mathbf{x}} = \mathbf{v} \tag{6.3}$$

Then the linear system becomes:

$$\mathbf{v} = \ddot{\mathbf{x_d}} + \mathbf{Kp}(\mathbf{x_d} - \mathbf{x}) + \mathbf{Kd}(\dot{\mathbf{x_d}} - \dot{\mathbf{x}}) \tag{6.4}$$

from (6.1) and (6.2), the entire control law is now

$$\mathbf{Wt} = -(\mathbf{M}(\ddot{\mathbf{x_d}} + \mathbf{Kp}(\mathbf{x_d} - \mathbf{x}) + \mathbf{Kd}(\dot{\mathbf{x_d}} - \dot{\mathbf{x}})) + \mathbf{f}_g) \tag{6.5}$$

Describe a wrench on the moving platform which allows us to calculate the desired cable tensions using cable tension distribution methods to be detailed as follows.

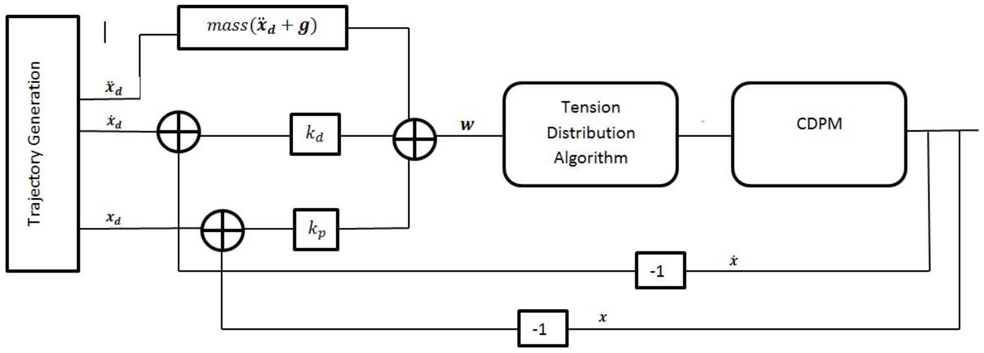


Figure 6.1: Control System Architecture

$\mathbf{K_p}$ and $\mathbf{K_d}$ are proportional and derivative gains applied to the moving platform positional and velocity errors respectively to achieve asymptotic stability. These gains are tuned to achieve the desired tracking performance and will depend on the cable robot. We tuned it by trial and error approach but desired controller performance such as stability and tracking or other sophisticated approaches can be used also to tune these gains using advanced control techniques.

6.2 Trajectory Generation

Several approaches exists for trajectory generation in order to move a robot from one initial configuration to a final configuration.The author in [59] uses a designed trajectory to solve the problem of discontinuity of the cable tension by selecting a feasible solutions for the desired trajectory and then selecting the coefficients that allow to determine this solution while respecting boundary conditions. For this work, we develop our controller using the 5th order polynomial function because we are working with the dynamic model which necessitates second-order dynamics.

For the desired position:

$$x_{(t)} = at^5 + bt^4 + ct^3 + dt^2 + et + f \tag{6.6}$$

For the desired velocity:

$$\dot{x}_{(t)} = 5at^4 + 4bt^3 + 3ct^2 + 2dt + e \tag{6.7}$$

For the desired acceleration:

$$\ddot{x}_{(t)} = 20at^3 + 12bt^2 + 6ct + 2d \tag{6.8}$$

Given the boundary conditions:

$$x(t_i) = x_i \tag{6.9}$$

$$\dot{x(t_i)} = \dot{x_i} \tag{6.10}$$

$$\ddot{x(t_i)} = \ddot{x_i} \tag{6.11}$$

$$x(t_f) = x_f \tag{6.12}$$

$$\dot{x(t_f)} = \dot{x_f} \tag{6.13}$$

$$\ddot{x(t_f)} = \ddot{x_f} \tag{6.14}$$

where: t_i is initial time and t_f is final time x_i = initial position $\dot{x_i}$ = initial velocity $\ddot{x_i}$ = initial acceleration x_f = final position $\dot{x_f}$ = final velocity $\ddot{x_f}$ = final acceleration

Where the total trajectory time can be given as:

$$T = t_f - t_i \tag{6.15}$$

The coefficients a,b,c,d,e,f can be determined from the relationships:

$$f = x_i \tag{6.16}$$

$$e = \dot{x_i} \tag{6.17}$$

$$d = \frac{\ddot{x_i}}{2} \tag{6.18}$$

$$c = \frac{20(x_f - x_i) - (8\dot{x_f} + 12\dot{x_i})T - (3\ddot{x_f} - \ddot{x_i})T^2}{2T^3} \tag{6.19}$$

$$b = \frac{30(x_f - x_i) + (14\dot{x_f} + 16\dot{x_i})T + (3\ddot{x_f} - 2\ddot{x_i})T^2}{2T^4} \tag{6.20}$$

$$a = \frac{12(x_f - x_i) - (6\dot{x_f} + \dot{x_i})T - (\ddot{x_f} - \ddot{x_i})T^2}{2T^5} \tag{6.21}$$

These co-efficients give us at any time step in the trajectory, the co-ordinates of the position, velocity and the accelerations.

The above trajectory generation has been implemented in a trajectory generation C++ class. This class is used in a rosnode to generate desired position, velocity and acceleration of all degree freedoms of the cable driven parallel robot which we control in simulation at each time step. Then theses desired values are published to the robot model through a rostopic. This rosnode makes use of the VISP library class (vpPlot) to make the calculations and to make plots at the end of the trajectory in order to analyse the results. An example of a trajectory used for experiment is shown below:

An example of trajectory that was is used for all the experiments in this work was defined using the following trajectory conditions $T = 100$, $x_i = [0, -2, 2, 0, 0, 0]$, $\dot{x_i} = [0, 0, 0, 0, 0, 0]$, $\ddot{x_i} = [0, 0, 0, 0, 0, 0]$,$x_f = [3, 1, 3, 0, 0, 0]$, $\dot{x_f} = [0, 0, 0, 0, 0, 0]$, $\ddot{x_f} = [0, 0, 0, 0, 0, 0]$ This trajectory was applied to the trajectory planner in our simulator and we get the plots below for the desired position, velocity and acceleration. We also developed the same trajectory in Matlab to make comparison as a control for the experiments.

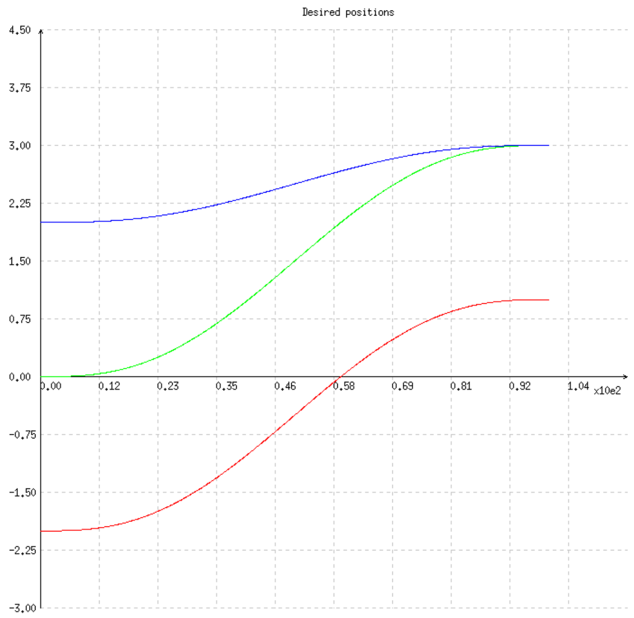


Figure 6.2: Desired Positions from the trajectory class of the simulator example

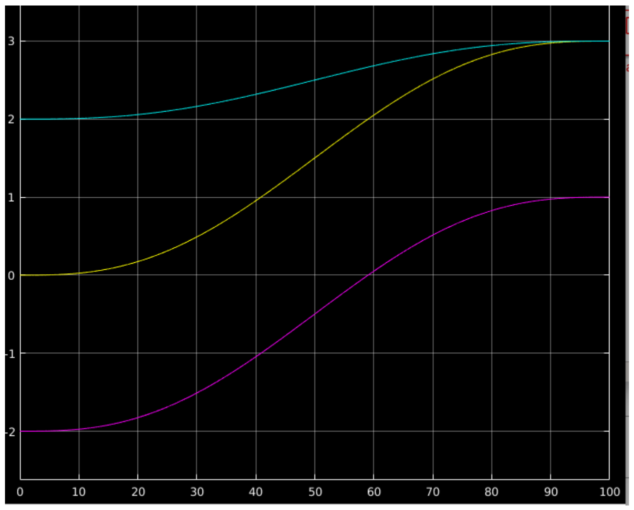


Figure 6.3: Desired Positions from the trajectory developed in MATLAB

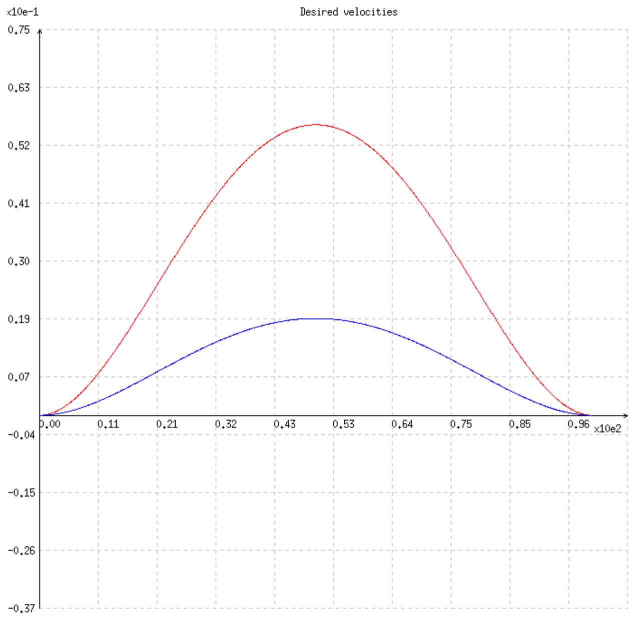


Figure 6.4: Desired Velocities from the trajectory class of the simulator example

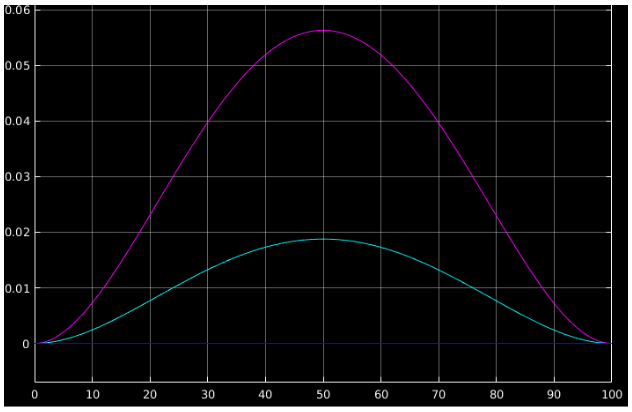


Figure 6.5: Desired Velocities from the trajectory developed in MATLAB

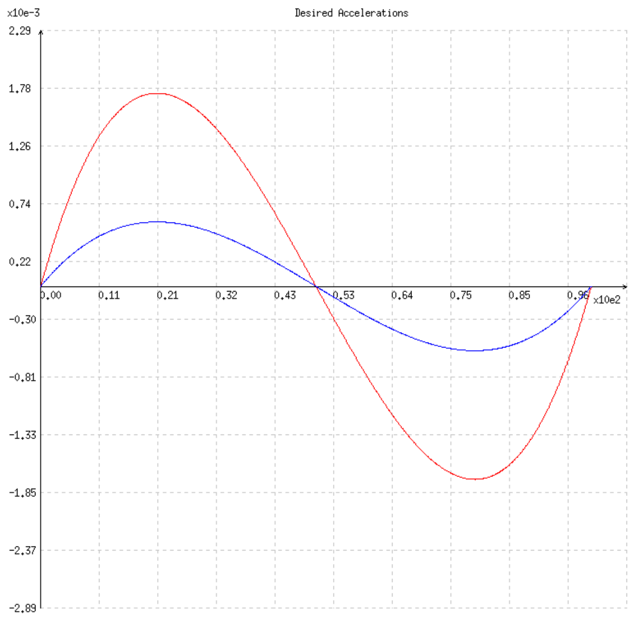


Figure 6.6: Desired Accelerations from the trajectory class of the simulator example

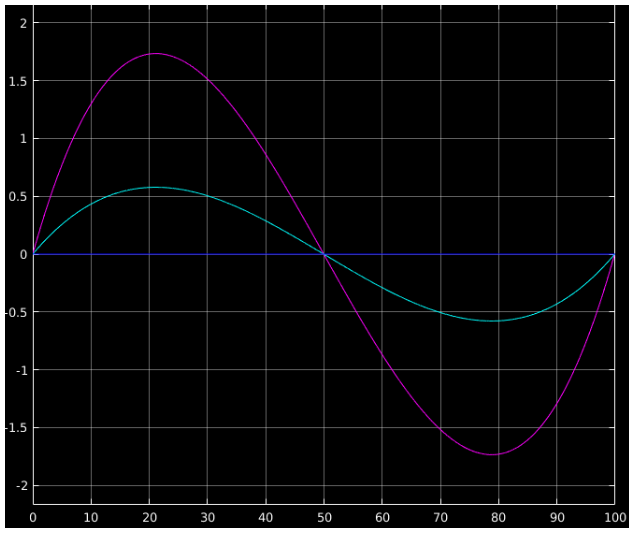


Figure 6.7: Desired Accelerations from the trajectory developed in MATLAB

6.3 Current Pose and Velocity of the moving platform

Gazebo provides functions that can be used to obtain different properties of a body, link or model under simulation. We have defined the moving platform as a link in gazebo hence we are able to get the pose and velocity of the moving platform in the absolute frame at any time step.

1. In order to get the current pose in the absolute frame we use the function as follows

   ```
   gazebo::math::Pose  body->GetWorldPose();
   ```

   And the function returns for us a vector containing the x, y ,z current position of the moving platform which we use to build our position error in simulation. The function also returns for us the current orientation of the moving platform in simulation and then we can use the Visp function:

   ```
    current_rotation = vpRotationMatrix(vpQuaternionVector(x,y,z,w));
   ```

2. Similar functions exist that enable us to obtain the current linear and angular velocities of the moving platform in the absolute frame if we wish to use them to build the velocity error as follows:

   ```
        gazebo::math::Vector3  body->GetWorldLinearVel();
        gazebo::math::Vector3  body->GetWorldAngularVel();
   ```

6.4 Tension Distribution

Tension distribution is a very important subject in the study of cable driven parallel robots. Tension distribution is used to find the solution the system:

$$\mathbf{W}\mathbf{t} + \mathbf{f}_p = \mathbf{0} \tag{6.22}$$

by delivering a unique solution vector containing all admissible tensions while respecting the conditions given in equations (2.12) and (2.13).

As we have noted before in this work, cable driven parallel robots, can be designed with no redundancy. In such a case where there is no redundancy, we have the same number of cables as the number of degrees of freedom ($m = n$) and so the wrench matrix,$\mathbf{W}$, is square. Therefore the solution of the problem in equation (6.5) is unique and can be obtained by simply inverting the matrix $\mathbf{W}$.

$$\mathbf{t} = -\mathbf{W}^{-1}\mathbf{f}_p \tag{6.23}$$

It has been shown in [25] that we cannot guarantee non-negative tensions in the cables with a cable driven parallel robot with no redundancy. Physically, it is undesirable to have negative cable tensions as this means cables which are able to push on the moving platform which is physically impossible and so infeasible in control. A cable-driven parallel robot with redundancy has more cables than the number of degrees of freedom that it controls ($m > n$) and its redundancy can be calculated as $r = m - n$. Then the solution to the control problem in (6.5) is non-unique and the system is under determined.

The author in [60] makes a comparison of the different approaches employed in order to obtain the solution to the tension distribution problem in literature on properties such as:

1. Real-time capability
2. Continuity of solution
3. Maximum redundancy
4. Tension limits
5. Computation speed

The author posits that no method for tension distribution is known to have all combined properties of being real-time capable, covering the full workspace, delivers continuous solution for control, when applied on a robot with arbitrary degree-of-redundancy and proposes an improved variant of the closed-form method to solve this problem in real-time. Below, we will make a brief study of some popular approaches for tension distribution by applying them in the control loop with our cable driven parallel robot simulator with a view to show which solution is most suitable for our cable driven parallel robot. We will also show in the preceding section, how we implement each of these different tension distribution algorithms and the results obtained from each algorithm on our robot.

6.4.1 1-norm Optimal Solution

This formulates the tension distribution problem as a Linear programming (LP) problem and writes the 1-norm solution as [26]:

$$\begin{aligned} \underset{t}{\text{minimize}} \quad & \mathbf{1}^T\mathbf{t} \\ \text{subject to} \quad & \mathbf{W}\mathbf{t} = -\mathbf{f}_p \quad \mathbf{t}_{min} \leq \mathbf{t}_i \leq \mathbf{t}_{max}; i = 1, \ldots, m. \end{aligned} \tag{6.24}$$

where $\mathbf{1}$ is a vector of size $\mathbf{R}^m$ and with this solution, we are able to find the minimum tension while respecting the conditions of the admissible cable tension solutions. The objective function is desired to minimize a linear program while respecting the constraints. We note that this kind of optimization routine might not be real-time capable as we cannot guarantee the worst-case computation time [60]. The 1-norm solution also does not deliver continuous solutions for the tension values which might lead to jumps in the tensions required to be applied which is not very good for the actuators.

We note that the LP problem above is a special form of the general quadratic problem when we the Hessian matrix $\mathbf{H}$ is a zero matrix and the gradient vector is the linear relation $\mathbf{c}^T = \mathbf{1}^T$ and the constant $c0 = 0$

$$\begin{aligned} \underset{t}{\text{minimize}} \quad & \mathbf{t}^T\mathbf{H}\mathbf{t} + \mathbf{c}^T\mathbf{t} + \mathbf{c}0 \\ \text{subject to} \quad & \mathbf{W}\mathbf{t} = -\mathbf{f}_p \qquad \mathbf{t}_{min} \leq \mathbf{t}_i \leq \mathbf{t}_{max}; i = 1, \ldots, m. \end{aligned} \tag{6.25}$$

The author in [27] notes that active-set solvers help us to solve LP AND QP problems by moving the solution from one feasible point to another that has a lower objective function value. Since the objective function is convex and that the feasible space is convex, the LP and QP problems can be solved by standard convex optimisation methods. A draw back of this approach is that there is still very few libraries that implement Linear Programming program solvers in C++ and there also few of such libraries that are free and open-source.

Implementation 1-norm approach for tension distribution

For our implementation, we make use of two approaches of two different open source optimisation solvers.

1. **qpOASES** [61] is used as an LP solver implemented as a class member function within the plugin.

   (a) First we create a QProblem object:

   ```
   QProblem( int_t nV, int_t nC, HST_ZERO );
   ```

   This creates an LP problem as a special type of QP problem by specifying a zero Hessian matrix using the flag HST_ZERO while passing as parameters the size of the problem in terms of the number of decision variable nV = number of DOF and the number of constraints nC= number of cables

   (b) Then we initialise all the data structures that allow to solve the LP problem as a special type of QP problem:

   ```
   returnValue init( const real_t* const H,
       const real_t* const c,
       const real_t* const W,
       const real_t* const tmin,
       const real_t* const tmax,
       const real_t* const lbA,
       const real_t* const ubA,
       int_t&          const nWSR,
       real_t* const const cputime,);
   ```

   (c) Setting $lbA = ubA = \mathbf{f}_p$ changes the constraints to equality constraints. nWSR= 1000 is the number of working set recalculations which we use as a tuning parameter that to determine the maximum number of evaluations of the function and ultimately, the speed and precision of our control scheme.

   (d) $returnValue$ is a type that we use to test for feasible solution on the flag SUCCESSFUL_RETURN and if the solution is valid, we obtain this solution using the function:

   ```
     getPrimalSolution( t );
   ```

2. **CGAL** [62] solver is used as an LP solver within the service that is called when required during our loop calculation and is explained as follows:

(a) The process of creating a service in ROS has been detailed in [63]. We have created a ros node that provides a rosservice over the rostopic "SOLVETheQP".

(b) The messages for the service request and the response are contained in SOLVEQP.srv file within the srv folder as shown below: "SOLVETheQP"

```
 float64[ ] WrenchMatrix
 float64[ ] wVector
 ---
 float64[ ] tensions
```

(c) The service receives the wrench matrix $\mathbf{W}$ and the vector $\mathbf{f}_p$ as the input from the client request message and creates an LP program and solution using the following type definitions:

```
typedef CGAL::Linear_program_from_iterators
<double**,
 double*,
 CGAL::Const_oneset_iterator<CGAL::Comparison_result>,
 bool*,
 double*,
 bool*,
 double*,
 double*> Program;
typedef CGAL::Quadratic_program_solution<ET> Solution;
```

(d) The ET flag refers to an exact type object used for the calculation in this library. After data structure initialisation, we can use the linear program as follows:

```
          Program lp(nV,nC, W, -fp, r, fl, tmin, fu, tmax, c, c0)
```

(e) where the variables fl = fu = true are vectors of booleans of the size R^m which define the active constraints for the solver. The solution to the program is obtained using the function which returns an iterator to the solution vector that is returned to the client.

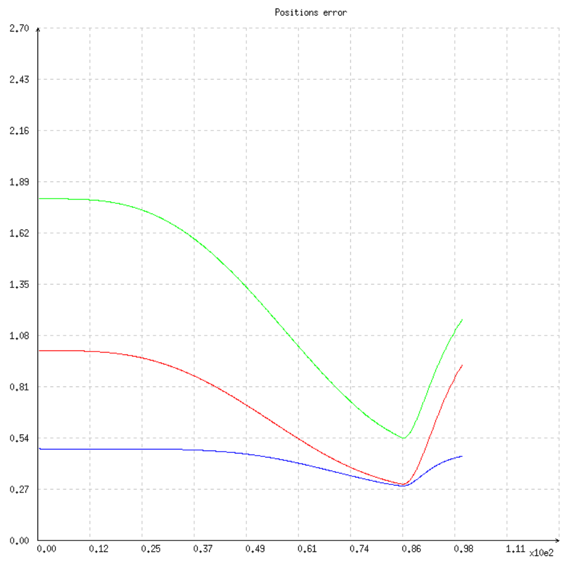
```
Solution s = CGAL::solve_linear_program(lp, ET());
```

3. After launching the robot on the terminal as describe previously, we can start up the controller using:

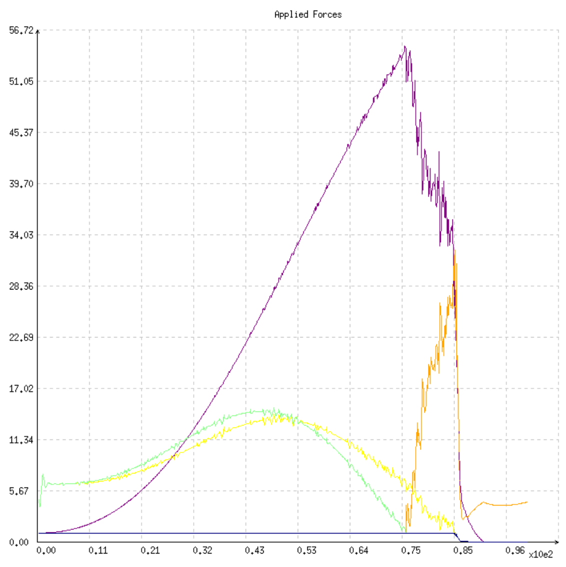
```
roslaunch cdpr3 control.launch
```

Results from LP approach

Below, we show the results obtained from the LP approach in terms of the position error and the applied cable tension along the entire trajectory. From the plots in figure 6.8, we see a decrease of the position error along the trajectory there is an error increase at the end indicating instability. We also see that the solver delivers positive tensions along the



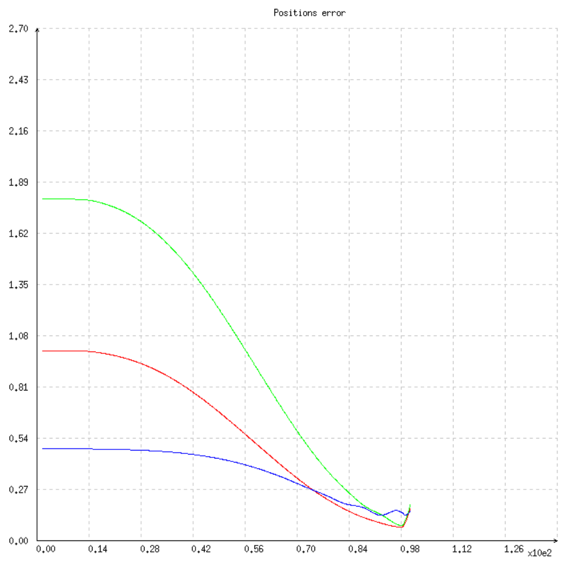
(a) Position error



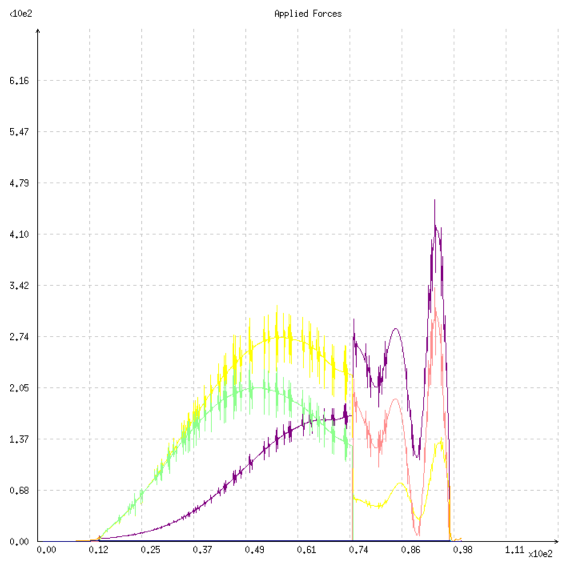
(b) Cable Tensions

Figure 6.8: Caroca Robot using LP solver $K_p = K_d = 100$ $t_{min} = 1$, $t_{max} = 6500$, $platformmass = 1$kg

trajectory but we also see jumps in these tension values indicating discontinuities experienced when using LP solver. The maximum calculated tension applied was 54N.



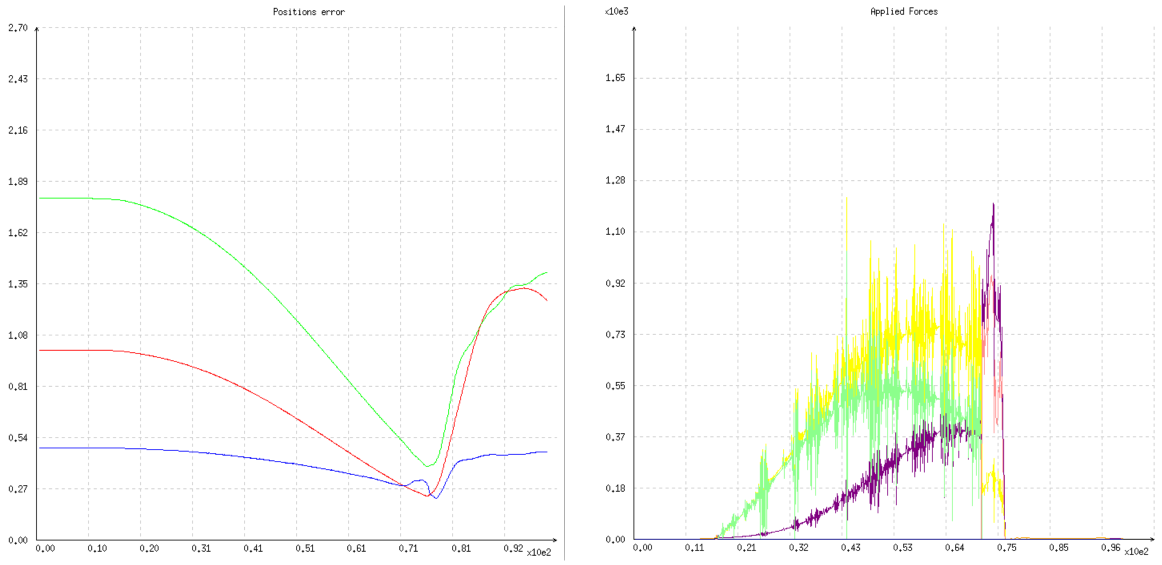
(a) Position error



(b) Cable Tensions

Figure 6.9: Caroca Robot using using LP solver $K_p = K_d = 800$ $t_{min} = 1$, $t_{max} = 6500$, platform mass = 1kg

In figure 6.9 we see that when we increase the K_p and K_d values further, we get a better convergence of the position error at a cost of even Higher tension values applied on the robot. We note that we are within the maximum allowable tension limits Max Calculated tension 456N.



(a) Position error (b) Cable Tensions

Figure 6.10: Caroca Robot using using LP solver $K_p = K_d = 1200$ $t_{min} = 1$, $t_{max} = 6500$, platform mass = 1kg

We have increased the gain further in figure 6.10 to see the limits of the simulator, we some instability in the control scheme and oscillation in of the position error. This is also seen in the forces applied on the cable where we see high forces with a maximum of 1224N even though the tensions are positive.

6.4.2 2-norm Optimal Solution

This formulates the tension distribution problem as a Quadratic programming (QP) problem and writes the 2-norm solution as [59]:

$$\begin{aligned} &\underset{t}{\text{minimize}} && \|\mathbf{t}\|^2 \\ &\text{subject to} && \mathbf{W}\mathbf{t} = -\mathbf{f}_p \qquad \mathbf{t}_{min} \le \mathbf{t}_i \le \mathbf{t}_{max}; i = 1, \ldots, m. \end{aligned}$$

We are able to minimise the energy used by the robot with this solution while respecting the conditions of the admissible cable tension solutions. The interesting thing about this approach is that it is real-time capable and delivers a continuous solution. The draw back is that there is still very few libraries that implement Quadratic Programming program solvers in C++ just like in the case of LP solvers and there are also few free and open source QP solvers.

Implementation 2-norm approach for tension distribution

From the relations in (6.25) which describe the general form of a quadratic programming problem, we note that our tension distribution is a special kind of such general form when the Hessian matrix, $\mathbf{H}$, is the identity matrix, the gradient vector $\mathbf{c}$ is a zero vector and the constant term $c0 = 0$. We use the same approach used for solving the LP problem by making few changes to the solvers that we have described in the previous section. The changes are described below:

1. If we want to use the QP solver that was implemented using the qpOASES library, we follow all the steps in the previous section while we make the following change that allows us specify the type of the Hessian matrix as the identity matrix :

```
QProblem( int_t nV, int_t nC, HST_IDENTITY );
```

2. If we want to use the solver that was implemented as a service using the CGAL library, we follow all the steps as in the previous section while we make the following change to use it as a QP solver.

```
typedef CGAL::Quadratic_program_from_iterators
<double**,
 double*,
 CGAL::Const_oneset_iterator<CGAL::Comparison_result>,
 bool*,
 double*,
 bool*,
 double*,
 double*> Program;
typedef CGAL::Quadratic_program_solution<ET> Solution;
```

3. We call the QP program in this method for this solver where we see that we now pass the Hessian Matrix as an argument:

```
        Program lp(nV,nC, W, -fp, r, fl, tmin, fu, tmax,H, c,c0)
```

4. the we solve the problem in this method within the service:

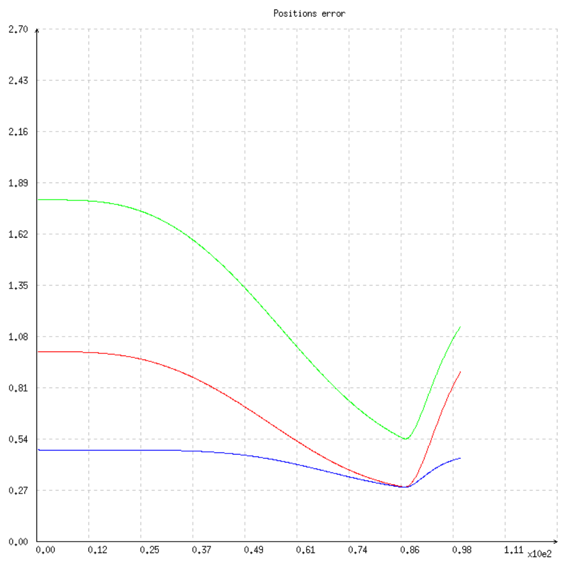
```
Solution s = CGAL::solve_quadratic_program(lp, ET());
```

Results from QP approach

Below, we show the results obtained from the QP approach in terms of the position error and the applied cable tension along the entire trajectory. From the plots in figure 6.12, we see a decrease of the position error along the trajectory there is an error increase at the end indicating instability. We also see that the solver delivers positive tensions along the trajectory but we also see jumps in these tension values indicating discontinuities experienced when using QP solver. The maximum calculated tension applied was 32N.

In figure 6.12 we see that when we increase the K_p and K_d values to 800, we get a better convergence of the position error at a cost of even Higher tension values applied on the robot. We note that we are within the maximum allowable tension limits max calculated tension 110N which is comparatively lower compared to the LP solver using the same gain.

We have increased the gain further in figure 6.13 to see the limits of the simulator. We see that the maximum calculated force is 342N, which is comparatively smaller when we


(a) Position error

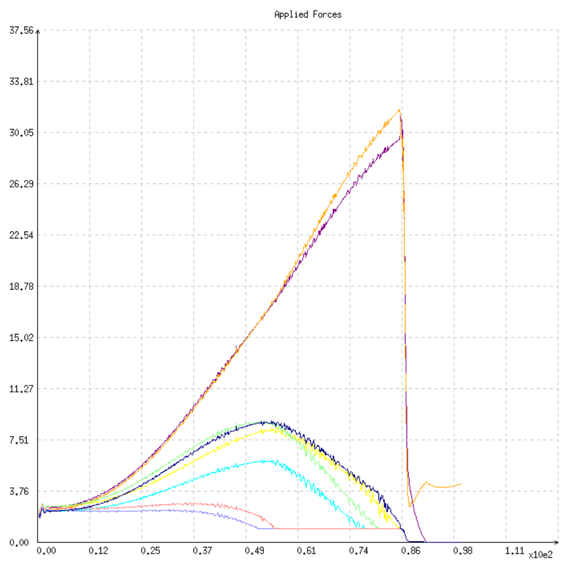
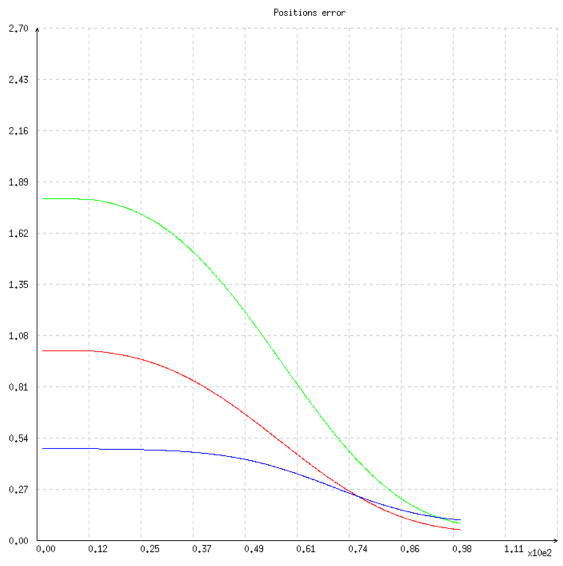

(b) Cable Tensions

Figure 6.11: Caroca Robot using QP solver $K_p = K_d = 100$ $t_{min} = 1$, $t_{max} = 6500$, $platformmass = 1$kg


(a) Position error

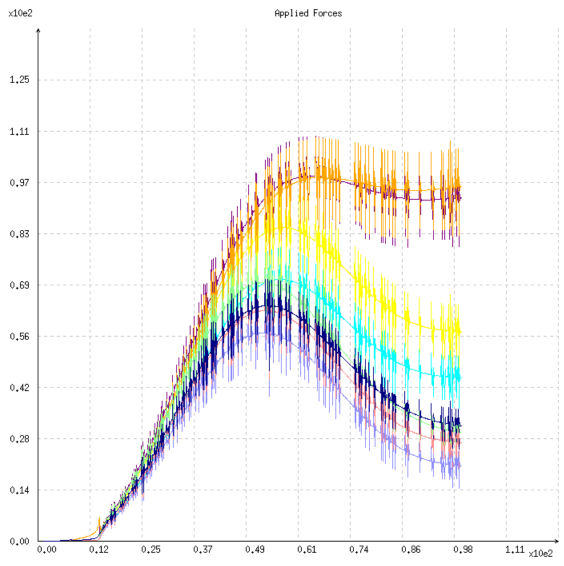
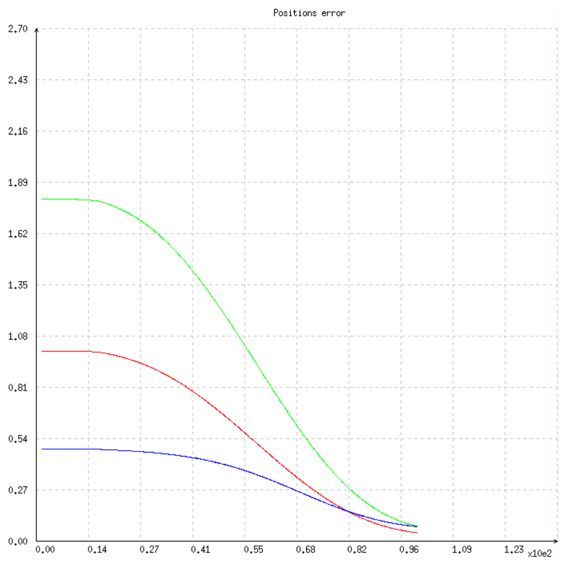

(b) Cable Tensions

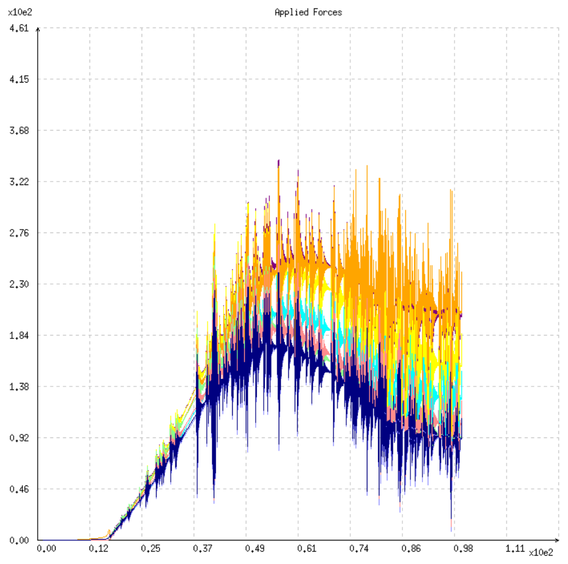
Figure 6.12: Caroca Robot using QP solver $K_p = K_d = 800$ $t_{min} = 1$, $t_{max} = 6500$, platform mass = 1kg

were using the same gain when we were using the LP solver approach. This tells us that QP approach can offer some advantages of using lower energy cable driven parallel robot systems. We also see a better convergence of the position error.

The figure 6.14 shows where we attempt to exceed the highest gain we have used on this controller to see if we can get an even better performance, we see that we do not get any appreciable improvement in performance from the system in terms of the positional error and the system starts to approach instability with big jumps in the cable tensions.

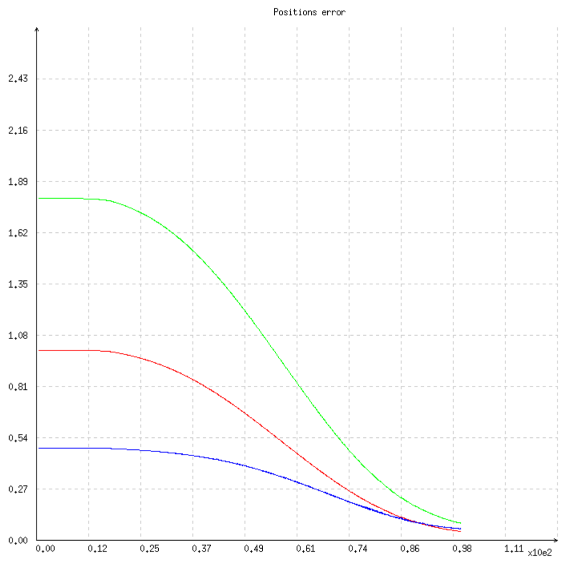


(a) Position error

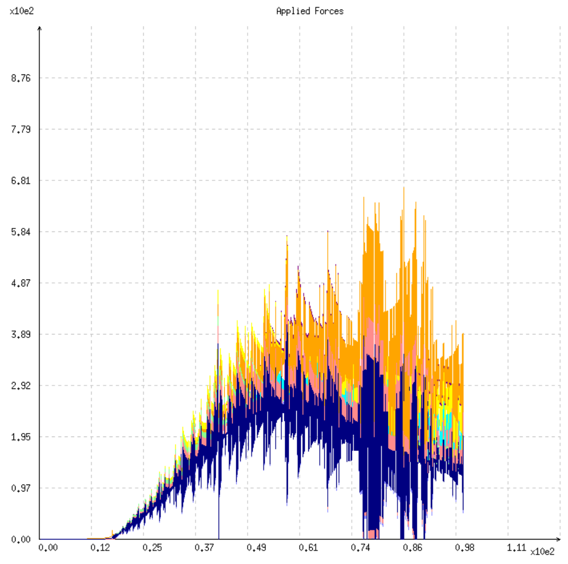


(b) Cable Tensions

Figure 6.13: Caroca Robot using QP solver $K_p = K_d = 1200$ $t_{min} = 1$, $t_{max} = 6500$, platform mass = 1kg



(a) Position error



(b) Cable Tensions

Figure 6.14: Caroca Robot using QP solver $K_p = K_d = 1500$ $t_{min} = 1$, $t_{max} = 6500$, platform mass = 1kg

6.4.3 Pseudo Inverse Solution

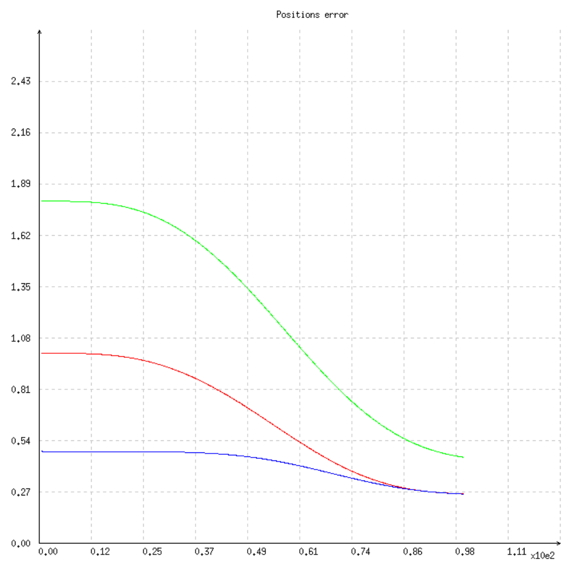
A possible solution to such an under-determined a system is pseudo-inverse solution which is given as:

$$\mathbf{t}_p = -\mathbf{W}^{+}\mathbf{f}_p \tag{6.26}$$

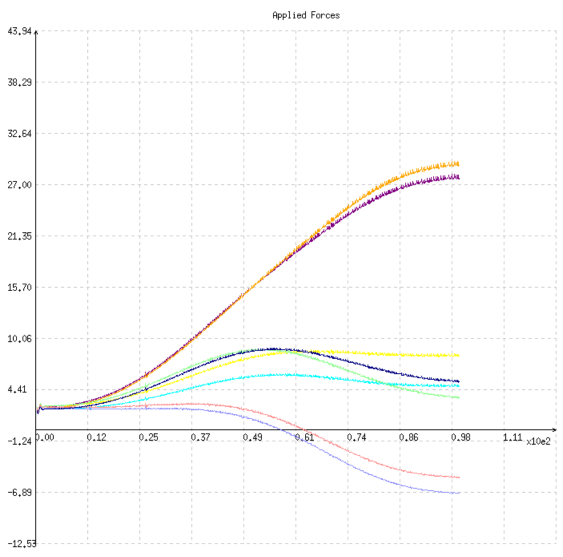
But the pseudo inverse solution can be less than optimal and can deliver solutions which are less optimal or containing negative tension values.

Results from Pseudo Inverse approach

Below, we show the results obtained from the pseudo inverse approach in terms of the position error and the applied cable tension along the entire trajectory.



(a) Position error



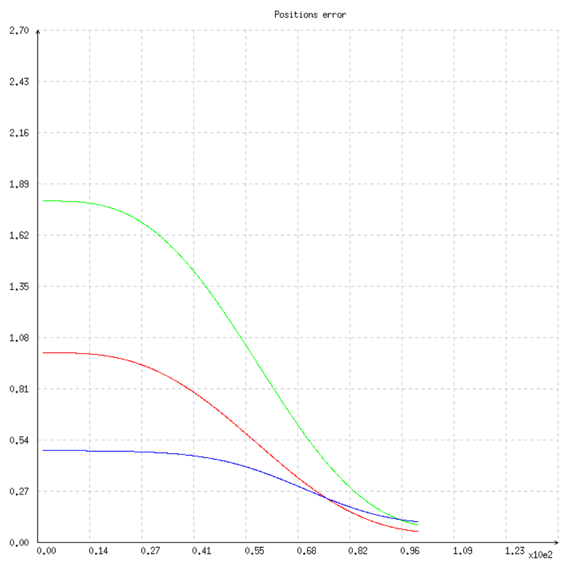
(b) Cable Tensions

Figure 6.15: Caroca Robot using pseudo inverse $K_p = K_d = 100$ $t_{min} = 1$, $t_{max} = 6500$, $platformmass = 1$kg

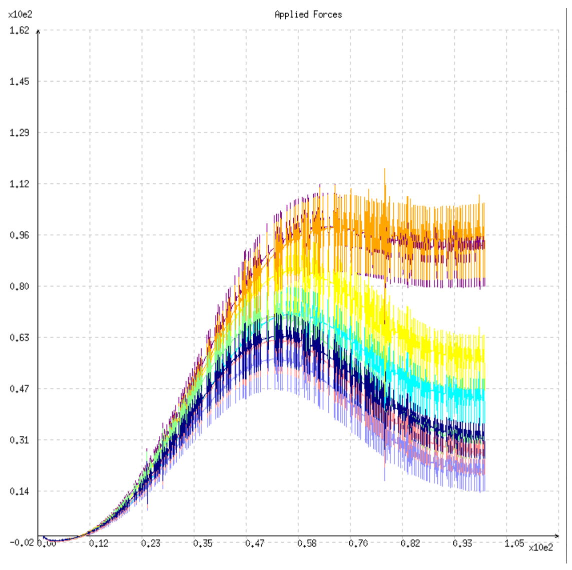
From the plots in figure 6.15, we see a decrease of the position error along the trajectory there is an error increase at the end indicating instability. We also see that the solver delivers negative tensions along the trajectory. The maximum calculated force here is 117N.

In figure 6.16 we see that when we increase the K_p and K_d values to 800, we get a better convergence of the position error at a cost of even Higher tension values applied on the robot. We note that at as can be seen at the beginning of the trajectory, we have negative tensions. This is due to the fact the we are using pseudo inverse solution. The maximum calculated force here is 117N.

We have increased the gain further in figure 6.17 to see the limits of the simulator. We see that the maximum calculated force is 117N, But we also see that we have negative tensions that produced by this calculation which is undesirable. Here also we see no better convergence of the position error due to the increased gain in the pseudo inverse approach.

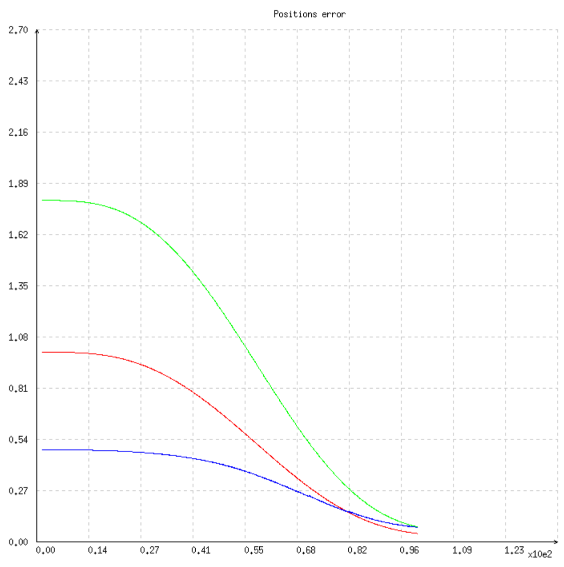


(a) Position error

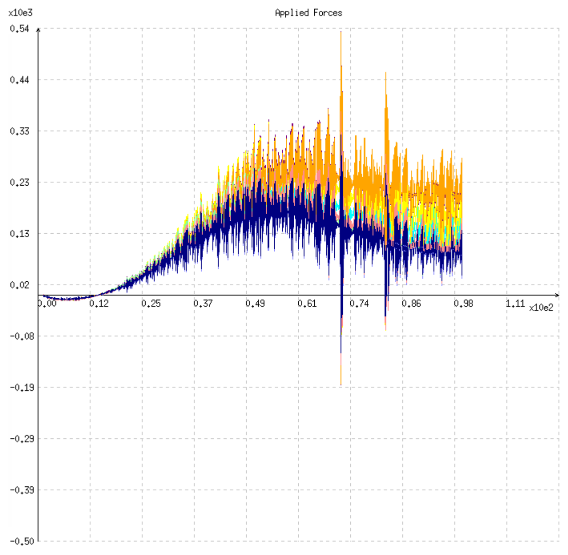


(b) Cable Tensions

Figure 6.16: Caroca Robot using pseudo inverse $K_p = K_d = 800$ $t_{min} = 1$, $t_{max} = 6500$, platform mass = 1kg



(a) Position error



(b) Cable Tensions

Figure 6.17: Caroca Robot using pseudo inverse $K_p = K_d = 1200$ $t_{min} = 1$, $t_{max} = 6500$, platform mass = 1kg

6.4.4 Closed-form Solution

The closed form solution [64] provides a real-time tension distribution by using an analytical technique to determine the solution of (6.5). The author performs a co-ordinate transformation of the mid-point of the cable tension limits.

$$\mathbf{t}_m = \frac{\mathbf{t}_{max} + \mathbf{t}_{min}}{2} \tag{6.27}$$

And then writes the solution as:

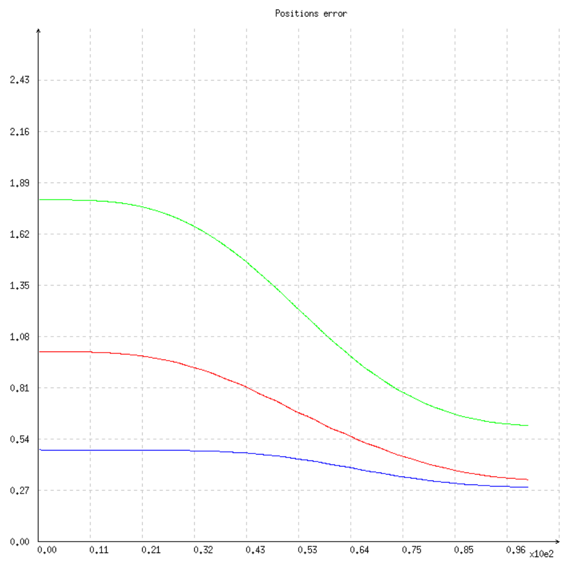
$$\mathbf{t} = \mathbf{t}_p + \mathbf{t}_v = \mathbf{t} - \mathbf{W}^{+}(\mathbf{f}_p + \mathbf{W}_m^{\mathbf{t}}) \tag{6.28}$$

The author notes that this technique might fail in some cases to provide admissible cable tension solution even though such a solution may exist when the condition :

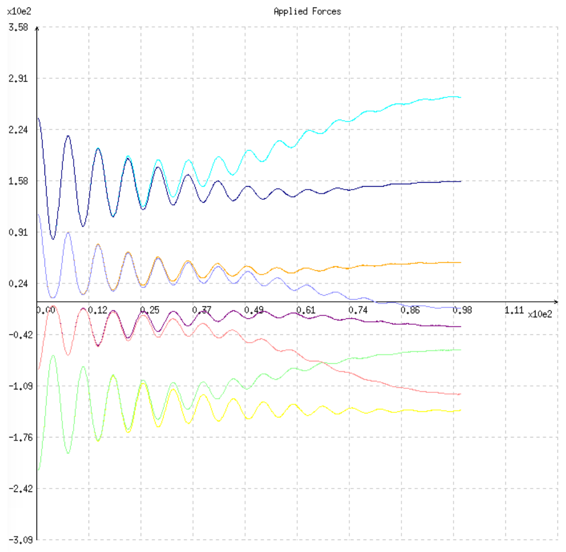
$$\frac{\mathbf{t}_m}{2} \leq \|\mathbf{t}_v\|_2 \leq \frac{\sqrt{m}\mathbf{t}_m}{2} \tag{6.29}$$

Results from Closed-form approach

Below, we show the results obtained from the closed form approach in terms of the position error and the applied cable tension along the entire trajectory.



(a) Position error



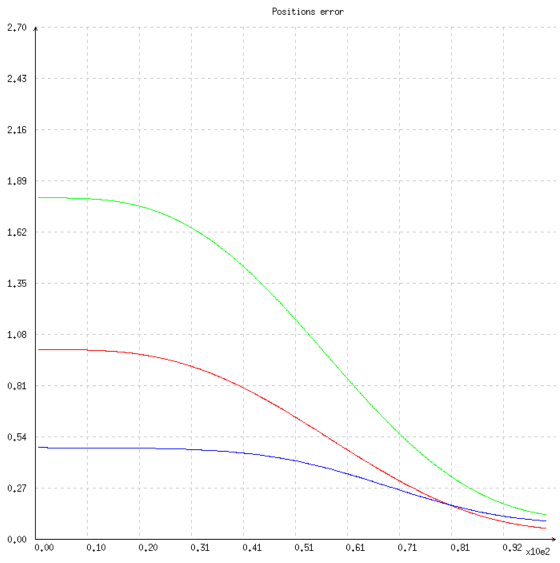
(b) Cable Tensions

Figure 6.18: Caroca Robot using closed-form approach $K_p = K_d = 100$ $t_{min} = 1$, $t_{max} = 6500$, $platformmass = 1$kg

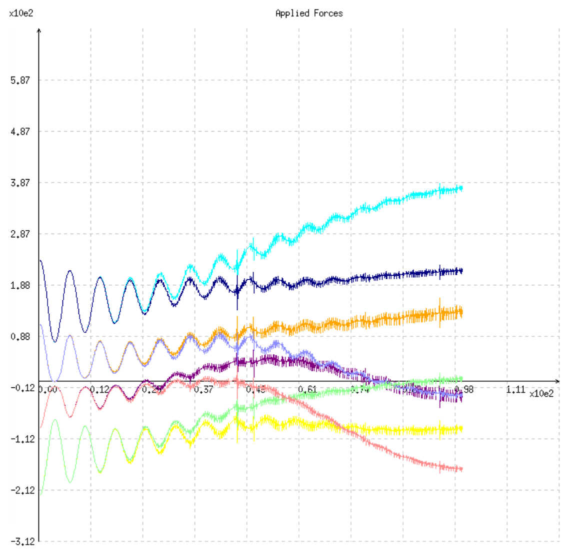
From the plots in figure 6.18, we see a decrease of the position error along the trajectory there is a steady-state error at the end indicating non-convergence. We also see that the solver delivers negative tensions along the trajectory. The maximum calculated force here is 267N.

In figure 6.19 we see that when we increase the K_p and K_d values to 800, we get a better convergence of the position error at a cost of even higher tension values applied on the robot. We note that we have negative tensions when using this solver even though there is a comparatively better convergence than when we are using the lower value of the gain. The maximum calculated force here is 385N.

We have increased the gain further in figure 6.20 to see the limits of the simulator. We see that the maximum calculated force is 517N, But we also see that we have negative tensions that produced by this calculation which is undesirable But we also see a slightly appreciable improvement in the position error.

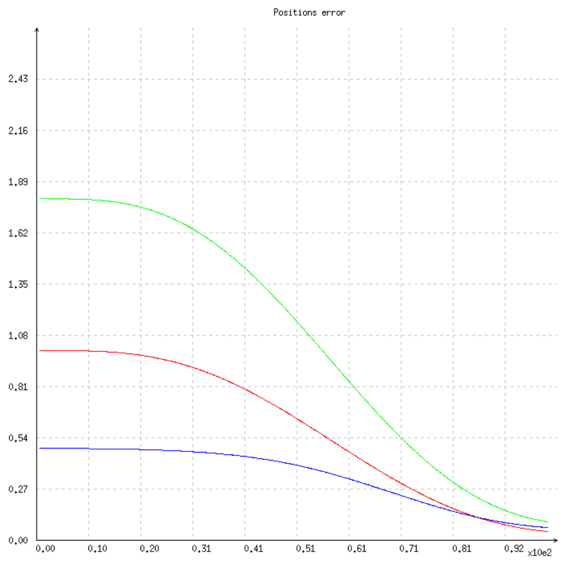


(a) Position error

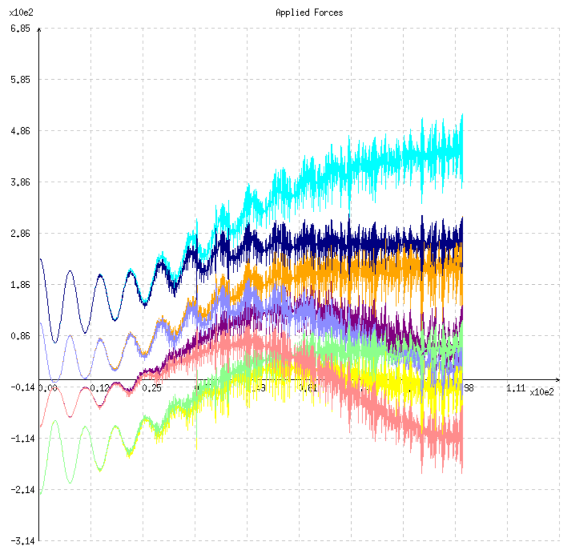


(b) Cable Tensions

Figure 6.19: Caroca Robot using closed-form approach $K_p = K_d = 800$ $t_{min} = 1$, $t_{max} = 6500$, platform mass = 1kg



(a) Position error



(b) Cable Tensions

Figure 6.20: Caroca Robot using closed-form approoach $K_p = K_d = 1200$ $t_{min} = 1$, $t_{max} = 6500$, platform mass = 1kg

6.4.5 Advanced Closed-form Solution

In order to overcome the problem seen in the closed-form solution when the conditions corresponds to (6.29), the author in [60] proposes an improved closed form solution which is called Advanced Closed form solution. We have implemented this algorithm in as part of the controller package for the cable driven parallel robot.

- First we calculate an estimate of the tensions in a loop using (6.29) and stop the loop if all conditions of (2.12) and (2.13) are met

- If a cable, i, fail the test, by being more than the maximum or less than the minimum, we reduce the redundancy by 1 and change the problem.
- we fix this particular cable, i, which is outside the limit to the minimum or maximum limit tension and change the problem as:

$$\mathbf{W}^{'}\mathbf{t}^{'} + \mathbf{f}_p^{'} = 0 \tag{6.30}$$

$$\mathbf{f}_p^{'} = t_{min}[\mathbf{W}]_i + \mathbf{f}_p \tag{6.31}$$

When the cable in question exceeded the minimum limit or

$$\mathbf{f}_p^{'} = t_{max}[\mathbf{W}]_i + \mathbf{f}_p \tag{6.32}$$

when the cable has exceeded the maximum limits

- where $\mathbf{W}^{'}$ and $\mathbf{t}^{'}$ are the wrench matrix and the cable tension vector respectively when we remove the $i-th$ column or row respectively
- This cycle is repeated in the loop until we find a feasible tension distribution or until the redundancy is 0.

6.4.6 Kernel Solution

A very popular approach is to make translation of the pseudo inverse solution so that all cable tensions lie on the positive side. This is done using the kernel of the wrench matrix as shown [65][55][3]:

$$\mathbf{t} = -\mathbf{W}^{+}\mathbf{f}_p + \lambda\mathbf{h} \tag{6.33}$$

where $\mathbf{W}^{+}$ is the moore-penrose pseudo inverse of the $\mathbf{W}$ matrix and its expression is given in (2.16). $\mathbf{h}$ is the kernel of the wrench matrix i.e. $\mathbf{h} = ker(\mathbf{W})$ and its columns form the null basis of the wrench matrix. λ is any arbitrary vector in the null space of the wrench matrix that can make the solutions positive. The challenge here is to make a choice for λ which allows for the solution in equation (6.33) to respect the conditions given in equations (2.12) and (2.13). A possible choice for λ can be [55]:

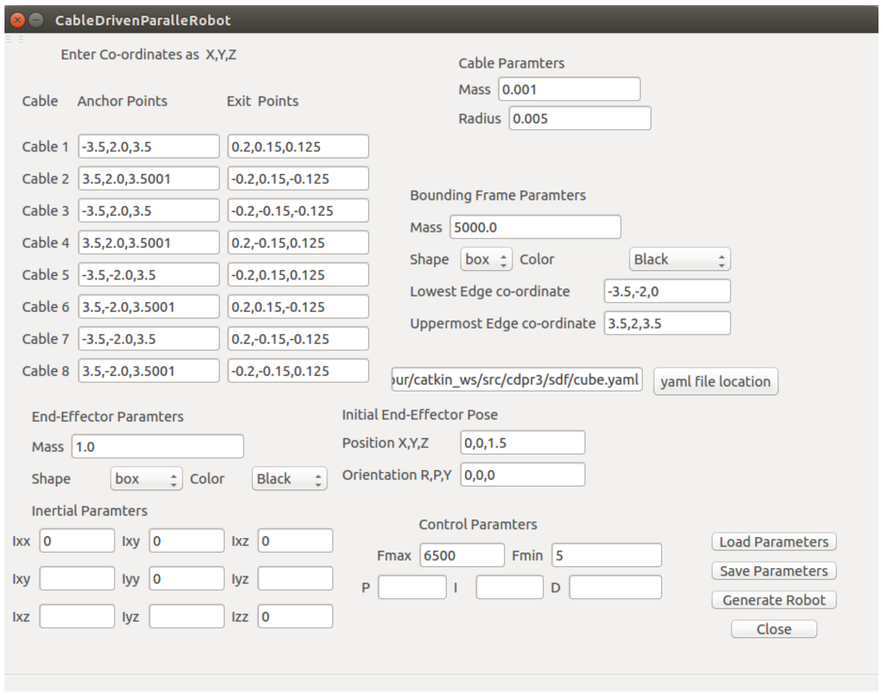
$$\frac{|t_{p,\mu_0}|}{h_{\mu 0}} = max\left\{\frac{|t_{p,\mu_0}|}{h_{\mu 0}} : 1 \leq \mu \leq m \wedge |t_{p,\mu_0}| < 0\right\} \tag{6.34}$$

$$\lambda = 2\frac{|t_{p,\mu_0}|}{h_{\mu 0}} \tag{6.35}$$

This approach works only where we can all positive or all negative components of the kernel of the wrench matrix $\mathbf{h}$ [25] and redundancy = 1 [55].

6.5 Graphical User Interface

In order to make the project more user friendly, we have developed a graphical user interface as shown below:



CableDrivenParalleRobot

Enter Co-ordinates as X,Y,Z

Cable	Anchor Points	Exit Points
Cable 1	-3.5,2.0,3.5	0.2,0.15,0.125
Cable 2	3.5,2.0,3.5001	-0.2,0.15,-0.125
Cable 3	-3.5,2.0,3.5	-0.2,-0.15,-0.125
Cable 4	3.5,2.0,3.5001	0.2,-0.15,0.125
Cable 5	-3.5,-2.0,3.5	-0.2,0.15,0.125
Cable 6	3.5,-2.0,3.5001	0.2,0.15,-0.125
Cable 7	-3.5,-2.0,3.5	0.2,-0.15,-0.125
Cable 8	3.5,-2.0,3.5001	-0.2,-0.15,0.125

Cable Paramters
Mass 0.001
Radius 0.005

Bounding Frame Paramters
Mass 5000.0
Shape box Color Black
Lowest Edge co-ordinate -3.5,-2,0
Uppermost Edge co-ordinate 3.5,2,3.5

ur/catkin_ws/src/cdpr3/sdf/cube.yaml

End-Effector Paramters
Mass 1.0
Shape box Color Black

Initial End-Effector Pose
Position X,Y,Z 0,0,1.5
Orientation R,P,Y 0,0,0

Inertial Paramters

Ixx	0	Ixy	0	Ixz	0
Ixy		Iyy	0	Iyz	
Ixz		Iyz		Izz	0

Control Paramters
Fmax 6500 Fmin 5
P I D

Figure 6.21: Graphical User Interface developed as part of this project

This interface enables a user to modify the robot parameters to suit their own robot and then save these parameters to the yaml file location as they wish. The user can also generate the robot SDF by clicking the appropriate button. Currently this plugin has been integrated into ROS user interface plug-in suite called rqt_plugins [66] . Users who do not have rqt_plugin installed may follow the installation process as detailed in [67]. After the installation, we can launch the rqt_gui using:

```
rosrun rqt_gui rqt_gui
```

Here we can find our cable driven parallel robot simulator interface as one of the plugins in this group. We can also find other ROS plugins that have been developed for other users or for other robots such as rosbag, rqt_plot etc. Thus we can use all these plugins in combination with our user interface to control and modify our robot.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

The study of cable driven parallel robot and robot simulators continues to be an ongoing field of research. Through this work, we have shown the possibility of developing a simulator that can be applied in the study and teaching of cable driven parallel robot.

In the first chapter we have made a brief study about the existing literature, the state of the art and research developments in the field of cable driven parallel robots. We also looked at the available options for use in the development of robot simulators and concluded on the reason for our choice of Gazebo and ROS.

In the second chapter, we have made an extensive study into how cable driven parallel robots are modelled. There we saw the Geometric, kinematic and dynamic models of a cable driven parallel robot. We also saw how some general properties such as the stiffness and workspaces of cable driven parallel robots are characterised.

In the third chapter, we made a study about the gazebo simulator and its properties that we can use for our development. We also saw how this software can be integrated with ROS in order to enable different software programs communicate at the same time in order to exchange information.

In Chapter four and five, we provide further detail about our work starting from the software installations needed to be done and hen we explain how we model our robot within gazebo in detail. Within this chapter we precise the assumptions that we have made for this modelling where we assume that the cables are light an inextensible thus simplifying the dynamic model.

In Chapter six, we talk about the control of the robot in order to perform some trajectories. We make use of a popular approach which is called the operational space dynamic controller whose architecture we detail in this chapter. We explain more about the trajectories used in the experiments and the problem of tension distribution in cable driven parallel robots. We explain the different approaches to tension distribution for cable driven parallel robots and we test these different approaches to tension distribution on our cable driven parallel robot. We also present the results of our different tests and make some observations that we got from these experiments.

7.2 Future work

This simulator is developed as a ROS package and is designed to be structured as a ROS package that can be available to other users. Currently, it requires an improvement in the coding style with addition of comments so that it can be more user/developer friendly. The coding for the controller and the trajectory require to be improved to conform to the ROS standards for writing ROS nodes or packages also.

Another possibility for future improvement is in the modelling of the robot SDF in gazebo where we have made assumption that the cable is light and inextensible. In reality cables have a mass and in order to make the simulator approach more to the reality, the cable dynamics must be reflected in the model. It is hoped also that with future developments in gazebo, there will approach a possibility of including pulleys in this model and subsequently to reflect the pulley dynamics.

Finally the user interface which has been developed enables us to change robot properties, but we expect that this interface can be extended to be more intuitive and give more options to the user.

Bibliography

[1] Robotics Smashing. Most advanced robotics simulation software overview. http://www.smashingrobotics.com/most-advanced-and-used-robotics-simulation-software/, 2015 (accessed January 1, 2016).

[2] Mathias Donadello. Cable driven parallel robot 3d simulator developed by means of matlab/simulink. https://www.youtube.com/watch?v=eO-EciibrkM, 2015 (accessed January 1, 2016).

[3] Johann Lamaury. *Contribution à la commande des robots parallèles à câbles à redondance d'actionnement*. PhD thesis, Université de Strasbourg, 2014.

[4] Imme Ebert-Uphoff, Philip Voglewede, et al. On the connections between cable-driven robots, parallel manipulators and grasping. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 5, pages 4521–4526. IEEE, 2004.

[5] Ana Lucia Cruz Ruiz, Stéphane Caro, Philippe Cardou, and François Guay. Arachnis: Analysis of robots actuated by cables with handy and neat interface software. In *Cable-Driven Parallel Robots*, pages 293–305. Springer, 2015.

[6] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2149–2154. IEEE, 2004.

[7] Open Source Robotics Foundation. Old versions (prior to gazebo4). http://gazebosim.org/tutorials?tut=install&ver=2.2&cat=install, 2015 (accessed January 5, 2016).

[8] Ricardo Tellez. Documentation ros wiki). http://www.theconstructsim.com/?p=3234, 2016 (accessed January 15, 2016).

[9] Tobias Bruckmann. *Auslegung und Betrieb redundanter paralleler Seilroboter*. PhD thesis, Universität Duisburg-Essen, Fakultät für Ingenieurwissenschaften≫ Maschinenbau und Verfahrenstechnik≫ Institut für Mechatronik und Systemdynamik, 2010.

[10] Clément GOSSELIN. Cable-driven parallel mechanisms: state of the art and perspectives. *Mechanical Engineering Reviews*, 1(1):DSM0004–DSM0004, 2014.

[11] James Albus, Roger Bostelman, and Nicholas Dagalakis. The nist robocrane. *Journal of Robotic Systems*, 10(5):709–724, 1993.

[12] Lawrence L Cone. Skycam-an aerial robotic camera system. *Byte*, 10(10):122, 1985.

[13] Nicholas G Dagalakis, James S Albus, B-L Wang, Joseph Unger, and James D Lee. Stiffness study of a parallel link robot crane for shipbuilding applications. *Journal of Offshore Mechanics and Arctic Engineering*, 111(3):183–193, 1989.

[14] Ying Mao and Sunil Kumar Agrawal. Design of a cable-driven arm exoskeleton (carex) for neural rehabilitation. *Robotics, IEEE Transactions on*, 28(4):922–931, 2012.

[15] Makoto Sato. Development of string-based force display: Spidar. In *8th International Conference on Virtual Systems and Multimedia.* Citeseer, 2002.

[16] Perry D Campbell, Patrick L Swaim, and Clark J Thompson. Charlotte^{TM} robot technology for space and terrestrial applications. Technical report, SAE Technical Paper, 1995.

[17] Guillaume Barrette and Clément M Gosselin. Determination of the dynamic workspace of cable-driven planar parallel mechanisms. *Journal of Mechanical Design*, 127(2):242–248, 2005.

[18] Cong Bang Pham, Song Huat Yeo, Guilin Yang, Mustafa Shabbir Kurbanhusen, and I-Ming Chen. Force-closure workspace analysis of cable-driven parallel mechanisms. *Mechanism and Machine Theory*, 41(1):53–69, 2006.

[19] Samuel Bouchard, Clément Gosselin, and Brian Moore. On the ability of a cable-driven robot to generate a prescribed set of wrenches. *Journal of Mechanisms and Robotics*, 2(1):011010, 2010.

[20] Venus Garg, Scott B Nokleby, and Juan A Carretero. Wrench capability analysis of redundantly actuated spatial parallel manipulators. *Mechanism and machine theory*, 44(5):1070–1081, 2009.

[21] Ethan Stump and Vijay Kumar. Workspaces of cable-actuated parallel manipulators. *Journal of Mechanical Design*, 128(1):159–167, 2006.

[22] Jean-Pierre Merlet. *Wire-driven parallel robot: open issues.* Springer, 2013.

[23] Marc Gouttefarde and Clément M Gosselin. Wrench-closure workspace of six-dof parallel mechanisms driven by 7 cables. *Transactions of the canadian society for mechanical engineering*, 29(4):541–552, 2005.

[24] K Azizian, P Cardou, and B Moore. Classifying the boundaries of the wrench-closure workspace of planar parallel cable-driven mechanisms by visual inspection. *Journal of Mechanisms and Robotics*, 4(2):024503, 2012.

[25] Rodney G Roberts, Todd Graham, and Thomas Lippitt. On the inverse kinematics, statics, and fault tolerance of cable-suspended robots. *Journal of Robotic Systems*, 15(10):581–597, 1998.

[26] Shiqing Fang, Daniel Franitza, Marc Torlo, Frank Bekes, and Manfred Hiller. Motion control of a tendon-based parallel manipulator using optimal tension distribution. *Mechatronics, IEEE/ASME Transactions on*, 9(3):561–568, 2004.

[27] Per Henrik Borgstrom, Nils Peter Borgstrom, Michael J Stealey, Brett Jordan, Gaurav S Sukhatme, Maxim Batalin, William J Kaiser, et al. Design and implementation of nims3d, a 3-d cabled robot for actuated sensing applications. *Robotics, IEEE Transactions on*, 25(2):325–339, 2009.

[28] Satoshi Tadokoro, Yoshio Murao, Manfred Hiller, Rie Murata, Hideaki Kohkawa, and Toshiyuki Matsushima. A motion base with 6-dof by parallel cable drive architecture. *Mechatronics, IEEE/ASME Transactions on*, 7(2):115–123, 2002.

[29] Aiguo Ming and Toshiro Higuchi. Study on multiple degree-of-freedom positioning mechanism using wires. i: Concept, design and control. *International Journal of the Japan Society for Precision Engineering*, 28(2):131–138, 1994.

[30] Alireza Alikhani and Mehdi Vali. Modeling and robust control of a new large scale suspended cable-driven robot under input constraint. In *Ubiquitous Robots and Ambient Intelligence (URAI), 2011 8th International Conference on*, pages 238–243. IEEE, 2011.

[31] Rémy Ramadour, François Chaumette, and Jean-Pierre Merlet. Grasping objects with a cable-driven parallel robot designed for transfer operation by visual servoing. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 4463–4468. IEEE, 2014.

[32] Pablo Blasco. Simulation in robotics. `http://fr.slideshare.net/pibgeus/simulation-in-robotics`, 2010 (accessed January 3, 2016).

[33] Noah J Needler. *Design of an Algae Harvesting Cable Robot, Including a Novel Solution to the Forward Pose Kinematics Problem.* PhD thesis, Ohio University, 2013.

[34] Olivier Kermorgant. A dynamic simulator for underwater vehicle-manipulators. In *Simulation, Modeling, and Programming for Autonomous Robots*, pages 25–36. Springer, 2014.

[35] Micaël Michelin, Cédric Baradat, Dinh Quan Nguyen, and Marc Gouttefarde. Simulation and control with xde and matlab/simulink of a cable-driven parallel robot (cogiro). In *Cable-Driven Parallel Robots*, pages 71–83. Springer, 2015.

[36] Serena Ivaldi, Vincent Padois, and Francesco Nori. Tools for dynamics simulation of robots: a survey based on user feedback. *arXiv preprint arXiv:1402.7050*, 2014.

[37] Andreas Pott. Forward kinematics and workspace determination of a wire robot for industrial applications. In *Advances in Robot Kinematics: Analysis and Design*, pages 451–458. Springer, 2008.

[38] Tobias Bruckmann, Dieter Schramm, Lars Mikelsons, Manfred Hiller, and Thorsten Brandt. *Wire Robots Part I: Kinematics, Analysis & Design.* INTECH Open Access Publisher, 2008.

[39] Saeed Behzadipour and Amir Khajepour. Stiffness of cable-based parallel manipulators with application to stability analysis. *Journal of mechanical design*, 128(1):303–310, 2006.

[40] Paul Bosscher, Andrew T Riechel, and Imme Ebert-Uphoff. Wrench-feasible workspace generation for cable-driven robots. *Robotics, IEEE Transactions on*, 22(5):890–902, 2006.

[41] K Azizian and P Cardou. The dimensional synthesis of spatial cable-driven parallel mechanisms. *Journal of Mechanisms and Robotics*, 5(4):044502, 2013.

[42] Jean-Pierre Merlet. Analysis of the influence of wires interference on the workspace of wire robots. In *On Advances in Robot Kinematics*, pages 211–218. Springer, 2004.

[43] ODE. Ode documentation. `http://opende.sourceforge.net/`, 2015 (accessed January 5, 2016).

[44] OpenGL. Opengl - the industry's foundation for high performance graphics. `https://www.opengl.org/about/`, 2015 (accessed January 1, 2016).

[45] OSRF. Sdf format specification. `http://sdformat.org/spec?elem=link`, 2015 (accessed January 5, 2016).

[46] Arios Synodinos. Parallel mechanism issues gazebo. `https://bitbucket.org/osrf/gazebo/issues/854/parallel-mechanisms`, 2015 (accessed January 5, 2016).

[47] Open Source Robottics Foundation. Documentation ros wiki). `http://wiki.ros.org`, 2016 (accessed January 15, 2016).

[48] Nate Koenig John Hsu. Gazebo roscon presentation). `http://www.osrfoundation.org/wordpress2/wp-content/uploads/2015/04/Gazebo-ROSCon-presentation.pdf`, 2012 (accessed January 15, 2016).

[49] Raj. Ubuntu installation). `http://www.itzgeek.com/how-tos/linux/ubuntu-how-tos/install-ubuntu-14-10-step-by-step-with-screenshot.html`, October 26, 2014 (accessed July 11, 2016).

[50] William Woodall. Ros jade installation). `http://wiki.ros.org/jade/Installation/Ubuntu`, May 18 2016 (accessed July 11, 2016).

[51] Open Source Robotics Foundation. Which combination of ros/gazebo versions to use. `http://gazebosim.org/tutorials?tut=ros_wrapper_versions&cat=connect_ros`, (accessed July 11, 2016).

[52] Dirk Thomas. Building and using catkin packages in a workspace. `http://wiki.ros.org/catkin/Tutorials/using_a_workspace`, Dec 18 2015(accessed July 11, 2016).

[53] Fabien Spindler. visp for ros jade. `http://wiki.ros.org/visp`, 20 June 2016 (accessed July 12, 2016).

[54] Éric Marchand, Fabien Spindler, and François Chaumette. Visp for visual servoing: a generic software platform with a wide class of robot control skills. *IEEE Robotics & Automation Magazine*, 12(4):40–52, 2005.

[55] Richard Verhoeven. *Analysis of the workspace of tendon-based Stewart platforms*. PhD thesis, Universität Duisburg-Essen, Fakultät für Ingenieurwissenschaften≫ Maschinenbau und Verfahrenstechnik, 2004.

[56] Pooneh Gholami, Mohammad M Aref, and Hamid D Taghirad. On the control of the kntu cdrpm: A cable driven redundant parallel manipulator. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2404–2409. IEEE, 2008.

[57] Erika Ottaviano, Cassino Lazio, Italy Vincenzo Gattulli, Francesco Potenza, and Pierluigi Rea. Modeling a planar point mass sagged cable-suspended manipulator. In *Proceedings of the 14th IFToMM World Congress*, pages 491–496.

[58] Kami ccolo. visp for ros jade. `http://wiki.ros.org/roslaunch`, 12 May 2015 (accessed July 13, 2016).

[59] So-Ryeok Oh and Sunil Kumar Agrawal. Cable suspended planar robots with redundant cables: controllers with positive tensions. *IEEE Transactions on Robotics*, 21(3):457–465, 2005.

[60] Andreas Pott. An improved force distribution algorithm for over-constrained cable-driven parallel robots. In *Computational Kinematics*, pages 139–146. Springer, 2014.

[61] H.J. Ferreau, C. Kirches, A. Potschka, H.G. Bock, and M. Diehl. qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, 2014.

[62] Michael Hemmer. Algebraic foundations. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.8.1 edition, 2016.

[63] Aniss Koubaa. Writing a simple service and client (c++). `http://wiki.ros.org/ROS/Tutorials/WritingServiceClient(c%2B%2B)`, 2016 (accessed July 20, 2016).

[64] Andreas Pott, Tobias Bruckmann, and Lars Mikelsons. Closed-form force distribution for parallel wire robots. In *Computational Kinematics*, pages 25–34. Springer, 2009.

[65] Marc Gouttefarde, Johann Lamaury, Christopher Reichert, and Tobias Bruckmann. A versatile tension distribution algorithm for-dof parallel robots driven by cables. *Robotics, IEEE Transactions on*, 31(6):1444–1457, 2015.

[66] Scott Livingston. rqt plugins in ros. `http://wiki.ros.org/rqt`, 2016 (accessed July 24, 2016).

[67] Juan Purcalla Arrufi. Installing rqt on ubuntu. `http://wiki.ros.org/rqt/UserGuide#Installation`, 2016 (accessed July 24, 2016).

View publication stats