Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Fachbereich Informatik
Department of Computer Science

Master Thesis

Accurate trajectory tracking on the KUKA youBot manipulator: Computed-torque control and Friction compensation

Jeyaprakash Rajagopal

A thesis submitted to the
University of Applied Sciences Bonn-Rhein-Sieg
for the degree of
Master of Science in Autonomous Systems

Supervisors : Prof. Dr. Paul G. Plöger
Prof. Dr. Roustiam Chakirov
M.Sc. Sven Schneider

Submitted : Aug 15, 2018

I, the undersigned below, declare that this work has not previously been submitted to this or any other university, and that unless otherwise stated, it is entirely my own work.

DATE

Jeyaprakash Rajagopal

ABSTRACT

Accurate trajectory tracking control is a desirable quality for the robotic manipulators. Since the manipulators are highly non-linear due to the presence of structured (i.e. model errors), and unstructured uncertainties (i.e. friction), it becomes very difficult to achieve high-precision tracking in the manipulator joints. Most common solution to this problem is the computed-torque control scheme where the inclusion of the dynamic model linearizes the non-linear system. Disadvantage of this method is that the controller's performance in high-speed operations relies heavily on the accuracy of the dynamic model parameters. Model parameters provided by the manufacturers are generally not accurate, and thus the identification of link parameters becomes necessary. Additionally, inclusion of friction compensation terms in the dynamic model improves the controller's performance, and helps us in achieving better dynamics control of the manipulator. This work considers the implementation of both identification of the dynamic model parameters and computed-torque controller. The identification procedure is the continuation of the previous research where the geometric relation semantics and the dynamics of the system needed correction. Real concern in any of the robotics based applications is safety of the hardwares used (i.e. motors, sensors) because they are generally quite expensive. So, this work gives highest priority regarding safety of the real system and the safety control layer monitors the state of the joints with the help of the encoder data such as joint positions, velocities and torques. The model-based controller uses the dynamic model of the youBot manipulator and feed-forward torques are computed by using the inverse dynamics solver based on a library. There are two kinds of control schemes considered in this work such as the basic and the alternate control. The alternate control method is chosen over the basic method, because the basic approach suffers in predicting the model torques due to the presence of inaccurate dynamic model parameters. Both of the schemes implement the same cascade PI controller which is the combination of both the position and velocity controllers for the purpose of better disturbance rejection. Controller gains are tuned empirically to the optimum for most controlwise challenging joints on the end of the kinematic chain. This work reports analysis on the semantics used by the existing rigid-body algorithms and these findings are useful in constructing the geometrical relation semantics between rigid bodies without introducing the logical errors. Then, a safety control check is performed in the real system that handles the breaches in the safety limit of the manipulator joints effectively with an understandable latency issues due to the use of non real-time operating system. The correctness of the dynamic model is tested with the gravity compensation task, and then the pure controller without the dynamic model is validated with the analytical trajectories. The friction modelling and compensation experiments are conducted in the basic control scheme implemented in this work. The computed-torque control scheme is evaluated on the farthest joints of the base with the help of the analytically formulated trajectories. In spite of using the inaccurate model parameters in the dynamic model, the controller tracks of the trajectory accurately with an acceptable tracking error on the manipulator joints.

NOTATIONS AND ACRONYMS

Notations

$\boldsymbol{q}, \boldsymbol{\theta}$	Joint Position
$\boldsymbol{\dot{q}}, \boldsymbol{\dot{\theta}}, \boldsymbol{qd}$	Joint velocity
$\boldsymbol{\ddot{q}}, \boldsymbol{\ddot{\theta}}, \boldsymbol{qdd}$	Joint acceleration
$\boldsymbol{q_d}, \boldsymbol{\dot{q}_d}, \boldsymbol{\ddot{q}_d}$	Desired joint position, velocity and acceleration
$\boldsymbol{q_m}, \boldsymbol{\dot{q}_m}, \boldsymbol{\ddot{q}_m}$	Measured joint position, velocity and acceleration
$\boldsymbol{\tau}$	Joint torque
$\boldsymbol{\tau_m}$	Model torque
$\boldsymbol{\tau_f}$	Friction torque
$\boldsymbol{\tau_c}$	Torque from the controller
$\boldsymbol{rad}$	Radians
$\boldsymbol{rad/s}$	Radians per second
$[\boldsymbol{A}]$	Orientation frame related to A
$\boldsymbol{\{A\}}$	Frame related to A
${}^{\boldsymbol{A}}\boldsymbol{T_B}$	Homogeneous transform from B to A
${}^{\boldsymbol{A}}\boldsymbol{X_B}$	General spatial transform from B to A
$\boldsymbol{\psi}$	Dynamic model parameters

Acronyms

DoF	**D**egrees **o**f **F**reedom
CoM	**C**enter **o**f **M**ass
Orocos	**O**pen **Ro**bot **Co**ntrol **S**oftware
KDL	**K**inematics and **D**ynamics **L**ibrary
PID	**P**roportional **I**ntegral **D**erivative
SP	**S**et **P**oint
PV	**P**rocess **V**ariable
CV	**C**ontrol **V**ariable
3-D	3-**D**imensional
w.r.t.	**w**ith **r**espect **t**o

ACKNOWLEDGMENTS

I dedicate this thesis to my family and friends and I would like to thank all my supervisors for the guidance. My sincere thanks to Prof. Dr. Paul G. Plöger for the support and guidance on both the RnD's and master thesis. I am thankful to Prof. Dr. Roustiam Chakirov for the valuable insights on my thesis. I am really grateful to M.Sc. Sven Schneider for keeping me in the right track throughout the course of these projects. I am grateful my fellow classmates of Master of Autonomous Systems for their support. My parents were also supporting during the course of MAS; I am grateful to them.

CONTENTS

Chapter 1

INTRODUCTION

The industrial manipulators are highly non-linear dynamic systems in general, so it becomes difficult to achieve the high-performance trajectory tracking. Though many methods have been proposed by different authors, the computed-torque control is the the most common control scheme that is suitable for this kind of a problem. In spite of having many process control methods for the industrial robots, the PID controller is selected as the process control method and it is briefly explained in the next section. The computed-torque control method is advantageous than the pure PID control in many aspects such as the trajectory tracking control performance, lower feedback gains, lower energy consumption and the compliant motion [16]. This work aims to achieve the advanced model based controller scheme where the linear feedback control system is mandatory. There are two important aspects considered in the computed-torque control are the control law and dynamic model. Though this method is effective in accurate trajectory tracking, the manipulator is subject to uncertainties even in a well-defined industrial setup such as the accuracy of the model parameters and the unmodeled dynamics [18]. The first uncertainty in the manipulator is that the model parameters provided by the manufacturers, these parameters are not accurate-enough and the findings on the same are presented by many researchers [4] [3]. The second uncertainty in the manipulator is unmodeled dynamics i.e. friction. The accurate model of the robot is mandatory as discussed in the previous section and the unavailability of the joint-torque sensors [10] in the youBot manipulator joints makes the tracking control problem even harder. Apparently, there are only limited advantages of the computed-torque control strategy where the accuracy of the model parameters are not good enough. So, the accurate model parameters of the links are mandatory to solve this problem and it has been widely investigated by many researchers [3] [25] where the authors propose the identification procedure that can be used to estimate the model parameters of the system but there are many challenges involved in achieving it due to the presence of the implicit conventions, semantic problems and the trajectory selection criteria. The modified recursive Newton-Euler formulation is used for this purpose and then model fitting methods can be used to identify the model parameters of the links. This work attempts to deploy the estimated model parameters along with the friction compensation that brings us close in obtaining the complete dynamics of the system which in-turn provides the robust trajectory tracking in high-speed operations. The control can be considered in two different spaces such as

- **Joint space control** The joint controller contains the feedback control system that makes sure that the execution of the expected motion is being closely followed by the joint coordinates.
- Whereas **in task/operational space control**, it is important to achieve the precise control of the end-effector motion for which the compliant motion control is necessary. The compliant control is nothing but having a proper algorithm that relates the interaction of the end-effector and the environment appropriately.

This work uses the joint space controller, so the torques can be commanded to the manipulator joints for executing the desired motion and it does not regulate the forces. The commanded torques go through an additional safety control layer that is implemented around the computed-torque control scheme. This layer performs the checking in all the three modes such as position, velocity and torque which depends particularly on the encoder data before applying the commanded torques to the manipulator joints. Feed-forward torques are generated through ID solver which is readily available in orocos KDL [23] library and the kinematic chain of the youBot manipulator needs to be created manually. There are two possible independent-joint control designs that can be used in a controller such as the independent single-joint and multi-joint model [7]. The independent single-joint model considers the dynamics of a single joint and it's response. Whereas the multi-joint model based control design considers the complete dynamic model of the robot. This work uses the multi-joint model where the interaction among the joints are handled. The desired end-effector motion can be executed by commanding the appropriate torques to the manipulator joints which depends on the motion plan in the joint level.

Mode of control

The joint can be commanded with three kinds of inputs such as position, velocity and torque. So, a joint can be controlled in three different modes as depicted in Fig. 1.1.

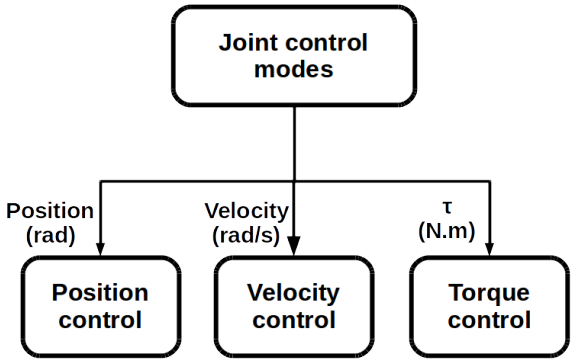


Figure 1.1: Joint's mode of control

In position mode, a joint is commanded with position (joint angle in radians) to move the joint from a particular position to an another position. Whereas, the velocity mode commands the velocity to the joints for achieving the desired motion. The final classification is the torque control mode that commands the torque to the manipulator joints. The trajectory tracking errors can be efficiently minimized via torque mode since the dynamics and the joint configuration is considered [4]. Since, the motor takes the current as an input, the commanded joint torque has to be converted to target currents using a motor model based on the following equation (1.1).

$$Current = (Commanded\_torque \times Gear\_ratio)/Torque\_constant(ampere) \tag{1.1}$$

The inverse of the above equation converts the current that is measured from the motors into joint torque.

$$Torque = (Current \times Torque\_constant)/Gear\_ratio(Newtonmeters) \tag{1.2}$$

The current to torque conversion as specified in the equation (1.2) is particularly useful in interpreting the results of the system easily. The equations (1.1) (1.2) are given based on the youbot driver[1]that is used in this work. The torque control is quite robust and follows the trajectories in an accurate manner. This work uses the advanced model based torque controller and it is implemented in the joint space rather than the task space. This helps us in executing/tracking the trajectories in an accurately. This work assumes that the existing joint level controller gains are optimum hence the tuning the controller gains in the joint level is not the primary focus.

Control systems overview

Basic definitions of this section are referred from [21] [30]. A system is considered to be linear when a change in output is directly proportional to the change in input. Since the robotic systems are highly non-linear, it is important to have the linear control strategy. This is achieved by using the linearization techniques that solves the non-linear systems problem. There are two kinds of control systems available such as open loop and the closed loop systems. Open loop, closed loop systems are depicted in the figures 1.2, 1.3 respectively. In the open loop systems, inputs are given to the system and the outcome is accepted regardless the deviations in the desired behaviour (feedback is not considered).

[1] `https://github.com/youbot/youbot_driver`

Figure 1.2: Open loop system

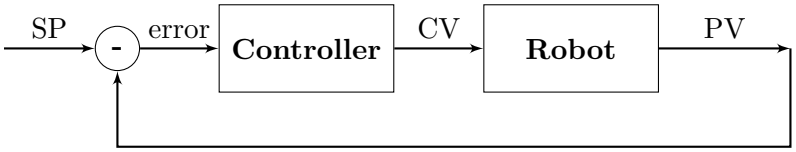


Figure 1.3: Closed loop system

Whereas, the closed-loop control systems account the feedback of the system to minimize the deviations between the desired and observed data. This work uses the closed loop system for achieving the precise control of the manipulators.

PID controller

The PID controller is the most commonly used controller in many of the industrial applications due to it's simplicity and it offers a lot of advantages such as fast response, brings the steady state error close to zero, avoiding oscillations and to achieve the stability in the system [26] [7]. It is possible to change the steady state response of the closed-loop system using this controller. The PID controller computes the error value between the desired and measured set-points repeatedly and it applies the correction based on the proportional(P), integral(I) and derivative(D) terms. P-term represents the current error that is being observed. I-term represents the history of errors which will be used to minimize the error between the expected set-point and it's response. D term is considered to predict the error based on it's rate of change in error. D term counteracts the overshoot caused by the P and I terms. The general mathematical representation of the complete PID control function can be expressed as follows

$$PV(t) = K_p \cdot error(t) + K_i \cdot \int_0^t e(t')dt' + K_d \cdot \frac{de(t)}{dt} \tag{1.3}$$

It is possible to use different combinations within this controller depending on the requirements such as

- **P only control** is used in some specific applications where the feedback is expected to be constant.
- **PI control** is commonly used controller.
- **PD control** is used to control the servo motors.

The terms that are commonly used in the controller is depicted in the figure 1.3 and it is explained below

- Process Variable(**PV**) is the actual feedback that is being fed back to the controller from a process/system.
- Set Point(**SP**) is the desired variable for the process variable that is discussed above.
- Control Variable(**CV**) is the actual output of the controller where the P, I and D terms are getting computed.
- Error(**e**) is computed by finding the difference between SP and PV which can be a position or velocity set-point.
- K_p, K_i, K_d represents the proportional, integral and the derivative gains of the controller.
  - If K_p is high, the system oscillates a lot before reaching the desired setpoint which inturn introduces an offset in the system.
  - K_i tend to counteract this offset introduced by P control. If this gain value is high, the setpoint reaches the process variable faster.
  - K_d tends to keep the stability of the system in control.

P control

Proportional(P) control [26] is the linear feedback control system which produces an output that is proportion to the present error value(i.e. position set-point versus the control output). The proportion of the controller can be adjusted by multiplying the error with the proportional gain. By increasing the proportional gain, it is possible to reach the steady-state error faster but it must be considered with caution that the overshoot can occur with the higher P gains. This kind of control is applicable only when the system has a tolerance with the constant steady state error and the oscillations are not a big concern. The disadvantages of using the P controller are a lot of deviations where the stability of the system is compromised an the system overshoot.

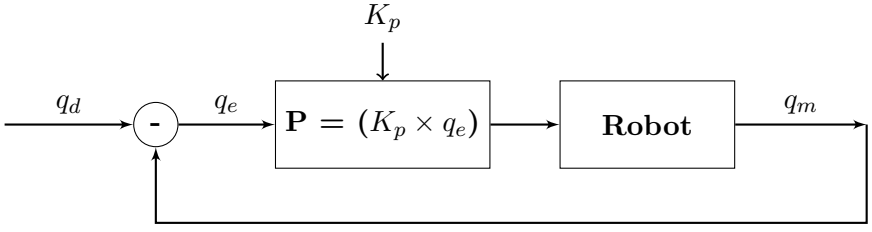


Figure 1.4: P Only controller

PI control

PI control depicted in Fig. 1.5 eliminates the steady state error from the P controller. The history are errors are accounted by the I controller to close the gap between the step input and it's response in the system. The P and I control terms together makes the system follow the step response quite closely. This reduces the overshooting and it helps achieving the stable system. This kind of a controller has no ability to predict the future errors(the rate of change in the error).

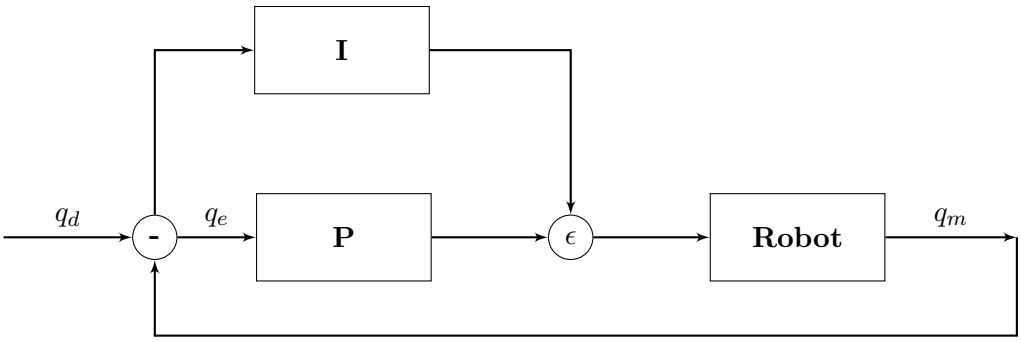


Figure 1.5: PI controller.

PD control

P-D controller depicted in Fig. 1.6 is used to increase the stability of the system with the inclusion of the derivative control or the damping factor to the proportional control.

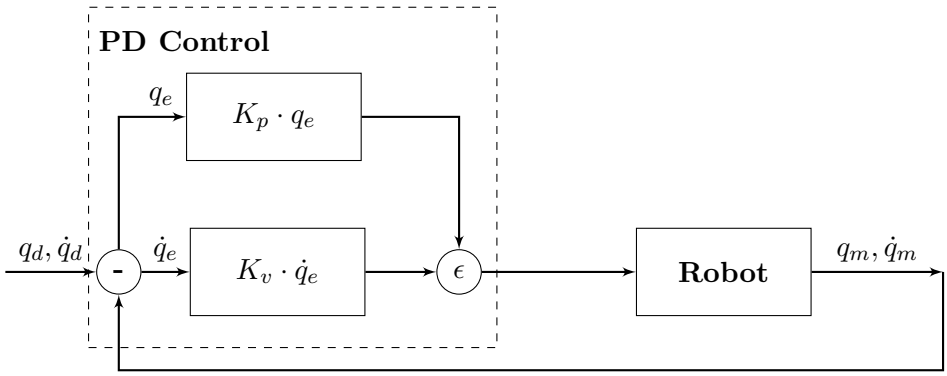


Figure 1.6: PD control

Friction modeling and compensation

An important aspect that needs to be accounted in the dynamic model of the manipulator is friction [17]. Friction is a highly non-linear phenomenon that resists the relative motion of the rigid bodies and it can be classified into two categories such as static, kinetic friction. At first, the static friction is present when a rigid body attempts or putting

an effort to move. Whereas dynamic or kinetic friction occurs when the rigid body is in motion and these two techniques are explained in detail in the preceding sections of this report. The frictional effect influences the manipulator performance in both the static and the dynamic configurations which could cause instability in the control. The friction can be compensated in two different ways such as model and non-model based scheme. The static friction estimates obtained manually through the automated testing in the previous work [19] for the youBot manipulator and it is particularly useful in adding the compensation terms with the dynamic model of the manipulator.

Report structure

The report outline is given as follows

- Section 2 state of the art on identification methods.
- Section 3 describes the hardware, software platform and the approaches used in this work. It discusses semantics of the rigid-body algorithms, the computed-torque control along with the friction modeling and compensation terms, finally trajectory generation method for evaluating the controller.
- Section 4 discusses and presents the experimental results of this work.
- Section 5 presents the conclusions and future work.

Chapter 2

STATE OF THE ART

The dynamic equations of kinematic chains of rigid bodies [25] represents the relationship between the motion of the system and the forces that cause it [8]. It can be considered as a mathematical model that computes forces and acceleration of the system which are relevant to the dynamics of the system. The computation of accelerations and forces based on two different algorithms can be named as forward and inverse dynamics respectively. Forward dynamics computes the accelerations based on the given joint position, velocity, torque inputs to produce the desired acceleration response in the system. It is mostly used in the simulation based systems. Whereas, inverse dynamics compute forces based on the given joint position, velocity and acceleration. These two algorithms have the system model that represents the physical properties of a system and the accuracy of the computations depend heavily on the model parameters. The algorithm that combines both the forward and inverse dynamics is referred as hybrid dynamics [11] where both the acceleration and force inputs are given to achieve a particular task. The equations of motion can be obtained through two different formulations such as the Newton-Euler and Lagrangian. Recursive Newton-Euler formulation is the most commonly used method [23] [11] since it is computationally efficient. The inverse dynamics computes the motion information of the rigid-body systems through the forward recursion that computes velocities and accelerations from proximal to distal links and a backward recursion is followed to compute the forces, torques from distal to proximal links. In [11], author uses spatial vector form that combines the linear and angular aspects of motions, forces into a single six-dimensional vector and the computational efficiency increases with it. Complete dynamics of the system can be achieved by considering the uncertainties such as model errors, unmodeled dynamics [18] which are present in the system. Model errors can be classified into kinematic and dynamic model errors but this work considers only dynamic model parameters. In [3], author modifies the recursive Newton-Euler formulation to a linearized matrix-form for identifying the model parameters and the same method is used in the previous work [19] where geometrical relation semantics of the rigid body are inaccurate and acceleration of the distal link that is being transmitted across joints due to the movement of the current joint is not accounted properly. In most of the rigid-body algorithms, inertial frame is attached to CoM of the rigid-body, but it is shifted to the joint origin by using the parallel-axis theorem. Re-expressing the inertia w.r.t. joint origin is essential due the unknown location of the CoM. The implementation is based on the coordinate calculations and the

semantics checking is not the part of it. The numerical calculations consider the integration as an addition of two matrices but the semantics check ensures the physical meaning of those operations. The operations involved with the pose, motion and force transforms need to be semantically correct where the constraints are also applicable but the only use of numerical computations can introduce the logical errors in the system [1] [14]. The unknown dynamic model parameters are mass, CoM and moments of inertia and these parameters can be identified with the use of ridge regression. The kinematic chain that is created in the dynamic model based on Orocos KDL [23] needs correction in the frame assignments since the frame diagram in KDL is not matching with it's implementation. The friction phenomenon is one of the important effects that is present in the unmodeled dynamics. Static friction modeling has been done in the previous work [19]. Friction modelling i.e. static friction models and the compensation [5] techniques are discussed [20]. Evolution of static friction model started with the static friction and Coulomb friction [17] which is the most simplified version of friction [25]. Then viscous friction is added with the static and Coulomb. Finally, static friction model includes the Stribeck effect along with the static and viscous friction. The model of friction that includes Stribeck friction is shown in the following figure

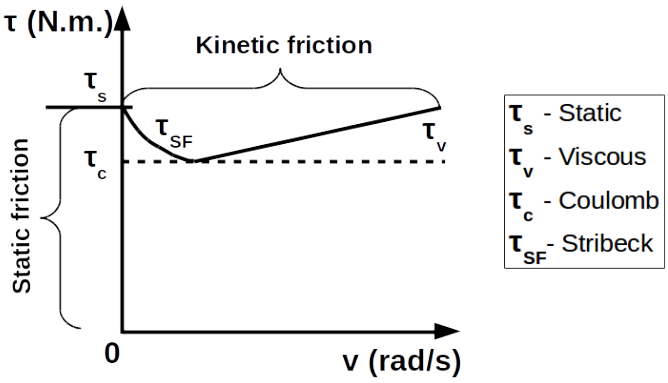


Figure 2.1: Friction model

The next important module of consideration in this work is the process control. PID controller is used by many of the standard industrial manipulators where each joint is attached with an independent PID. Mostly, single axis PID control scheme is used in the control applications [15] and the controller gains can be tuned with different methods starting from empirical tuning (step-response based tuning) [30] to auto-tuning methods. The tracking accuracy of this method suffer in high-speed operations due the constraints in the mechanical systems. As discussed before, the manipulators are non-linear systems, it is important to account the complete dynamics of the system for the improved trajectory tracking [25]. There are different kinds of motion control methods available for the robot manipulators such as impedance control, computed-torque control which also has

computed-torque like control [7]. Impedance control considers the interaction between the robot and the environment by accounting the dynamic characteristics of the system [9] rather than just doing force regulation based on the positions. In [4], computed-torque control with the PD controller is put to work in the youBot manipulator along with dynamic model. The author has considered only the gravitational torques $\tau_g(q)$ in mathematical model and it does not account for torques due to friction. But this method needs accurate model parameters that can improve the overall result of the control scheme. It is important to consider the finding provided by the author [4] that the maximum applicable torque provided by the manufacturers are not sufficient to move the joints of the robot in some of the configurations of the manipulator. So, the author proposes a method that can compute the maximum applicable torque for the individual joints of the manipulator with the following equation (2.1).

$$\overbrace{\tau_{max}}^{\text{nx1}} = \overbrace{\mathbf{M}_{max}}^{\text{nxn}} \overbrace{\mathbf{K}_p}^{\text{nxn}} \overbrace{\mathbf{e}_{max}}^{\text{nx1}} \tag{2.1}$$

where, τ_{max} represents the $n \times 1$ matrix of torque that represents the maximum torque allowed in n joints of the manipulator and it is obtained by computing the product of the maximized manipulator inertia matrix, the proportional gain K_p of the controller and the maximum possible joint position error e_{max} and the error depends on the specifications of the particular joint. Input to the maximized manipulator inertia matrix is the joint positions where the manipulator joints are pushed to it's maximum in order to get the maximized inertia matrix.

Chung [7] discusses computed-torque control scheme with the following formulation

$$\tau = H(q)v + C(q, \dot{q})\dot{q} + \tau_g(q) \tag{2.2}$$

where computed torque-control consists of an inner non-linear compensation loop and the outer-loop with the control signal v. This *control law is then substituted in the* dynamic model of the manipulator where $\ddot{q}_d$ is replaced with the control input v. The substitution of this control law in the model solves a non-linear problem hence the linear control strategies can be applied. PD feedback control mechanism is used in the outer-loop control v.

$$v = \ddot{q}_d + CV \tag{2.3}$$

Computed-torque control described above is the proper control scheme and the author also provides a simple control scheme where the model and the controller works independent of each other (control input is not fed to model). Feed-forward and computed-torque control schemes discussed by [2] results with reduction in the tracking error with the complete dynamics of the system. The model based controller requires accurate control

on joint torque and it depends on the accuracy of joint position and velocity data. The computed torque-like control works by adding the variable compensation in the outer loop for accurate trajectory tracking [7] due to the uncertainties in the dynamic model. YouBot manipulator can be visualized in Simbody [22] where the dynamics simulations can be achieved based on the recursive Newton-Euler formulation. Torque sensor can be created with the measured forces of gravity, bias and body forces, and it completes formulation of the generalized forces. Then the forces observed in the Cartesian space can be converted to joint torques. This library can be very useful in simulating the behaviour of the manipulator for the given motion control applications before testing it in the real robot.

Chapter 3

APPROACH

This chapter discusses the detailed information about the approaches used in this work. At first, the basic approach depicted in Fig. 3.1 is an advanced model based controller which gives the complete overview about all the components used in this work.

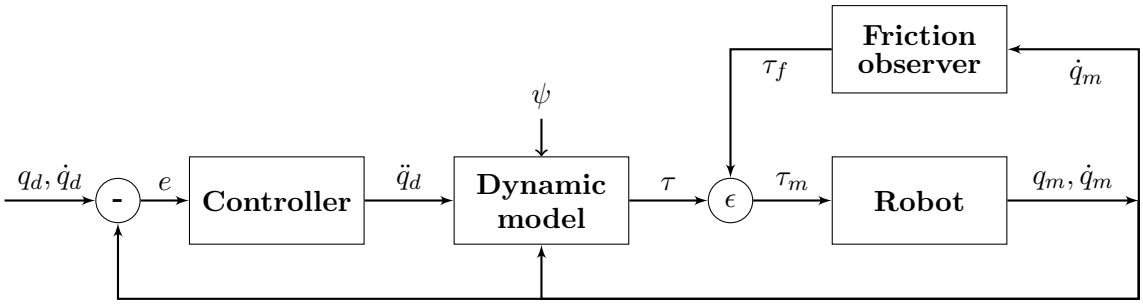


Figure 3.1: Basic approach

The control scheme given above 3.1 feeds the control variable as an acceleration $\ddot{q}_d$ input to the dynamic model for computing the model torques, whereas Chung et. al. [7] fed the actual desired acceleration $\ddot{q}_d$ as the model input in the defined control scheme. The reason for this difference in the controller scheme is that there is no accelerometer present in the manipulator joints. In such a case, it is only fair to answer the following question, why not differentiating the measured velocity to get the acceleration data?. The answer to the above question is that the measured velocity which is obtained from the encoders are generally noisy and taking the derivative of this data would amplify the noise much further, so the desired acceleration input is not considered in this work. An another important observation in this work is that the model considers only the link's moment of inertia and it can be subtracted from the motor's moment of inertia. The motor's moments of inertia can be computed with the specifications provided by the manufacturers. So, the alternate method is selected in this work for the model-based control as depicted in Fig. 3.2.

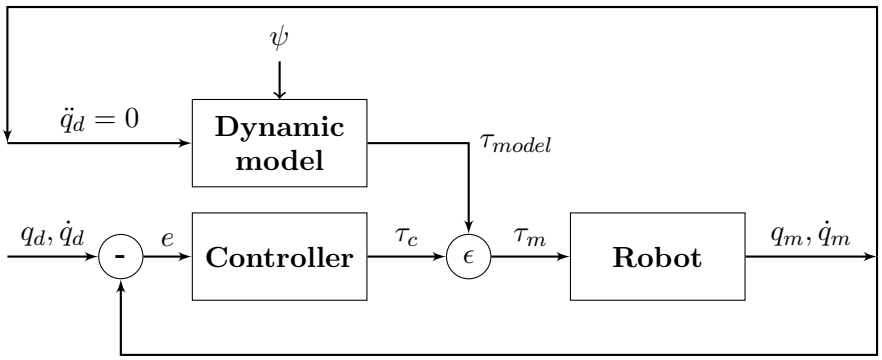


Figure 3.2: Alternative method - Computed torque-control which includes dynamic model

The following components are really important to reach the ultimate goal of this project that is specified in Fig. 3.2.

- **Robot/Process/Plant** represents the real system and it contains an actuator through which the process is controlled. The system that is used in this work is explained in detail in the preceding section. The disturbance in the system is the non-controlled variable that has an influence on the controller performance but the controller adjusts the control variable for compensating that influence.

- **Dynamic model** is required to generate the model torques based on Orocos KDL ID solver [23] and the manufacturer's 3D model [29] where ψ represents the dynamic model parameters which can be replaced with the identified parameters once the estimation is successful. The friction compensation term τ_f is also considered in this work with the basic approach.

- **Parameter estimation** that estimates the accurate model parameters. ψ represents the dynamic model parameters. The excitation trajectories can be identified using the Finite Fourier series [24] and the coefficients can be optimized for better identifying the model parameters. This section presents an overview about the parameter estimation and the trajectory parameterization and optimization.

- **Controller** solves the control problem where it reduces the error by introducing the regulation in the system with the linear feedback control method. This work uses the cascade control mechanism and velocity of the joint is controlled in velocity PI controller.

3.1 Description of the hardware platform

This work uses the KUKA youBot for the implementation of the identification, control modules it comprises omnidirectional mobile platform and the manipulator.

YouBot omnidirectional platform

The base of the robot has 4 omnidirectional wheels with the separate motors attached to each of the wheel joints. It is particularly useful in testing the controller before performing the test in the manipulator where the joint limits are applicable.

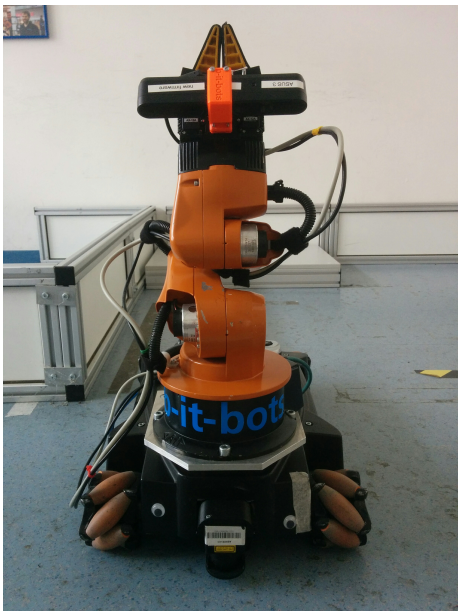


Figure 3.3: youBot platform and manipulator

YouBot manipulator

The main focus of this work is to achieve the computed-torque control in the youBot manipulator. The youBot arm consists of five rotary joints with 5-DoF controlled by the individual motors and it includes gripper in the last link which has two fingers. But this work does not consider the gripper both in modelling and the torque computation. The manipulator joints are commanded with the position or velocity or torque inputs through EtherCAT which has a real-time capability. The 3-D model of the manipulator is provided by the manufacturer [29] and it is used for modelling purposes and it is also used in computed-torque control where the model torques are computed. In order to increase the tracking performance, the gain values are tuned to a high value where there is a possibility of the overshoot problem. Due to this, the joint might cross the limits provided by the manufacturers hence it is important to introduce an artificial position limits to the joints for avoiding any damages to the system. The introduction of the limiting features

and the safety controller is explained in the proceeding sections. Since the joints are commanded in torque mode, it is necessary to verify the maximum applicable torque for the individual joints of the manipulator. By considering the specifications provided by the manufacturers, the maximum applicable joint torques were identified with the following equation (3.1)

$$\tau_{max} = \tau_{nominal} \cdot gear\_reduction\_ratio \tag{3.1}$$

where nominal torque is represented in N.m., and gearbox reduction ratio values are referred from youBot-store's detailed specifications[2]. The computed maximum joint torques are listed in the following table 3.1

3-D Model	**Joint1**	**Joint2**	**Joint 3**	**Joint 4**	**Joint5**
	In Newton meters (N.m.)				
Existing τ_{max}	9.500	9.500	6.000	2.000	1.000
New τ_{max}	12.901	12.901	8.270	4.175	1.755

Table 3.1: The maximum applicable joint torques provided by the manufacturers and it is computed from the equation (3.1)

The youBot-store specifications does not clearly state the nominal torque of the joint 5 hence the motor details are referred from [20] which has the specifications of 15Watt brushless Maxon EC32 flat motor with the Hal sensor number 267121[3].

[2] `http://www.youbot-store.com/wiki/index.php/YouBot_Detailed_Specifications`
[3] `https://www.maxonmotor.com/medias/sys_master/root/8825434800158/17-EN-262.pdf`

3.2 Description of the software platform

This section briefly discusses the software and libraries used in this work such as the youBot driver [12], Orocos KDL [23] and Simbody [22]. All the software used in this work is developed in Ubuntu 14.04 operating system.

YouBot driver

The youBot base and manipulator can be controlled via youBot-driver [12] and the cmake version of the driver is used in this work. The communication with the robotic joints can be established via EtherCAT master [12] which runs in a frequency of 1KHz (1 ms) hence a delay of 1 ms is introduced in computed-torque controller that is implemented in this work. The youBot-driver's architecture overview is given in Fig. 3.4.

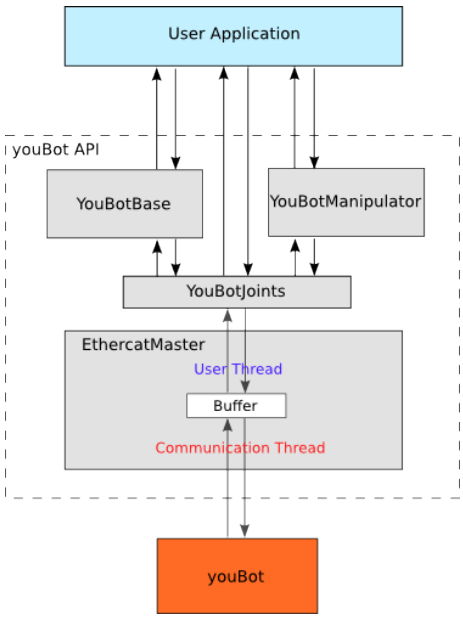


Figure 3.4: youBot driver architecture overview and the image is taken from [13]

Simbody

Simbody[4] [22] is the simulation engine that is used in this work to simulate and visualize youBot manipulator. Simbody follows recursive equations of motion similar to Featherstone [11] for the n joint DoF. It is a part of the simulation toolkit named SimTK which is an interface that gives access to the physics based simulation features. It is possible to perform the internal coordinate simulations of the multi-body systems by using Simbody API[5]. The multi-body system can be seen as the combination of rigid

[4] `https://simbody.github.io/simbody-3.6-doxygen/api/index.html`
[5] `https://simtk.org/docman/?group_id=47`

bodies that are connected through joints. Semantics of a rigid-body is given in Fig. 3.5. Simbody comprises two important classes which are related to the multi-body system such as system and state. The system class provides access to the logic for simulation which is constant and the state information stores the data overtime.

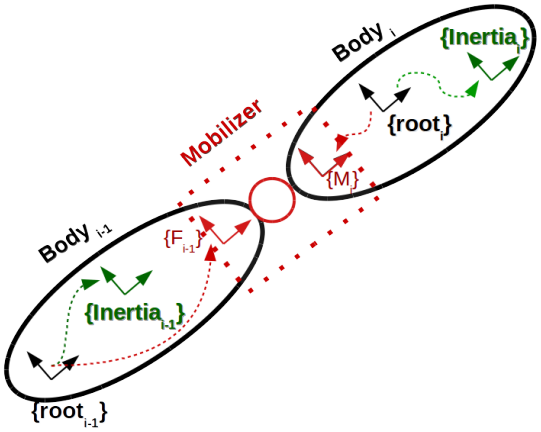


Figure 3.5: Rigid body semantics in SimTK

In Simbody, MobilizedBody represents the basic body and joint object which is added to the multibody tree. The MobilizedBody connects the child to the existing parent body which is assumed to be ground with the use of the mobilizer and also assigns the reference frame to the parent body. The mobilizer has two frames attached with it such as the fixed frame F is attached to the parent body which is the reference to account the motion of the moving frame M of the child body respectively. The Pin joint is used in this work as manipulator joints are revolute joints where the axis of rotation is possible around only one axis. The above figure describes the mobility of the child body ($body_i$) with respect to the reference $body_{i-1}$ with the use of the coordinate frames. Simbody considers that the parent ($body_{i-1}$) is attached to the ground and the $body_i$ is the child body that does all the motions. The root frames of the rigid bodies i-1 and i are represented as $root_{i-1}, root_i$ respectively. The transformation from the body i to body i-1 can be given as

$$^{root_{i-1}}T_{root_i} = ^{root_{i-1}}T_{F_{i-1}} \cdot ^{F_{i-1}}T_{M_i} \cdot ^{M_i}T_{root_i} \tag{3.2}$$

The mobilizer transformation from M_i to F_{i-1} mainly depends on the joint's coordinate $^{F_i}T_{M_i}$ which helps to locate the relative pose of the coordinate frame M with respect to the frame F. The other two segment transforms of both the parent and child body are composed with the mobilizer transform gives us the relative pose of the child body with respect to its parent.

Before getting into further details, the following changes are made in the construction

of the youBot manipulator in the visualizer. The pose of the CoM vector is represented w.r.t. the body's reference frame and the moments of inertia is in the principle axis. It is very clear from the Fig. 3.5 that the inertial frame is represented with respect to the body's reference frame. But the kinematic chain for the 3-D model has a convention that the body's reference frame is coinciding with the joint reference frame. Hence, the second transformation in the joint description is an identity. In order to adapt to the convention used by Simbody, the inertial frame has to be re-expressed and shifted. Simbody provide sufficient methods to re-express and shift the inertia and the implementation details are discussed in appendix C. Torque sensor is built in Simbody for getting the torques from the manipulator joints, it gives the way to get the external forces such as bias, gravity forces and the mapping of the Cartesian forces into joint torque by using inverse Jacobian. Since the same 3-D model is used in Orocos-KDL, the offset computation is carried out to match the input joint positions with respect to the model. The joint positions are then fed to the Simbody where the inverse of the KDL offset computation is fed to Simbody since the same model is used and it takes raw youBot configuration as an input. Note that the result of the Simbody joint position measurements are inverted back to youBot's raw data since the joint position feedback is given back to KDL based ID solver. The Simbody's convention is semantically mismatching with KDL based ID solver in the implementation. The mismatch is observed when applying model torques that are computed using KDL to the manipulator joints in Simbody for the purpose of gravity compensation task. The result of this task is that the gravity compensation terms in youBot manipulator joints are accounted but the joints are slowly drifting away from the candle configuration overtime in further iterations of the visualizer.

Orocos KDL

An important requirement for the model-based controller is to have a mathematical model that can predict the torque which is close-enough to the real system. A model comprises the physical properties of the robot such as kinematic, geometric and dynamic properties which are particularly useful in computing the model torques based on the given trajectory information. For this purpose, an open source library called Orocos KDL [23] is selected with an intend of modelling, computing the kinematic chain and predict torques using the recursive Newton-Euler ID (Inverse Dynamics) solver. The kinematic chain of the robot comprises segments of the manipulator itself and each segment has both the link and the joints attached between the links. The joint axis is negative for the joint number 5 hence the torque and velocity of the joint 5 has be inverted before commanding the model torques to the real robot.

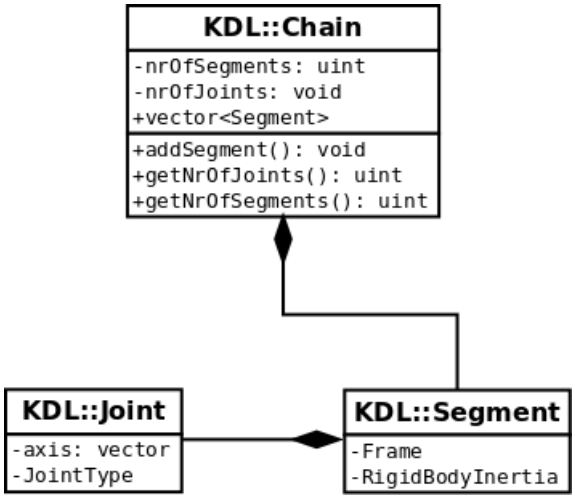


Figure 3.6: Kinematic chain in Orocos KDL which is a combination of both the joint and link

The kinematic chain (KDL::Chain) of the robot is the collection of segments (KDL::Segment $0 \cdots totalnumberofsegments$) where the segment is the combination of the link and joint (KDL::Joint) properties as depicted in Fig. 3.6. The implementation details are discussed briefly in appendix B.

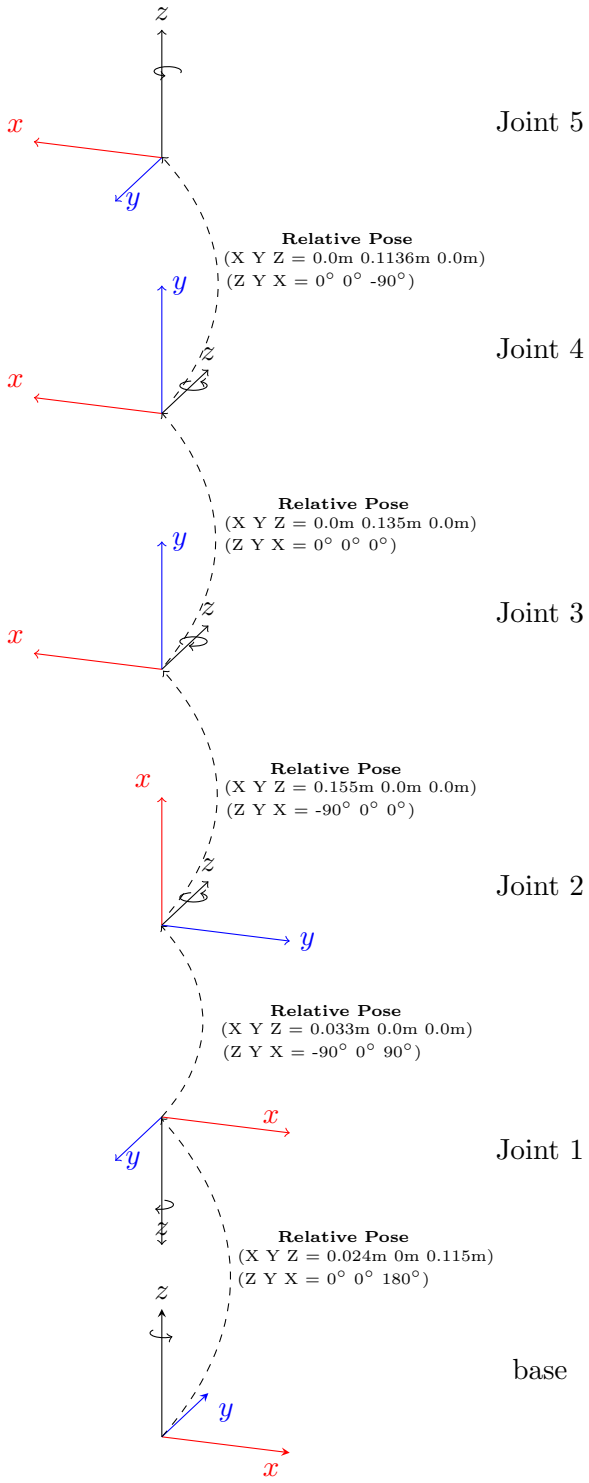
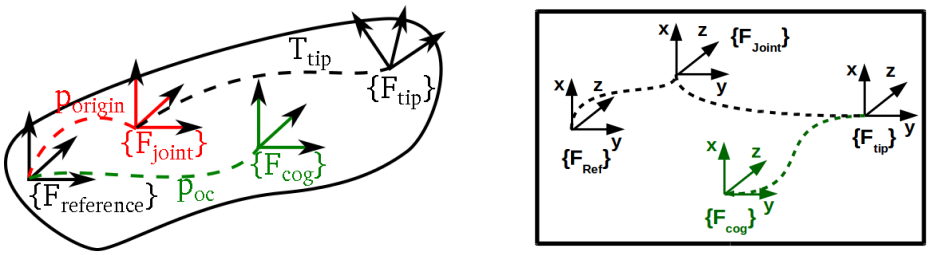


Figure 3.7: Frame representation of the youBot-store 3-D model based on [29] and image is taken from [19]

The raw joint angles of the youBot can't be used in kinematic and dynamic computations in KDL since the model has a different coordinate system representation comparing the real robot. So, it is important to compute the offsets for the model w.r.t. the youBot coordinate system i.e. by default, the youBot manipulator is present in the minimum configuration in the real system but 3-D model represents candle configuration as the minimum configuration hence offset calculation is needed to represent the joint information properly in KDL before inputting the joint positions to the forward position kinematics, torque computations.

Joint No.	**youBot raw values(in Radians)**				**youBot-store 3D model (in Radians)**			
	Min	**Max**	**Candle**	**Folded**	**Min**	**Max**	**Candle**	**Folded**
1	0.0	5.8992	2.9496	0.0	-2.9496	2.9496	0.0	-2.9496
2	0.0	2.7053	1.1345	0.0	-1.1345	1.1345	0.0	-1.1345
3	-5.1836	0.0	-2.5482	0.0	-2.6354	2.5482	0.0	2.5482
4	0.0	3.5779	1.7890	0.0	-1.7890	1.7890	0.0	-1.7890
5	0.0	5.8469	2.9234	0.0	-2.9234	2.9234	0.0	-2.9234

It is important to consider the correctness of the frame diagram provided by KDL library before jumping into the implementation of the same. This work has an important finding with KDL's frame diagram and it is explained in detail as follows



(a) KDL segment, image from kinematic-trees section of the kdl user-manual [23]

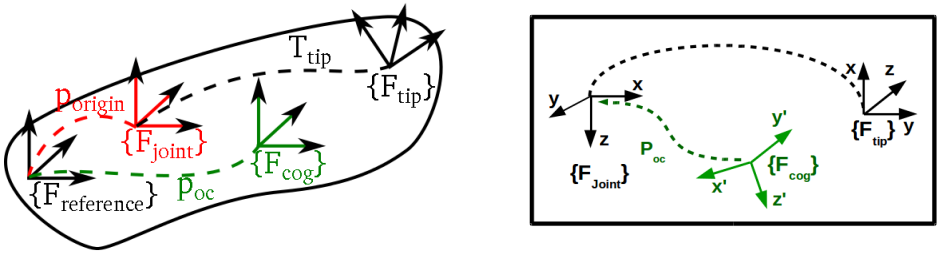
(b) Modified KDL segment

Figure 3.8: Correction in KDL frame diagram

The sample application is created to check whether the inertia is being felt in the reference frame or the tip frame of the segment. Consider an example where the tip of segment is translated by [0.5, 0.0, 0.0] (in meters) from the joint by 0.5 meters in x-axis and the CoM can be located by the vector [x = 0.5, y = 0.0, z = 0.0] (in meters) from the reference frame of the body. Where the result of this experiment is -9.81 Nm due to the lever arm of the tip frame and the CoM vector. According to this experiment, it is proven that the CoM vector is not located w.r.t. the reference frame of the body rather it is defined w.r.t. the tip frame. So, it is mandatory to represent the CoM vector

w.r.t. the reference frame as per the convention used by the youBot-store's 3D model. This transformation can be easily done by multiplying the inverse of the transform from ${}^{tip}X_{joint}$ (the relative pose from the reference frame to tip frame) with the CoM vector.

The convention used in KDL is different from the conventions in youbot-store model [29]. For example, the following Fig. 3.9(a) depicts the convention used by Orocos KDL library and the orientation of the inertial frame is same as the reference or joint frame. This means that inertial frame is just translated w.r.t. the segment's reference frame. On the other hand, youbot-store model uses the convention where the inertial frame is both translated by P_{oc} vector and oriented differently by Z-Y-X angle. This work considers that both the segment's reference and joint frames are the same.



(a) KDL segment, image from kinematic-trees section of the kdl user-manual [23]

(b) Youbot-store segment

Figure 3.9: Different conventions used by kdl and youBot-store

Based on the finding given above, if the developer wants to use the KDL library for kinematic and dynamic computations then the convention of the 3-D model has to be the same as KDL representation i.e, this work uses the youbot-store model to compute model torques hence the convention is changed according to KDL's convention as follows. The orientation of the inertial frame in the youBot-store model is not matching up with the convention used in KDL. So, the following equation (3.3) can be used to align the orientation of the inertial frame w.r.t. the orientation of the joint reference frame.

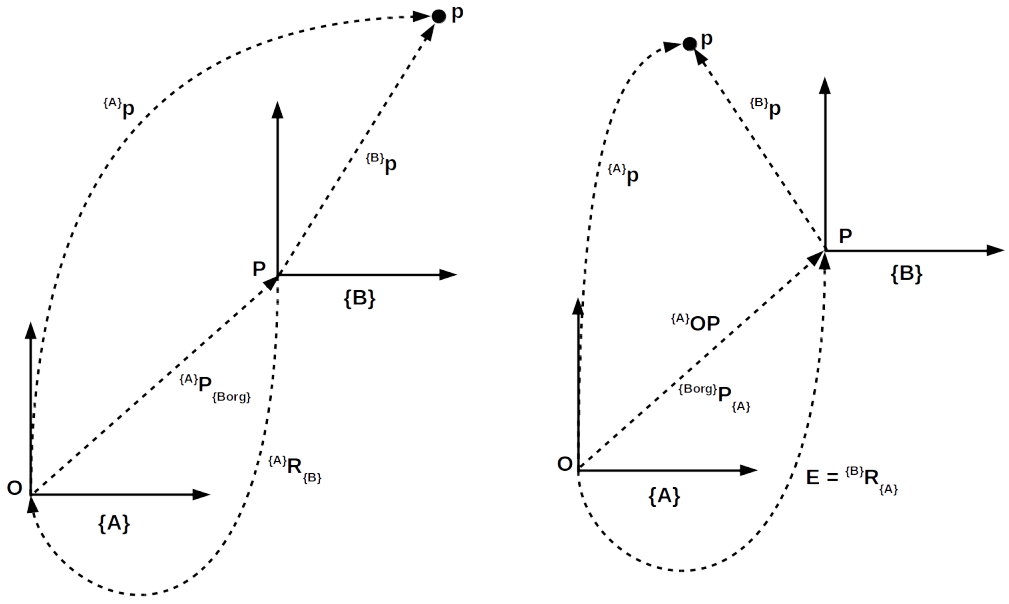
$$I_{\{Joint\}} = R^T \cdot I_{\{cog\}} \cdot R \tag{3.3}$$

where $I_{\{Joint\}}$, $I_{\{cog\}}$ represents the rigid body inertia with respect to the joint frame, center of gravity respectively and the rotation matrix that is used as the reference orientation R is taken from the joint frame.

3.3 Semantics for rigid-body algorithms

The definitions and the conventions of this section are written based on the reference from articles [1] [14]. An important characteristics of the manipulation involves the analysis of the motion of the rigid bodies in three dimensional space and the rigid body which

can't deform is considered in this work. Each and every rigid-body has 6-DoF (3-DoF each for the rotation and translation) in space where there are no constraints present. Basically, a constraint refers to the joint when it is attached to the rigid body, only 1-DoF exists. The geometrical relations between the rigid bodies such as position, orientation, pose, linear, angular velocity and twist, forces, torques and wrenches are important for the rigid-body algorithm. The pose, relative motion and the relative effort transmission between the rigid bodies must be represented in a very well defined coordinate system. If the developer fails to choose the coordinate system carefully it might lead to the logical errors that are difficult to trace. The coordinate system representations are mostly implicitly defined by the libraries such as Simbody [22], KDL [23] and the authors define their own representations which are semantically correct. So, it is important to decode the implicit conventions in order to avoid the logical errors in the geometrical relation semantics representation of the rigid bodies. For example, the following diagram shows different conventions used by two different authors hence the homogeneous transformation matrix that is useful in finding the relative pose between the two frames changes the coordinate calculations accordingly.



(a) JJ. Craig (b) Featherstone

Figure 3.10: Different conventions used by different authors [11] [8].

The inverse of the transform can be computed numerically with the following equa-

tions.

JJ Craig	**Featherstone**
${}^{A}P = {}^{A}_{B}R \cdot {}^{B}P + {}^{A}P_{Borg}$	${}^{B}P = {}^{B}_{A}R^{T} \cdot {}^{A}P^{-1}_{Borg} + {}^{A}P$
	${}^{B}P = {}^{B}_{A}R \cdot {}^{A}P - {}^{B}_{A}R \cdot {}^{A}P_{Borg}$
	${}^{B}P = {}^{B}_{A}R \cdot ({}^{A}P - {}^{A}P_{Borg})$
${}^{A}_{B}T = \begin{bmatrix} {}^{A}_{B}R & P_{Borg} \\ \mathbf{0} & 1 \end{bmatrix}$	${}^{B}_{A}T = \begin{bmatrix} {}^{B}_{A}R & -{}^{B}_{A}R \cdot {}^{A}P_{Borg} \\ \mathbf{0} & 1 \end{bmatrix}$

In Fig. 3.10(a), the pose of the frame {b} can be located w.r.t. the reference frame {A}. Whereas the Featherstone represents the pose of the frame {A} w.r.t. the reference {B} as depicted in Fig. 3.10(b).

3.3.1 Three-link manipulator semantics and overview

Semantics with poses

The above computations for the difference in convention between two methods 3.10 are numerically performed using the coordinate representations of the rigid bodies where the geometrical relation semantics checking is not happening. The numerical computations has led us into the logical errors in the previous work [19]. But this work makes a valid attempt to adapt to the standard use-cases defined by [14] which are relevant in computing the geometric relations between rigid bodies instead of considering only the coordinate calculations. The use cases are the standardized semantic checks that assists the developer in avoiding the logical mistakes in the geometrical relations construction and it is briefly discussed in [1]. The four use-cases are

- Coordinate calculations
- Semantics checking
- Coordinate semantics checking
- Finally the combination of the coordinate calculations and the coordinate semantics checking.

The following example as depicted in Fig. 3.11 is taken as a reference to explain all the use cases on a 3-link manipulator in a detailed manner.

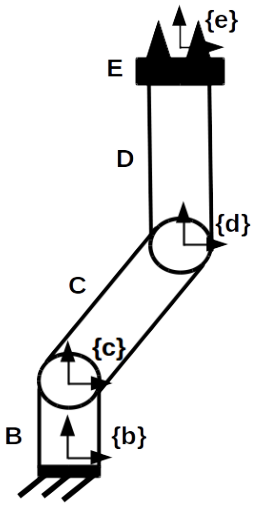


Figure 3.11: 3-link manipulator in order explain all the four use cases

The relative pose of the rigid body B w.r.t. the body A can be given as $Pose((c,c)|C,(b,b)|B)$ where (c, {c}) represents the position and orientation of the body C w.r.t. the reference body B. The first use case for the inverse and compose operations can be computed as follows

$$^{\{b\}|B}T_{\{c\}|C} = ^{\{c\}|C}T^{-1}_{\{b\}|B} \tag{3.4}$$

$$^{\{b\}|B}T_{\{d\}|D} = ^{\{b\}|B}T_{\{c\}|C} \,.^{\{c\}|C}T_{\{d\}|D} \tag{3.5}$$

$$^{\{b\}|B}T_{\{e\}|E} = ^{\{b\}|B}T_{\{d\}|D} \,.^{\{d\}|D}T_{\{e\}|E} \tag{3.6}$$

The second use case involves the semantics checking

$$Pose(\{c\}|C,\{b\}|B) = inverse(Pose(\{b\}|B,\{c\}|C)) \tag{3.7}$$

$$Pose(\{d\}|D,\{b\}|B) = compose(Pose(\{d\}|D,\{c\}|C), Pose(\{c\}|C,\{b\}|B)) \tag{3.8}$$

$$Pose(\{e\}|E,\{b\}|B) = compose(Pose(\{e\}|E,\{d\}|D), Pose(\{d\}|D,\{b\}|B)) \tag{3.9}$$

The third use case involves the coordinates semantic checking which also considers the coordinate frame check as given below

$$PoseCoord(\{c\}|C,\{b\}|B,[b]) = inverse2(PC(\{b\}|B,\{c\}|C,[c])) \tag{3.10}$$

$$PoseCoord(\{d\}|D,\{b\}|B,[b]) = compose(PC(\{d\}|D,\{c\}|C,[c]), PC(\{c\}|C,\{b\}|B,[b])) \tag{3.11}$$

$$PoseCoord(\{e\}|E,\{b\}|B,[b]) = compose(PC(\{e\}|E,\{d\}|D,[d]), PC(\{d\}|D,\{b\}|B,[b])) \tag{3.12}$$

where PC is the pose coordinates. The $PoseCoord(\{d\}|D, \{b\}|B, [b]$ is determined by composing the pose coordinates $PC(\{c\}|C, \{b\}|B, [b])$ and the pose coordinates $PC(\{d\}|D, \{c\}|C, [c])$ as given in the above equation. The reference point, reference orientation frame, and the reference body of the first pose in the compose operation in equation 3.11 have to be equal to the point, orientation frame, and the body of the second pose. The fourth use case checks the coordinate semantics of the poses that results from the inverse, compose operations can be verified from the arguments of the compose operation. At last the numerical operations based on the coordinate representations can be generated from the semantic equations of the third use case. If the software is built based on these semantic rules, it is possible to avoid the logical errors in the geometrical relation semantics between the rigid bodies. The possible logical error in the pose related semantics is that the poses and the orientation coordinate representations are composed in the wrong order. Since the coordinate system selection varies from one developer to another as depicted in Fig. 3.10 and the order of the composition indicates the geometric relation between the rigid bodies which is not commutative.

Motion and force related semantics

The motion transforms discuss about the semantics that needs to be considered to transmit the velocity or the acceleration of a particular rigid-body to its successor body. The motion transforms are considered as easy to implement but the real problem comes when different libraries are combined together for the purpose of achieving the common goal which in-turn introduces the different semantic representations used by the different authors i.e., Orocos KDL [23], Simbody [22], Atkeson et. al. [3], Featherstone [11]. It is only fair to explain the concept of kinematics before getting into further details. Kinematics is the study of motion of the objects without taking the forces into consideration that cause the motion. The kinematics can be classified into forward and inverse kinematics. This work uses only the forward kinematics which includes position, velocity and acceleration kinematics. The forward kinematics have undergone logical changes comparing the previous RnD [19] since the implicit conventions are analyzed and briefed in a detailed way. The twist or acceleration of a body has the point, reference and target body, coordinate frame information attached to it in the level of semantics.

3.3.2 Orocos KDL's geometrical relation semantics

It is really important to understand the semantics used by the library before jumping into the implementation details of it. If the developer fails to do so, the computations are going to be introduced with the logical errors in different levels such as the pose, motion and force level. So, lets start with the semantics considered by the Orocos KDL [23] library and then the changes required to keep up with the standard semantic conventions.

Semantics with poses

The position and orientation of the segment in KDL can be given as

$$\begin{aligned} p &: Position(tip_i, joint_i) \\ R &: Rotation([tip_i], [root_i]) \end{aligned} \tag{3.13}$$

where p represents the position of the tip_i w.r.t. the $joint_i$ position and R represents the relative orientation of the frame $[tip_i]$ w.r.t. the frame $[root_i]$.

Semantics with forward velocity kinematics

A twist is the combination of both the translational and angular/rotational velocities. The twist v_i of the $body_i$ can be computed by adding the twist of the predecessor body with the twist contribution of the $joint_i$ (3.14).

$$v_i = \overbrace{{}^iX_{i-1} \cdot v_{i-1}}^{transform} \overbrace{+}^{compose} \overbrace{S_i \cdot \dot{q}_i}^{V_J} \tag{3.14}$$

where the twist at joint i is projected into the joint's internal axis using the subspace matrix S_i, the motion transform transmits the twist contribution of the predecessor body v_{i-1} to the current body v_i. The addition operation of the twists in the equation (3.14) can be viewed in two different ways. At first, it can be seen that the numerical calculation adds the twists of two different bodies with the simple matrix operations. Another consideration is that the addition of the twists has to be semantically right. The compose operation adds the Twist($S_i \cdot \dot{q}_i$) with the Twist(${}^iX_{i-1} \cdot v_{i-1}$) to get the Twist v_i. Based on the coordinate semantic representation of the twist during the integration discussed by [1] [14], the twists can be integrated only when the twists are represented at a common point and at the same coordinate frame. If the semantics of the twists are different from these conventions then the developer has to consider the change point operation and change coordinate operation to form the proper geometrical relation between the rigid bodies.

The twist of the $body_i$ w.r.t. the $Body_{i-1}$ at point $joint_i$ can be given as

$$v_J = Twist(joint_i | Body_i, Body_{i-1}, [root_i]) \tag{3.15}$$

where v_J represents the semantics representation of the twist at joint i. The twist of the previous body can be given semantically

$$v_{i-1} = Twist(joint_{i-1} | Body_{i-1}, Body_w, [root_{i-1}]) \tag{3.16}$$

where $Body_w$ is the predecessor of the $Body_{i-1}$ or it can be considered as the ground body. The forward velocity kinematics are supposed to integrate the twists of the current body V_J as given in the equation (3.15) and the twist of the predecessor v_{i-1}. Where the

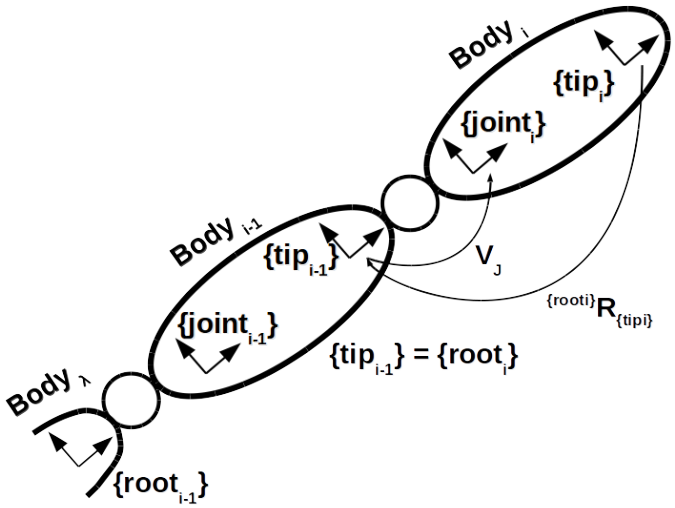


Figure 3.12: Velocity motion transform semantics used in Orocos KDL

semantics checking fails due to the reason that both twists are represented in the different coordinates and in different point. So, the change point and the change of coordinates are really important for the compose operation and it is computed in the following equations. The change point operation can be semantically given as

$$\begin{aligned} v'_{i-1} = Twist(joint_{i-1}|Body_{i-1}, Body_w, [root_{i-1}]) \\ .changePoint(joint_i|Body_{i-1}, joint_{i-1}|Body_{i-1}, [root_{i-1}]) \end{aligned} \tag{3.17}$$

The change point operation in the predecessors' velocity result in the following semantics

$$v_{cp} = Twist(joint_i|Body_{i-1}, Body_w[root_{i-1}]) \tag{3.18}$$

where v_{cp} represents the change point operation on the twist and the change coordinate operation on the same can be semantically extracted as

$$\begin{aligned} v_{cc} = TwistCoordinates(joint_i|Body_{i-1}, Body_w, [root_i]) \\ .changeCoordinateFrame(OrientationCoordinates([root_{i-1}]|Body_w, \\ [root_i]|Body_w, [root_i])) \end{aligned} \tag{3.19}$$

The change coordinate frame v_{cc} results with the following semantics

$$v_{cc} = Twist(joint_i|Body_{i-1}, Body_w, [root_i]) \tag{3.20}$$

Now, the twists in the equations (3.20) and (3.15) are semantically correct for the compose operation.

$$v_i = compose(v_J, v_{cc}) \tag{3.21}$$

and the geometric relation of the twist v_i results in the following equation

$$v_i = Twist(joint_i|Body_i, Body_w, [root_i]) \tag{3.22}$$

Semantics with forward acceleration kinematics

Similar to the velocity motion transform, the acceleration semantics can be extracted for the KDL library. The acceleration kinematics can be computed by

$$a_i = \overbrace{{}^{i}X_{i-1} \cdot a_{i-1}}^{transform} \overbrace{+}^{compose} \overbrace{S_i \cdot \ddot{q}_i}^{a_J} \overbrace{+}^{superpose} \overbrace{v_i \times S_i \cdot \dot{q}_i}^{Bias} \tag{3.23}$$

where the first compose operation of the acceleration at the joint i with the predecessor body i-1 can be done as same the velocity kinematics which involves motion transmission. The second addition operator is referred as superpose operation since the physical meaning of the addends are different. The above equation 3.23 can be simply viewed as the following

$$a_i = a_{i-1} + a_J + bias \tag{3.24}$$

The acceleration at joint i has the following relation

$$a_J = Acceleration(joint_i|Body_i, Body_{i-1}, [root_i]) \tag{3.25}$$

The acceleration of the body i-1 can be represented with the geometric relations as

$$a_{i-1} = Acceleration(joint_{i-1}|Body_{i-1}, [root_{i-1}]) \tag{3.26}$$

The change point and coordinates of the predecessor body's acceleration. The change point operation can be computed as

$$\begin{aligned} a_{cp} = {} & Acceleration(joint_{i-1}|Body_{i-1}, Body_w, [root_{i-1}]) \\ & .changePoint(joint_i|Body_{i-1}, joint_{i-1}|Body_{i-1}, [root_{i-1}]) \end{aligned} \tag{3.27}$$

which results in the following equation

$$a_{cp} = Acceleration(joint_i, Body_{i-1}, Body_w, [root_{i-1}]) \tag{3.28}$$

The change coordinate frame can be done with the following equation

$$a_{cc} = AccelerationCoordinates(joint_i|Body_{i-1}, Body_w, [root_i]) \\ .changeCoordinateFrame(OrientationCoordinates([root_{i-1}]|Body_w, \\ [root_i]|Body_w, [root_i])) \tag{3.29}$$

which results in the following equation

$$a_{cc} = Acceleration(joint_i, Body_{i-1}, Body_w, [root_i]) \tag{3.30}$$

Now the semantics checking is succeeded and the superpose operation is performed (a_{cc}+a'_i) based on [27] in equation 3.24 results in the following geometric relation semantics

$$a_i = Acceleration(joint_i|Body_i, Body_w, [root_i]) \tag{3.31}$$

Semantics with forces and torques

The recursive Newton-Euler formulation [23] in the spatial form which is used by the KDL library can be given as

$$f_I = I_i \cdot a_i + v_i \cdot \times (I_i \cdot v_i) - f_{ext_i} \tag{3.32}$$

where the acceleration a_i is mapped with the function I which gives the forces required, I represents the inertia of the rigid-body i. The semantics of the inertia has the information such as frame, and the body. The reference frame of the inertia can be assumed to be the frame that is attached in the target body as inertia represents the particular body. So the semantics of the operation $I_i \cdot a_i$ is same as the semantics of a_i given in the equation (3.31). The other forces such as bias and external forces should have the same semantics as the first term for the integration operation. The torques can be extracted by projecting the spatial forces into the joint's internal axis as

$$\tau_i = S_i \cdot f_i \tag{3.33}$$

where S represents the subspace matrix that projects the force into internal axis of the joint. It is assumed that the torque is represented in the target joint frame as the joint has a reference as well as the target frame. The reference frame is present in the predecessor and the target frame is the one which is attached in the target body that is being referred to. Formally, the force transform does the change point and coordinate frame operations as same as the semantics derived in the velocity and acceleration kinematics and once the semantics with the transform matches up with the semantics of f_i, the compose operation can be performed.

3.3.3 Featherstone's geometrical relation semantics

Before getting into the details of the pose, motion and force semantics, it is mandatory to go through the conventions used for the rigid body.

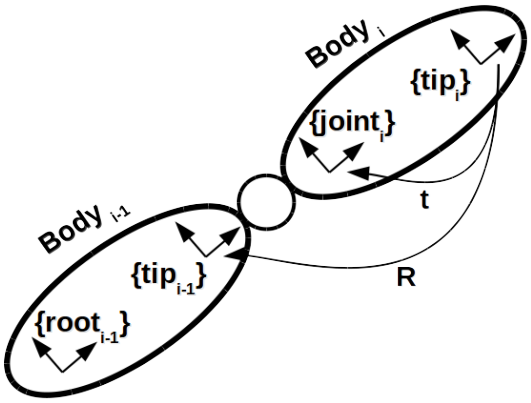


Figure 3.13: Rigid body semantics in the article [11]

Semantics with poses

The position and orientation of the segment is the same as the KDL segment

$$\begin{aligned} p &: Position(tip_i, joint_i) \\ R &: Rotation([tip_i], [root_i]) \end{aligned} \tag{3.34}$$

Semantics with forward velocity kinematics

In [11], author considers the point at which the twist is represented is same as the origin of $root_i$ hence there is absolutely no need of either the change point and orientation coordinates. The change point and orientation coordinates are accounted in the form of spatial transforms. This method is the inverse of the semantic representations used by KDL.

$$v_J = Twist(root_i | Body_i, Body_{i-1}, [root_i]) \tag{3.35}$$

Semantics with forward acceleration kinematics

The acceleration semantics can be given as

$$a_J = Twist(root_i | Body_i, Body_{i-1}, [root_i]) \tag{3.36}$$

Semantics with forces and torques

The semantics for forces, torques and force transforms in this rigid body algorithm can be extracted as same as the KDL. The only difference is that the propagation of forces between the rigid bodies does not require the change point and coordinate frame operation.

3.4 Inertial parameter estimation

The random trajectories were used for the identification procedure in the previous work [19]. These trajectories are not periodic and it is not possible to make sure whether the whole robot workspace is covered or not. With excitation trajectory analysis concentrating on even the smallest changes of the dynamic model parameters, we can better estimate the parameters.

3.4.1 Excitation trajectories

This section discusses the method that produces optimal robot excitation trajectories that are suitable for identifying the dynamic model parameters with trajectories covering the most of the workspace [24] [28]. The parameterization and optimization of the robot excitation trajectories can be generated with the following method.

Parameterization of the excitation trajectories

This section is based on an understanding from [24] [25]. The aim of the trajectory parameterization method is to generate the periodic and bandlimited data that improves the accuracy in parameter estimation. The parameterization can be done by using the finite Fourier series which is the sum of harmonic sine and cosine functions. This function generates the data for all the manipulator joints N and it can be generated using the following equation

$$q_i(t) = \sum_{j=1}^{N_i} \frac{a_j^i}{w_f \cdot j} sin(w_f \cdot j \cdot t) - \frac{b_j^i}{w_f \cdot j} cos(w_f \cdot j \cdot t) + q_i(0) \tag{3.37}$$

where w_f represents the frequency which is common for all the joints, t represents time, $a_j^i b_j^i$ represents the coefficients or amplitude of the functions sine and cosine, $q_i(0)$ is the initial position of the joint. The Fourier series has the periodic function with the period $T_f = 2\pi/w_f$ and it has 2N + 1 parameters that represent the DoF for the trajectory optimization. The basic idea of the parameterization of the trajectory involves selecting a low fundamental frequency which offers the longer excitation period. The velocities and accelerations can be generated by taking the derivative and double derivative of the equation (3.37).

Optimization of the excitation trajectories

The optimal values of the coefficients can be selected through experiments based on the trial-and-error method or by solving the non linear optimization problem with the constraints imposed on the robot motion [25]. The d-optimality criterion is used as an objective function that depends on the robot trajectory data that has been applied to the observation matrix K and the resulting matrix is stacked up vertically with the feasible solutions. The general optimization problem [6] finds the best possible solution from the available solutions. The main objective of the optimization problem is to minimize the objective function f(x) with the input vector x is subject to the equality and inequality constraints. In this work, x has parameters a, b, q(0) to optimize/minimize and the f(x) can be optimized with two different methods as given below. The determinant of the matrix gives sufficient details about the scores of the optimization.

$$f(x) : min(a_j^i, b_j^i, q_i(0))det(K^T \cdot K) \tag{3.38}$$

If the $det(K^T \cdot K)$ is equal to 0, it means that the matrix $K^T K$ is singular and the rows and columns of the matrix $K^T K$ are linearly not independent. Whereas the other method is based on finding the condition number

$$f(x) : min(a_j^i, b_j^i, q_i(0))cond(K^T \cdot K) \tag{3.39}$$

The above given equation is constructed based on the approach given by [28][25]. The large condition number means that the small change in the observation matrix can lead to a large change in output hence the highest scores are considered to be inefficient. The equality constraints of this optimization problem can be given as

$$\begin{aligned}
&h^1(x) : lower_joint_limit <= q(a, b, q_0) <= upper_joint_limit \\
&h^2(x) : lower_joint_limit <= \dot{q}(a, b, q_0) <= upper_joint_limit \\
&h^3(x) : lower_joint_limit <= \ddot{q}(a, b, q_0) <= upper_joint_limit \\
&h^4(x) : lower <= FPK(q)(nocollision)
\end{aligned}$$

The K matrix is constructed by vertically stacking up the result of the way-points of the optimized trajectory that is applied to the K matrix.

$$K = \begin{vmatrix} K(q(1), \dot{q}(1), \ddot{q}(1)) \\ . \\ . \\ . \\ K(q(N), \dot{q}(N), \ddot{q}(N)) \end{vmatrix} \tag{3.40}$$

3.5 Model-based controller for a non-linear system

This section explains the model-based controller for the non-linear system which is considered in this work. As discussed in the previous sections, the joint-space controller is implemented in this work and the pictorial representation is shown in Fig. 3.14.

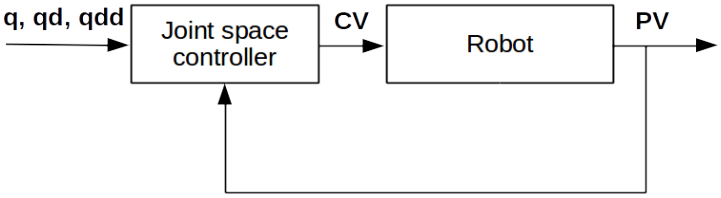


Figure 3.14: Joint space controller

This work uses the advanced joint control mode that commands torque to the manipulator joints in order to achieve the particular target. Since the controller is attached in the joint space of the youBot manipulator and it is not defined in the operational task space. The accuracy of this control method heavily relies on the accuracy of the dynamic model parameters and the encoder that is attached to the individual joints of the manipulator. The following Fig. 3.15 gives overview of operations before commanding the torque to the real system.

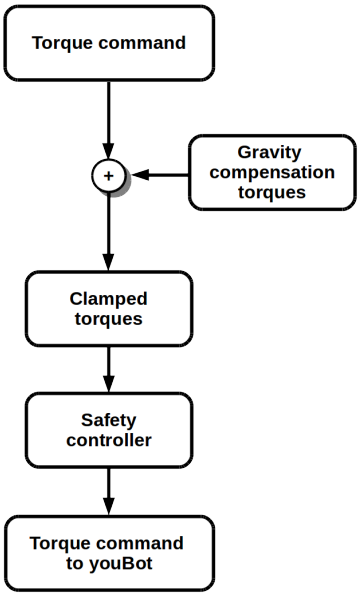


Figure 3.15: Torque control mode

In addition to the commanded torque, the gravity compensation torque is also added as a feed-forward mechanism. The result of this addition goes through clamping and this feature is particularly important in handling the out-of-bounds or NaN torque commands. The class diagram for the package implemented in this work is depicted in Fig. 3.16.

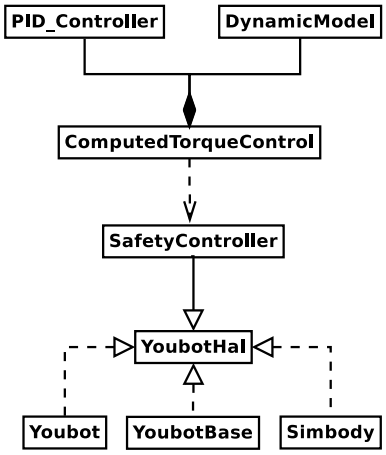


Figure 3.16: The complete class diagram for the model-based controller which is implemented in this work.

3.5.1 Dynamic model based on Orocos KDL

There are two kinds of controllers as depicted in Fig. 3.1 and 3.2. The approach specified in Fig. 3.1 is implemented on the youBot manipulator where the problem persists with the inaccurate rotational inertia parameters, hence the alternate method is chosen as depicted in Fig. 3.2. An important feature of consideration in computed-torque control [4] is the dynamics of the manipulator whereas the joint position or velocity control do not account the dynamics and the joint configuration. Dynamics is the field of study where the relationship between the forces and the motion are investigated. The general equation of motion can be written as based on [4]

$$\tau = M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q,\dot{q}) \tag{3.41}$$

where τ is the torque applied to the manipulator links. M represents the manipulator inertia matrix which comprises of mass, moments of inertia and C represents the Coriolis and Centrifugal forces which basically depends on the joint position and velocity. G represents the vector of external forces acting on the manipulator links such as gravity and friction effect. The non-linear dynamic equation of motion for the manipulator can be used to compute the joint torques for the given joint position, velocity and acceleration. The dynamics equation (3.41) has non-linear functions M, C and G of the model parameters. In order to control the non-linear system with the linear control, it is important to select the control input τ that follows the desired trajectory. The above equation (3.41) can be rewritten as

$$\ddot{q} = M^{-1}(q)(\tau - C(q,\dot{q})\dot{q} - G(q,\dot{q})) \tag{3.42}$$

if the non-linear terms in the above equation (3.42) are cancelled, it is possible to

apply the linear control strategies. If the non-linear terms are known, it is possible to choose the control input as given in the equation (3.43)for the PD control loop

$$\tau = M(\theta)\ddot{q} + C(\theta, \dot{\theta})\dot{\theta} + G(\theta, \dot{\theta}) \tag{3.43}$$

The following equation can be regarded as the closed-loop error dynamics where the K_D term is equal to 0 and it is referred from the article [7]

$$\dot{q}_d = (q_e \cdot K_p + \sum q_e \cdot K_i) \tag{3.44}$$

$$\ddot{q} = limit(\dot{q}_e \cdot K_p + \sum \dot{q}_e \cdot K_i) \tag{3.45}$$

where K_p, K_i represents the proportional and derivative gains of the PI control respectively and the gains are going to be optimized by manually tuning for each and every link of the manipulator. The manual tuning can be done based on the experiments by investigating the individual step responses for the particular joints. Otherwise, overshoot occurs when the gains are not optimal and the stability of the system cant be achieved. $q_e, \dot{q}_e$ represents the position and the velocity error as given in the following equations

$$q_e = q_d - q_m \tag{3.46}$$

$$\dot{q}_e = \dot{q}_d - \dot{q}_m \tag{3.47}$$

where q_d represents the desired joint angle and q_m represents the measured joint angle of an individual joint. $\dot{q}_d$ represents the desired joint velocity and $\dot{q}_m$ is the measured joint velocity of an individual joint.

Alternate method is chosen in this work which is a simple computed-torque control scheme. The dynamic model gets the measured joint positions and velocities from the manipulator joints with the zero acceleration. The $\ddot{q}$ is zero as depicted in Fig. 3.2, so the equation (3.41) can be rewritten as

$$\tau = C(q_m, \dot{q_m}) \cdot \dot{q_m} + \tau_g(q_m) \tag{3.48}$$

The above equation can't accelerate the joints but it computes the gravitational term and the Coriolis, Centrifugal terms.

Gravity compensation

The gravity compensation torques $\tau_g(q)$ are computed based on the KDL model which has been specified in the previous section. The model based gravity compensation on the manipulator joints and the gravity compensation for joint 1 and 5 can not be computed since these two joints are fixed.

Friction modelling and compensation

Friction is one of the important effects that need to be accounted in the control systems. The control systems try to compensate the friction effects by tuning the controller gains to a higher value, so there is a need of a friction model and the compensation technique. Friction has a significant effect on the mechanical systems where the precision is the ultimate goal and there are many friction modelling schemes have been introduced [20]. It can be classified into static and kinetic(dynamic) friction.

Algorithm 1: Direction of the desired velocity

```
1  directionOfDesiredVelocity (q̇_d);
   Input  : q̇_d
   Output: scale
2  if (q̇_d < 0 then
3  |  scale = -1; return scale;
4  else if q̇_d > 0 then
5  |  scale = 1: return scale;
6  else
7  |  scale = 0; return scale;
```

Algorithm 2: Friction observer

```
1  frictionModel (q̇_d, q̇_m);
   Input  : m, q̇_m,,
            n ← 5,
            static_friction,
            kinetic_friction
   Output: τ_f
2  if |q̇_d| > 0.0 then
3  |  for joint_number ← 0 to n by 1 do
4  |  |  if |q̇_m| <= 0.01 then /* The joint is not moving          */
5  |  |  |  τ_s = static_friction[joint_number] · sgn(q̇_d[joint_number]);
6  |  |  |  τ_f = τ_s;
   |  |  /* Coulomb friction modelling                             */
7  |  |  τ_k = τ_c · sgn(q̇_d[joint_number]);
   |  |  /* Kinetic friction torque                               */
8  |  |  τ_f = τ_k;
9  |  end
10 return τ_f;
```

The friction values are identified based on trial-and-error method and the static friction estimates are readily available [19]. The Coulomb friction is used to model the kinetic friction where the torque is constant irrespective of the input velocity value as given in the Fig. 3.17. The Coulomb friction basically restricts the rotational motion of

the joint. The static friction torque is applied to a particular joint makes the joint to overcome the static friction and it starts to move. This particular technique is referred to static friction compensation. Once the joint is in motion, the kinetic or dynamic friction torque is applied as the compensation term. This technique is referred to kinetic friction compensation.

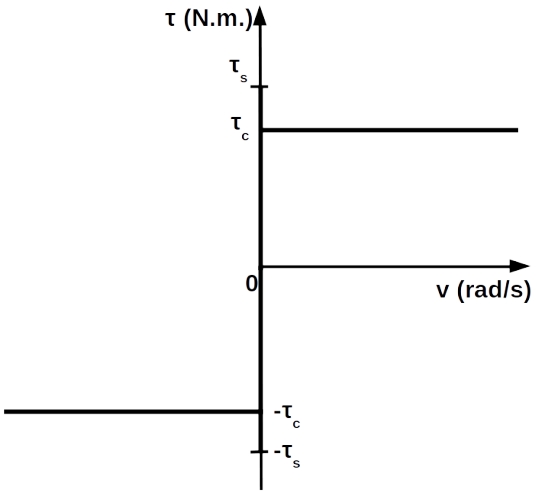


Figure 3.17: Friction model

3.5.2 Computed-torque control

Computed-torque control technique is widely used in the industrial robots and it works with the feedback mechanism to control the non-linear systems. The class diagram for the computed-torque controller is shown in Fig. 3.18. It is possible to apply the linear control strategies i.e., PI controller after the system is linearized. The linearization is achieved with the help of the KDL based ID solver which is the recursive Newton-Euler formulation. The working principle of the cascade controller is explained in the following section.

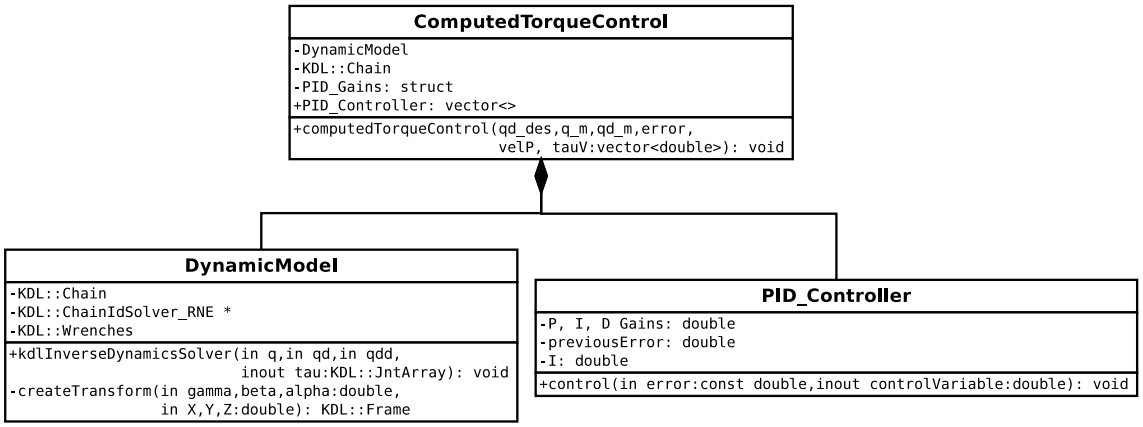


Figure 3.18: The class diagram for the computed-torque control.

Cascade PI controller

Most of the control applications use only the single-loop control for solving a control problem. The basic definitions of this section are referred from [15] [30]. But this work uses the cascade control method that has two controllers where one controller's output drives the set-point of the secondary controller. The cascade control system can have one or more inner loops inside the main control loop (outer-loop), and the controllers are in cascade. The advantages of the cascade controller is the fast response and the better disturbance rejection. This work is not considering the D-term of the controller as it is assumed that the damping factor is already present in the system as the friction force. The main loop can be called as master loop and the inner loop is the secondary controller. The position PI controller is attached in the outer loop and the velocity PI controller is attached in the inner-loop. The position PI controller output is limited to a desired velocity before feeding it into the velocity controller. Based on this limiting feature, the velocity of the joint can be controlled and saturated when the limits are breached. It is particularly useful in avoiding the high-speed motions in joint level. The presence of the dynamic model makes this controller model-driven whereas the system is considered to be pure cascade PI control scheme if the dynamic model is not considered.

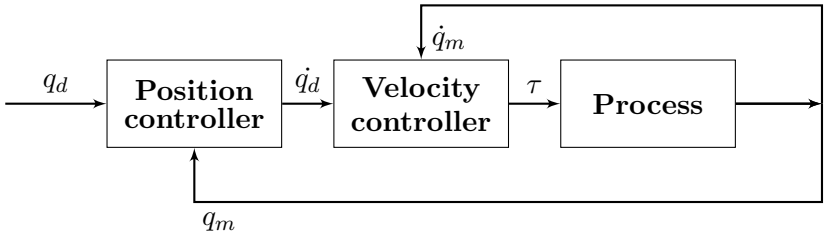


Figure 3.19: Cascade PI controller

Position PI control Position PI is the most commonly used control scheme which is used to perform the closed-loop control based on the position feedback. The joint position measurements are obtained from the encoder. This kind of a control is very useful and easy to use in tracking based applications. The control system is given with the set-point(SP) and the process/system executes the given command where the feedback will be given back to the control scheme where the regulation happens. The error q_e is computed by finding the difference between the set-point and the measured positions is $q_d - q_m$. This error is given as an input to the PI controller where the p, i terms are getting computed.

$$\dot{q}_d = (q_e \cdot K_p + \sum q_e \cdot K_i) \tag{3.49}$$

where the output of the position PI controller is commanded as the velocity set-point to the inner-loop as $\dot{q}_d$.

Velocity PI control In Velocity PI, a velocity set-point can be commanded to the system. The velocity set-point is is commanded by the position PI controller. This cascade mechanism is really useful in controlling the velocity set-points. The error is then calculated by subtracting the commanded set-point with the measured velocity. The error term $\dot{q}_e$ is computed by finding the difference between the velocity set-point and the process variable $\dot{q}_d - \dot{q}_m$. The error term is then fed to the PI controller for computing the P, I terms.

$$\tau_{controller} = limit(\dot{q}_e \cdot K_p + \sum \dot{q}_e \cdot K_i) \tag{3.50}$$

Eventually the controller produces the torque $\tau_{controller}$ that regulates the system by reducing the error terms very close to zero in the manipulator joints. The gravity compensation torques are then added to the controller torques as given as follows

$$\tau = \tau_{controller} + \tau_g(q) \tag{3.51}$$

Manual tuning of the controller gains

The controller gains are adjusted to optimum values to achieve the desired and stable response. The proportional K_p and integral K_i gains are tuned manually by adjusting the gain based on the step response of each joint. It is obvious that the increase in controller gain degrades the stability of the system since the gain has an influence on the rise time. Increasing the proportional gain K_p results in the faster rising time and it might cause the overshooting which affects the stability of the system. So, the gradual increase in the proportional gain till the better step response is achieved. The joint level PID controller gains are retrieved from all the individual joints of the youBot manipulator through youBot driver and the gains are observed to be relatively close to the state-of-the-art method [4] and the observed values are given in the following table.

Table 3.2: Joint level PID gains of the youBot manipulator in the current mode of control

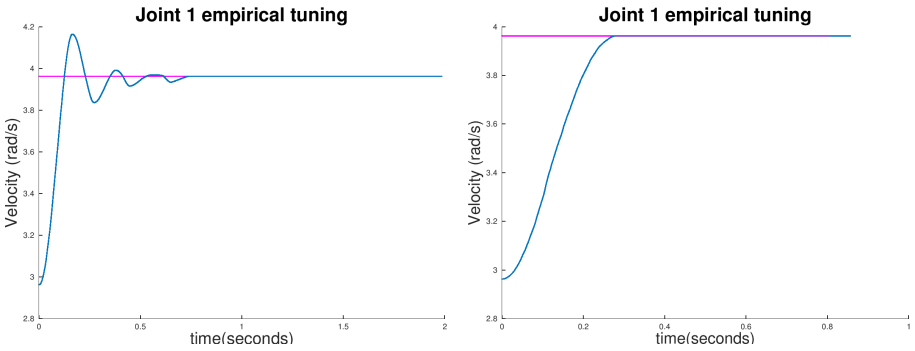
Joint No.		1	2	3	4	5
Current Mode	K_p	1200	1500	1500	3000	4000
	K_i	1000	1500	1500	3000	4000
	K_d	0	0	0	0	0

In this work, cascade PI controller is used where the velocity PI controller is present in the inner loop and the position PI controller is the outer loop control which is the main controller. It is necessary to tune the gains to achieve the optimal controller performance, so both the controllers have to be tuned empirically to get the optimal controller gains. An important trade-off must be made when empirically tuning the gains of the joint. The ultimate goal of the controller is to achieve the accurate trajectory tracking for which the short rise time (as depicted in Fig. 3.20(a)), faster settling time and the zero steady-state error are the mandatory requirements, but the short or faster rise time introduces oscillations, overshoot in the system which might cause damages to the motor, moreover the overshoot might breach the joint limits if the setpoints are close to the maximum or minimum limits of the joints. One of the major concerns of this work is the safety of the hardware irrespective of the accuracy that the developer wants to achieve in the controller. The other possible method is also depicted in the following Fig. 3.20(b).

So, the trade-off has been made which avoids the oscillations and the

Safety controller

An upper layer has been created around the controller to prevent the damages to the manipulator joints i.e. the torque is applied when the joint is already present in it's minimum or maximum positions. This controller enables the user to apply the control algorithms to the robot safely without thinking about the mishaps that can occur in the manipulation tasks. Basically, there are three kinds of checks are being performed such



(a) Fast rise time (b) Slow rise time

Figure 3.20: The requirements of the controller and this figure represents the tradeoff between the slow and faster rise time.

as position, velocity and torque. At first, the position control check is being performed to make sure that the torque is not commanded when the joint is in it's limits. Secondly, the velocity control checks the joint velocity for avoiding the high-speed motions on the manipulator that might hit the joint limits. So, this mode stops the controller when the velocity of the joint goes over the defined threshold. Finally, the torque control check is performed in order to keep the torque input in check before commanding it to the real robot. The safety controller allows the user to set the joint torque after all these three conditions are satisfied. If any of these three conditions fail, the velocity of the manipulator joints will be set to zero where the movement of the manipulator joints will be completely stopped irrespective of the manipulation task that is getting executed by the controller then the controller stops as well due to the safety concerns. This particular safety controller interface can be used where the joint limits are applicable.

3.5.3 Trajectory generation

Since the controller is attached in the joint-level, it is important to validate the controller with the use of a target trajectory. In this work, the trajectory way points are generated by using a simple sine wave-form where the amplitudes of the wave depends on the maximum and minimum limits of the individual joints.

$$q_{desired}(t) = A \cdot sin(\omega \cdot t) + \delta \tag{3.52}$$

where $q_{desired}$ represents the analytical time based function which has all the way-points to achieve the particular target. $2 * \pi / T_f$ is the frequency ω_f of the wave, A represents the amplitude, T_f is the time taken for a complete single wave and finally δ represents the offset, t represents the time taken by the controller to complete one cycle. The waypoints are fed to the controller and the measured joint values are compared against the wave to check the controller's performance. Based on the generated wave, the controller's parameters can be verified for optimality. If the joint is not following the wave close-enough, the controller gains have to be reconsidered since there are not optimum. The maximum and minimum velocity of the joints can be identified by taking the differentiation of the equation (3.52)

$$\dot{q}(t) = A \cdot \omega \cdot cos(\omega \cdot t) \tag{3.53}$$

The analytical sine waveform is generated for the joint positions ($q_{desired}$) where the amplitude of the wave depends on the joint limits of the youBot manipulator. The sine wave is generated in every 10 seconds (T_f) and the way-points of the wave is commanded as the set-point to the cascade PI controller. The derivative operation on the function q(t) provides the velocity set-point $\dot{q}(t)$ for all the individual points in the generated sine wave.

In order to find the maximum velocity A of the joint can be computed with the following equation. The ω is already specified where the maximum possible value for a cosine wave is 1.

$$\dot{q}_{max} = A \cdot \omega \tag{3.54}$$

similarly the minimum velocity of the joint can be identified with the following formulation where the amplitude can be computed by taking the difference between the minimum joint limit with the initial position of the joint.

$$\dot{q}_{min} = A \cdot \omega \tag{3.55}$$

The mechanical system's introducing the delayed start comparing the waveform

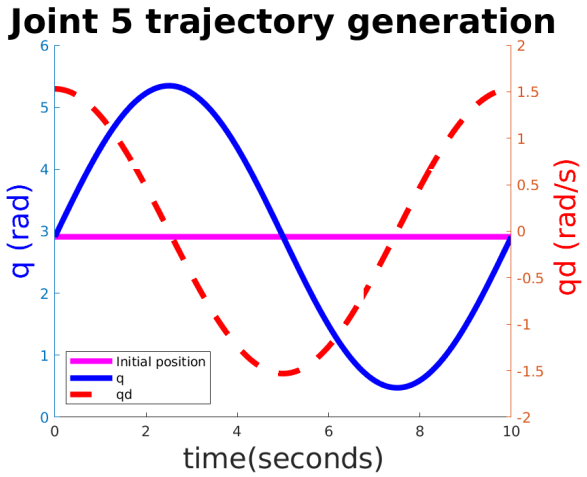


Figure 3.21: The sine waveform based trajectory generation and the relative velocity qd ($\dot{q}$) is obtained by taking the derivative of q.

generation. So, the derivative of q(t) is considered as the desired velocity and it is added with the position PI's control variable (which is also a velocity). The result of this addition is commanded as the velocity set-point for the velocity PI controller. This technique helps reducing the delayed start in the mechanical systems where the desired velocity at a particular set-point is caught-up. This method can be called as computed-torque like control [7] for the purpose of accurate trajectory tracking. The above generated wave corresponds to the joint positions and the derived velocities can be used to validate the model-based controller for all the manipulator joints.

Chapter 4

EXPERIMENTAL RESULTS

This chapter analyses the performances of the advanced model-based computed-torque controller. The experiments are conducted in the system that has the following configuration

Operating system - Ubuntu 14.04 LTS 64-bit
Processor - I5-7^{th} generation

4.1 Model-based controller

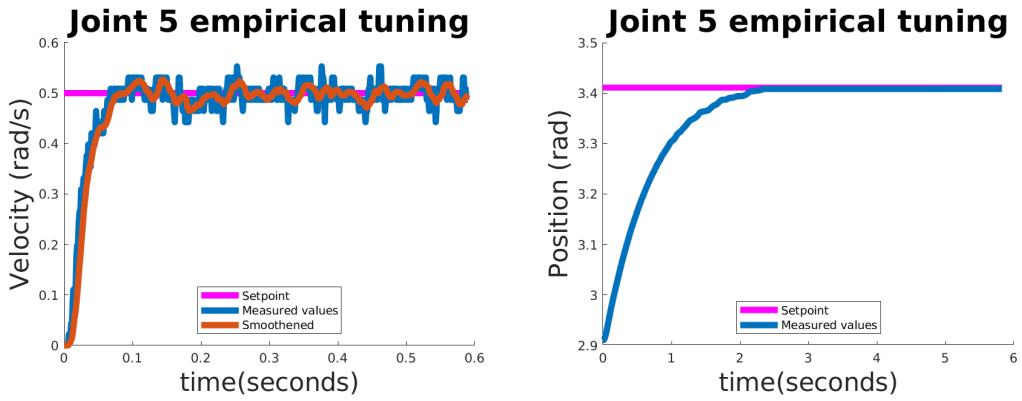
Torque controller is tested in youBot base at first to verify the logic of the controller before it is tested in the manipulator where the joint constraints are applicable. At first, the experiments are conducted for the safety controller interface to make sure that the safety layer is functioning properly or not in case of a logical error in the controller. The initial configuration of manipulator is assumed to be in candle as same as the 3-D model provided by the manufacturer, then the torque is commanded to the joint where the artificial position, velocity, torque limits are applied which is then replaced with the actual joint limits with a threshold of 0.5 rad in both the maximum and minimum limits of the manipulator joints. The threshold is introduced to avoid hitting the limits of the joints if the set-point is located near the joint limits. The artificial joint limits for the position, velocity, torque are 1.0 rad, 0.5 rad/s, 1.0 N.m. respectively. Once the joint limits are reached, the safety layer gets active and stops the controller to avoid any damages to the manipulator since the prolonged motion even after the breach in the safety limits could damage the system. The same experiments are conducted with the actual joint limits and the safety controller interface is breaking in the right place where the safety limits are breached. The real concern of the safety controller in high-speed motions is that the joint crosses the safety limit before the safety check is performed. The cause of this problem is that the controller is functioning in a non real-time system.

PI Controller	Gains	Joint No.	
		4	5
Position	P	0.820	1.900
	I	0.063	0.033
Velocity	P	2.900	1.500
	I	0.180	0.120

Table 4.1: Controller gains for the joints 4,5 after the empirical tuning procedure

Basic approach is experimented prior to the alternate approach. The model-based controller based on the basic approach is successfully achieving the gravity compensation in the youBot manipulator that is based on KDL ID solver and the cascaded PI controller is also tested in the youBot manipulator. The main observation from this experiment is that the control variable $\ddot{q}$ computed by the cascaded controller is fed as an acceleration input to the KDL based solver where the model torques are generated which results in a very minimal torque ($10^{-5}N.m.$). These model torques were not sufficient enough to move the joints of the manipulator to the desired configuration. The reason behind this problem is that the inertial parameters are not accurate and rotor inertia should also be considered rather than just considering the links' moments of inertia. The static friction compensation is also experimented to move the joint from the equilibrium state and the identified static friction values from the previous work [19] were not sufficient to move the joint from the rest state. So, the static friction value for the joints are increased to overcome the static friction on the manipulator joints. Once the joint started moving, the Coulomb friction torques were added with the model torques that compensates the dynamic friction. The alternate method that is used in this work is not having the friction compensation with it due to the fact that the model does only two things such as the gravity compensation and accounting the bias forces. Since the cascaded controller is implemented in this work, it is important to tune both the position and velocity controller gains to the optimum, empirically. The inner-loop and outer-loop controller has been tested with the different velocity set-points to check whether the tuned parameters are optimum or not. The empirical tuning of the controller gains for both the inner-loop and outer-loop with the velocity of 0.5 rad/s are depicted in the Figures 4.1(a) 4.1(b) respectively for the joint number 5. Since the measured velocity from the encoders are quite noisy as it is shown in Fig. 4.1(a), the measured data is smoothed with the moving average filter that filters the noise with the window size 10. The smoothed data is helpful in understanding whether the set-point in the inner-loop has reached or not. The steady-state error with the threshold of 0.02 rad/s are assumed to be accepted in the inner-loop when tuning the controller gains due to the presence of noise but the ultimate goal of this

project is to achieve the zero steady-state error. The outer-loop tuning is quite straightforward where the joint position set-point 0.5 rad is commanded to the joint and the result of the movement confirms the optimality of the controller gains. The additional tuning experiments with the different position and velocity set-points for the joint 5 can be found in the appendix D.



(a) Inner loop tuning

(b) Outerloop tuning

Figure 4.1: Inner, outer loop tuning for joint 5 with the velocity, position set-points of 0.5 rad/s, 0.5 rad respectively

After the controller gains are tuned to optimal values based on the set-points and the controller response, the individual joints of the manipulator can be evaluated with the analytical sine wave form as depicted in Fig. 4.2 since the computed-torque controller is attached in the joint level. The way-points of the sine waveform are generated in accordance with the controller frequency. In each and every cycle of the controller, a new position set-point (the way-points of the sine waveform) is commanded and the response is observed for the purpose of evaluation.

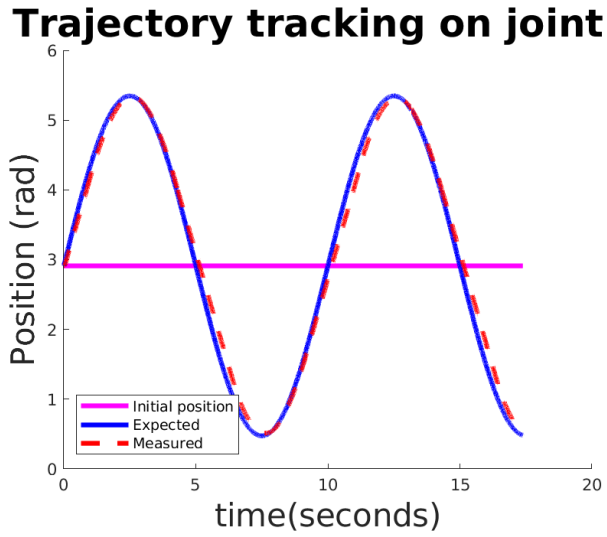


Figure 4.2: Trajectory tracking experiment using the sine wave on joint 5

The error between the set-point and the controller response for all the way-points of the sine wave are depicted in Fig. 4.3. The maximum error that has been observed in joint number 5 is 0.3171 rad at some point in time.

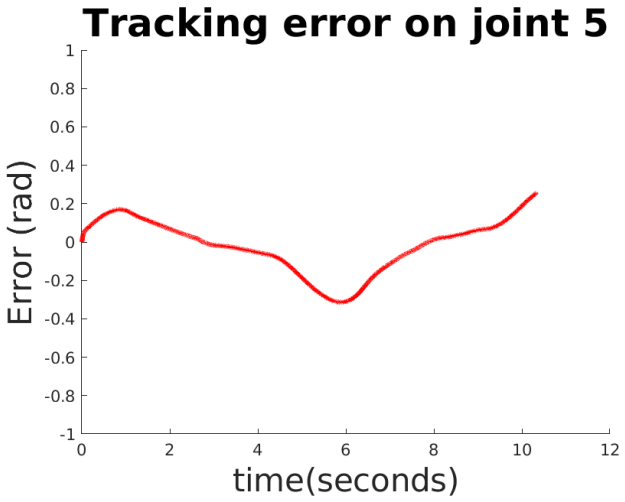
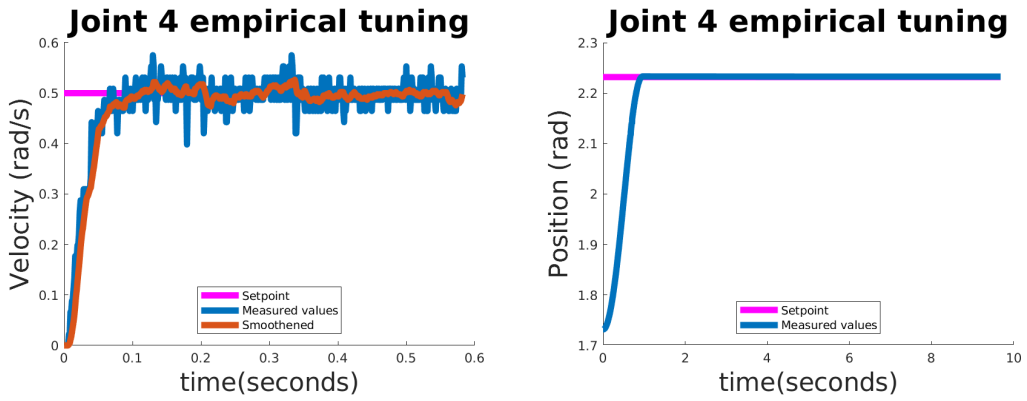


Figure 4.3: Trajectory tracking error on joint 5

The controller gains are tuned for the joint 4 as depicted in the Figures 4.4(a) 4.4(b) which follows the same procedure that has been followed for the joint number 5.



(a) Inner loop tuning

(b) Outerloop tuning

Figure 4.4: Inner, outer loop tuning for joint 4 with the velocity, position set-points of 0.5 rad/s, 0.5 rad respectively

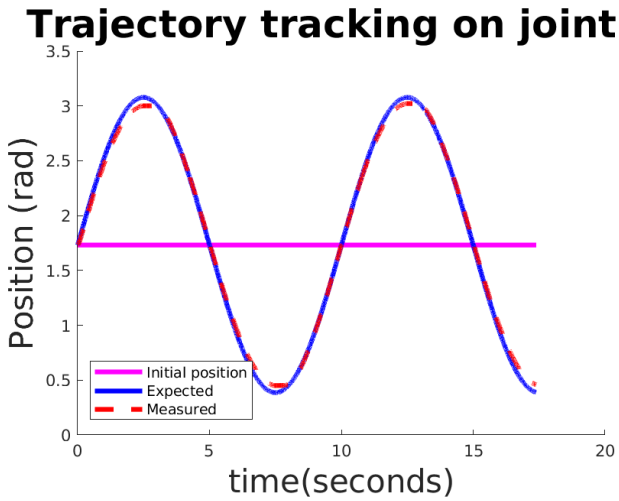


Figure 4.5: Trajectory tracking experiment using the sine wave on joint 4

The error between the set-point and the controller response for all the way-points of the sine wave are depicted in Fig. 4.6. The maximum error that has been observed in joint number 4 is 0.0779 rad at some point in time.

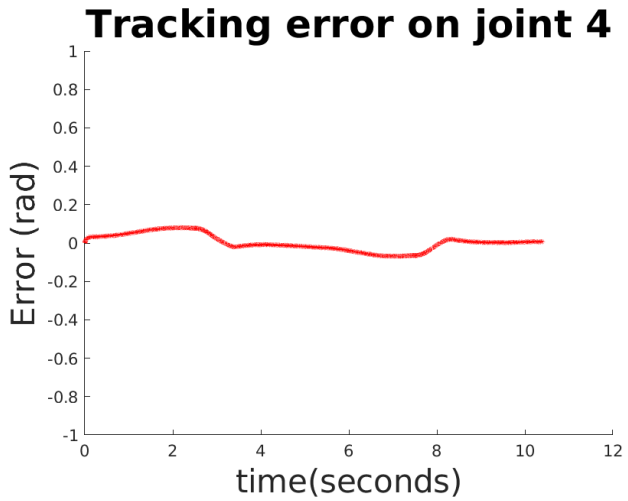


Figure 4.6: Trajectory tracking error on joint 4

The trajectory tracking error is caused due to many reasons such as the controller gains are not be optimal yet hence the tuning has to be re-considered, the presence of the cascaded controller where the optimality of the inner-loop control gains are suffering due to the noise in the measured data. The following Fig. 4.7 depicts the model, controller torques w.r.t. the position of joint 4.

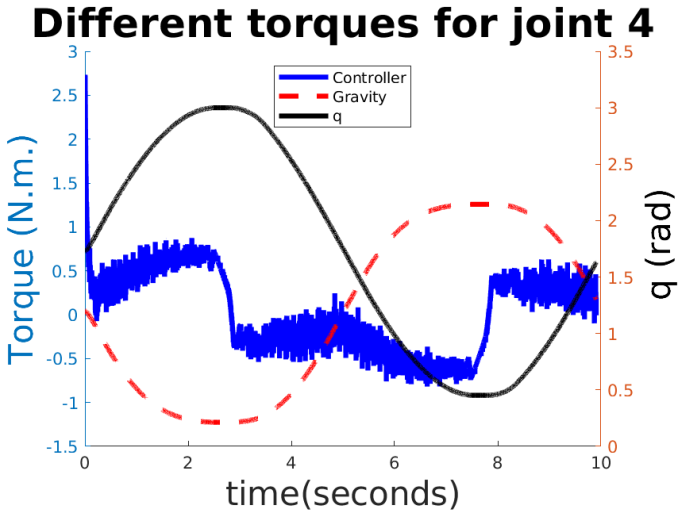


Figure 4.7: Modeled torque and controller torques are shown w.r.t. time along with the angular position for joint 4

It can be clearly seen that torque generated by the model do not have high frequency noise as expected. Again, the controller handles the disturbances in the manipulator joints producing high-frequency components. An important cause of this problem is that the model discrepancies introduce deviations in torque prediction which in-turn has a huge impact on the advanced model based controller. The friction compensation terms in the

dynamic model would improve the controller's performance further. Though the dynamic model is incomplete, the controller is able to follow the trajectories close-enough with a minimal deviations. The non real-time system is used for the development that produces latency issues in the feedback-control system.

4.2 Inertial parameter estimation

The inertial model parameters provided by the manufacturers are inaccurate and the identification of the model parameters are not complete. It is decided to go with the mass and CoM where the rotational inertia parameters are ignored in the system model of the youBot manipulator. The link parameters without the rotational inertia has been tested on the real robot for the gravity compensation and the joints are not sliding due to gravitational pull hence the link properties of the dynamic model in Orocos KDL excludes the rotational inertia in the computations.

Chapter 5

CONCLUSIONS AND FUTURE WORK

In this work, an attempt has been made to achieve the accurate trajectory tracking control on the youBot manipulator. Initially, the plan was to implement and test the controller in Simbody visualizer but the manipulator has an impact due to gravity and it starts sliding overtime where it is supposed to stay in the candle configuration. So, the conclusion on this module is that the model used in both the Simbody and KDL libraries are the same but there is a mismatch between these two libraries in the semantic level introduces such a problem. It is then decided to implement the controller in the youBot base at first to verify the correctness of both the safety and the cascade control. The model-based controller requires two important modules such as the model of the system and mainly the controller. It is not just enough to have the model of the system rather it has to be accurate enough to predict the feed-forward torques of the real system precisely. For which, the identification procedure is considered and the findings of the same are presented. It is really important to understand the semantics of the rigid-body algorithms given by different authors before implementing the algorithm itself. The lesson learned from the identification modules' findings are that the semantics differ from one author to another and the implicit conventions must be decoded before implementing the rigid-body algorithms provided by the authors. So, the identification procedure is partially completed with the findings and the model parameters can be used effectively once the identification of the parameters are complete. The Orocos KDL library is used for computing the model torques and this work has an important finding in the KDL frame diagram. The author mentions that the CoM point and the inertial frame can be located w.r.t. the root frame. But this representation is mismatching with the actual implementation where the inertial frame, CoM point can only be located w.r.t. the tip frame. The next important contribution of this work is the control module and there are two kinds of control schemes got investigated and implemented in this work. The alternate approach with the simple computed-torque controller is selected for proof of concept. The cascade PI control is implemented with KDL based feed-forward torque computation. It is necessary that the controller gains of the joints 4, 5 are tuned to the optimum empirically for achieving the better performance in tracking. The same procedure can be followed for all the other manipulator joints. An important contribution of this work is the inclusion of the safety control layer which is responsible for preventing any damages to the system. The safety controller shows promising results but there is a presence of the latency problem since

the implementation is done in the non real-time operating systems. Although the basic approach does the gravity compensation, the inaccurate rotational inertial parameters causes the problem in the torque prediction. It is also observed that the rotor moments of inertia has to be accounted in the model based controllers since the current system accounts only for the links' moments of inertia. The friction has also been investigated and tested with the basic approach where the static friction and Coulomb frictions are modeled and compensated in the manipulator joints based on the assumptions made. In spite of having the inaccurate model parameters of the system in this work, the controller has been evaluated in the joint level with the use of the simple sine waveform and the results were presented. The tracking errors on joint 4, 5 are 0.3171 N.m. and 0.0779 N.m. respectively.

Future work

The trajectory tracking control can be improved by adapting the current implementation with the following changes in the model-based controller.

- The real-time operating system can be used for avoiding the latency issues in the controller.
- The model discrepancies need to be resolved for which the identification procedure with the optimal excitation trajectories must be achieved, then the inaccurate model parameters can be replaced with the identified dynamic model parameters. The semantics involved in the rigid-body algorithm has to be corrected in both the motion and force transforms based on the findings of this work.
- Moving back to the actual computed-control scheme that is presented in the **basic approach** since this is the actual computed-torque control method.
- One of the important aspects of the unmodeled dynamics is called friction. So, the friction observer can be added to the control scheme as given in the basic approach.

BIBLIOGRAPHY

[1] T. D. aet, S. Bellens, R. Smits, E. Aertbelien, H. Bruyninckx, and J. D. Schutter. Geometric relations between rigid bodies (part 1): Semantics for standardization. *IEEE Robotics Automation Magazine*, 20(1):84–93, March 2013.

[2] C. H. An, C. G. Atkeson, J. D. Griffiths, and J. M. Hollerbach. Experimental evaluation of feedforward and computed torque control. *IEEE Transactions on Robotics and Automation*, 5(3):368–373, Jun 1989.

[3] C. H. An, C. G. Atkeson, and J. M. Hollerbach. Estimation of inertial parameters of rigid body links of manipulators. In *1985 24th IEEE Conference on Decision and Control*, pages 990–995, Dec 1985.

[4] K. Benjamin, M. Elias, M. Fässler, D. Scaramuzza, S. Huck, and J. Lygeros. Torque control of a kuka youbot arm. 2013.

[5] B. Bona and M. Indri. Friction compensation in robotics: an overview. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 4360–4367, Dec 2005.

[6] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[7] W. K. Chung, L.-C. Fu, and T. Kröger. *Motion Control*, pages 163–194. Springer International Publishing, Cham, 2016.

[8] J. J. Craig. *Introduction to robotics: mechanics and control*, volume 3. Pearson/Prentice Hall Upper Saddle River, NJ, USA:, 2005.

[9] C. C. de Wit, B. Siciliano, and G. Bastin. *Theory of robot control*. Springer Science & Business Media, 2012.

[10] A. Del Prete, N. Mansard, O. E. Ramos, O. Stasse, and F. Nori. Implementing torque control with high-ratio gear boxes and without joint-torque sensors. *International Journal of Humanoid Robotics*, 13(01):1550044, 2016.

[11] R. Featherstone. *Rigid body dynamics algorithms*. Springer, 2014.

[12] P. Jan. youbot driver. *online at* `https://github.com/youbot/youbot_driver`, 2011.

[13] P. Jan. Youbot driver's architecture. *online at* `https://janpaulus.github.io/d8/d74/architecture__overview.html`, 2011.

[14] T. D. Laet, S. Bellens, H. Bruyninckx, and J. D. Schutter. Geometric relations between rigid bodies (part 2): From semantics to software. *IEEE Robotics Automation Magazine*, 20(2):91–102, June 2013.

[15] K. H. Ng, C. F. Yeong, E. L. M. Su, and A. R. Husain. Implementation of cascade control for wheeled mobile robot straight path navigation. In *2012 4th International Conference on Intelligent and Advanced Systems (ICIAS2012)*, volume 2, pages 503–506, June 2012.

[16] D. Nguyen-Tuong, M. Seeger, and J. Peters. Computed torque control with nonparametric regression models. In *2008 American Control Conference*, pages 212–217, June 2008.

[17] H. Olsson, K. J. Åström, C. C. De Wit, M. Gäfvert, and P. Lischinsky. Friction models and friction compensation. *Eur. J. Control*, 4(3):176–195, 1998.

[18] T. Ozaki, T. Suzuki, T. Furuhashi, S. Okuma, and Y. Uchikawa. Trajectory control of robotic manipulators using neural networks. *IEEE Transactions on Industrial Electronics*, 38(3):195–202, June 1991.

[19] J. Rajagopal. Dynamic robot model parameter identification via domain specific optimization. WS17 HBRS - Plöger, Schneider Supervising, Dec 2017.

[20] M. C. Ruiz. *Haptic Teleoperation of the youBot with friction compensation for the base*. PhD thesis, MA thesis. Departamento de Ingeniería de Sistema y Automática, Universidad Carlos III de Madrid, 2012. URL: http://hdl. handle. net/10016/16267 (cit. on p. 168), 2012.

[21] D. Sellers. An overview of proportional plus integral plus derivative control and suggestions for its successful application and implementation. 2001.

[22] M. A. Sherman, A. Seth, and S. L. Delp. Simbody: multibody dynamics for biomedical research. *Procedia Iutam, Online at* `https://simtk.org/projects/simbody`, 2:241–261, 2011.

[23] R. Smits, H. Bruyninckx, and E. Aertbeliën. Kdl: Kinematics and dynamics library. *Avaliable: http://www.orocos.org/kdl*, 2011.

[24] J. Swevers, C. Ganseman, D. Tukel, J. DeSchutter, and H. VanBrussel. Optimal robot excitation and identification. *IEEE Transactions on Automation Science and Engineering*, 13(5):730–740, 1997.

[25] J. Swevers, W. Verdonck, and J. D. Schutter. Dynamic model identification for industrial robots. *IEEE Control Systems*, 27(5):58–71, Oct 2007.

[26] S. TEMEL, S. YAĞLI, and S. GÖREN. P, pd, pi, pid controllers. *Middle East Technical University, Electrical and Electronics Engineering Department*, April 2013.

[27] S. Traversaro and A. Saccon. Multibody dynamics notation. *Technische Universiteit Eindhoven, Tech. Rep*, 2016.

[28] J. Vantilt, E. Aertbeliën, F. De Groote, and J. De Schutter. Optimal excitation and identification of the dynamic model of robotic systems with compliant actuators. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 2117–2124. IEEE, 2015.

[29] youBot developers. Dynamic robot model. *online at* *http://www.youbot-store.com/developers/kuka-youbot-kinematics-dynamics-and-3d-model-81*, Nov 2016.

[30] K. J. ström and T. Hägglund. Pid controllers: theory, design, and tuning. *Instrument Society of America, Research Triangle Park, NC*, 10, 1995.

Appendix A

YouBot driver

The youBot base or the manipulator can be controlled in three different modes as discussed in the introduction section. The interface details [12] [13] of the driver that are used in this project are briefed in this section. It is important to obtain the handle reference for the youBot manipulator or the base in order to command the data or retrieve the sensed data from the joints. The handle to the youBot manipulator can be obtained by creating an object to the following class

```
youbot::YouBotManipulator *youbotArm(Arm_name, Path_to_config_directory);
```

and the handle for the youBot base can be obtained similarly

```
youbot::YouBotBase *youbotBase(Base_name, Path_to_config_directory);
```

This work explains only the implementation details of the youBot manipulator and the youBot base can also be controlled in the similar way. The robot joints can be controlled individually or collectively. The position/velocity/torque commands to the joints has to be prepared and can be given as follows

```
vector<youbot::JointAngleSetpoint> jointAngleCommands(Total_Joints);
vector<youbot::JointVelocitySetpoint> jointVelocityCommands(Total_Joints);
vector<youbot::JointTorqueSetpoint> jointTorqueCommands(Total_Joints);
```

where Total_Joints represents the number of joints in the manipulator. The measured joint information for all the joints can be read with the use of the following classes

```
vector<youbot::JointSensedAngle> jointAnglesMeasured(Total_Joints);
vector<youbot::JointSensedVelocity> jointVelocitiesMeasured(Total_Joints);
vector<youbot::JointSensedTorque> jointTorquesMeasured(Total_Joints);
```

With the use of the above classes, it is now possible to set or get the angle (in radians) for all the joints of manipulator with the use of the following methods

```
youBotArm->setJointData(jointAngleCommands);
youBotArm->getJointData(jointAnglesMeasured);
```

The velocity, torque modes can also be controlled similarly with the use of the above given methods.

Appendix B

Orocos KDL

At first the kinematic chain of the youBot manipulator has to be created as follows

```
KDL::Chain kinematicChain;
```

The manipulator is represented by the kinematic chain of rigid bodies or segments. The next step is to add the segments of the manipulator to the kinematic chain that has been created in the previous step.

The segments can be created by instantiating an object to the following class

```
KDL::Segment segment(name, joint_axis, frame, rigid_body_inertia);
```

where the segment defines both the joint information and link properties which are then added to the chain by using the following method

```
kinematicChain.addSegment(segment);
```

Once the complete kinematic chain of the manipulator is created, the kinematics and dynamics of kinematic chains can be computed. The Orocos KDL library provide solvers for both the forward and inverse kinematics computations. This work made use of the forward kinematics solver to find the end-effector's pose and the solver can be instantiated and invoked as given below.

```
KDL::ChainFkSolverPos_recursive::JntToCart(q, T)
```

where q represents the joint position inputs for the manipulator joints and T represents the frame (KDL::Frame) that gives the pose information of the end-effector. The dynamics of the kinematic chain can be computed by using the recursive Newton-Euler formulation which is implemented based on the article [11]. Though the semantics used is same as the article, there is an important difference in the semantics of the algorithm in KDL comparing the article, i.e. In [11], author uses the spatial vector transforms in the algorithm whereas KDL does not uses the spatial form. The ID solver which is used in KDL can be invoked as follows

```
KDL::ChainIdSolver_RNE idsolver;
idsolver.CartToJnt(position, velocity, acceleration, wrench, torques);
```

Appendix C

Simbody

At first the visualizer has to be created with the use of the class SimTK::Visualizer where the properties can be adjusted based on the system needs and an event reporter (PeriodicEventReporter) is attached to handle the events of the system at the defined intervals. The time stepper (SimTK::TimeStepper) uses SimTK::RungeKuttaMersonIntegrator to take the system forward through time. The system is initialized after the system model is created. The forces can be commanded to the manipulator joints as generalized forces by using the class SimTK::GeneralForceSubSystem. The more detailed reference can be found in[6] and the implementation details are specified as follows. In Simbody, the multi-body system is created at first and the sub-systems are attached for the further operations. The system has a subclass SimTK::MultibodySystem to work with the general multi-body systems along with the sub-system SimTK::SimbodyMatterSubSystem which is responsible for defining all the rigid bodies in the system.

```
SimTK::MultibodySystem system;
SimTK::SimbodyMatterSubSystem matterSubSystem(system);
```

The gravity can be simulated with the use of the following class

```
SimTK::Force::Gravity gravity = SimTK::Force::Gravity();
```

A rigid body can be created and the link properties can be attached with the help of the following classes

```
SimTK::Body rigid_body;
rigid_body = SimTK::Body::Rigid(SimTK::MassProperties(mass, CoM, Inertia));
```

The joints can be created with the use of the MobilizedBody class

```
SimTK::MobilizedBody body(Number_of_joints);
```

Before attaching the rigid bodies to the sub-system, it is important to attach the base body to the ground (parent body) which is the origin by using the weld joint

```
body = SimTK::MobilizedBody::Weld();
```

[6] https://simtk.org/api_docs/simbody/3.5/namespaceSimTK.html

The revolute joints can be attached to the mobilized body with the help of the Pin class which attaches a new body to the system with the one DoF constraint to the predecessor.

```
body = SimTK::MobilizedBody::Pin();
```

The joint information such as position, velocity and acceleration are retrieved from the state of the system that gets updated in a periodic manner. The torque sensor has been created in this work with the use of the SimTK interface that measures the gravitational, bias and the custom forces from the multi body system. It is possible to command the generalized coordinate with the particular motion as a function which can be step or sinusoid use of the following class

```
SimTK::Constraint::PrescribedMotion
```

The two-link manipulator has been created to test the prescribed motion in this work to verify the semantic correctness and setting the joint positions required the update in the state variable which is a constant in Simbody and the setup for the 2-link manipulator is depicted in Fig. C.1.

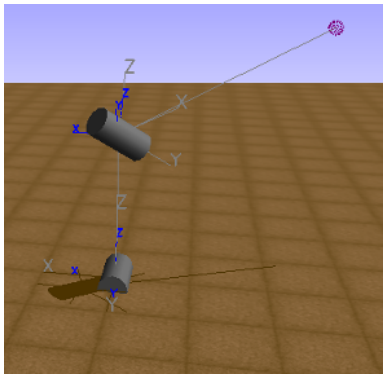
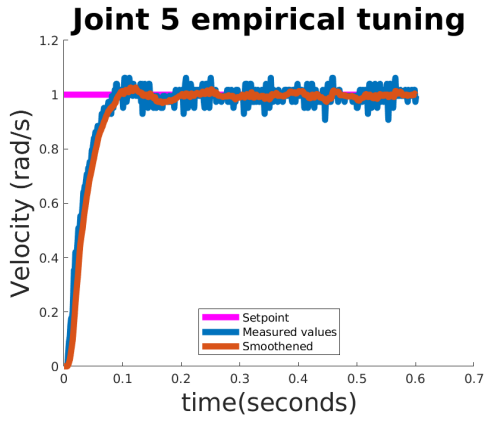


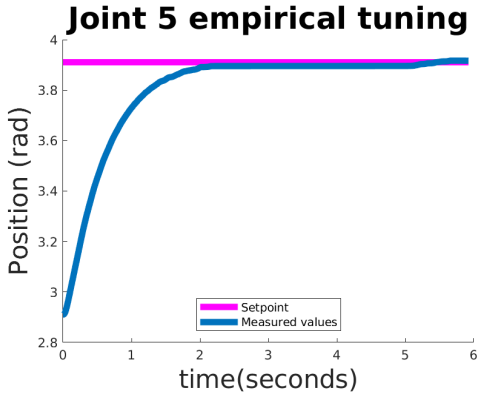
Figure C.1: Two-link manipulator is constructed with the cylinder objects in Simbody visualizer for commanding the prescribed motion to the mobilizer

Appendix D

Tuning of the controller gains

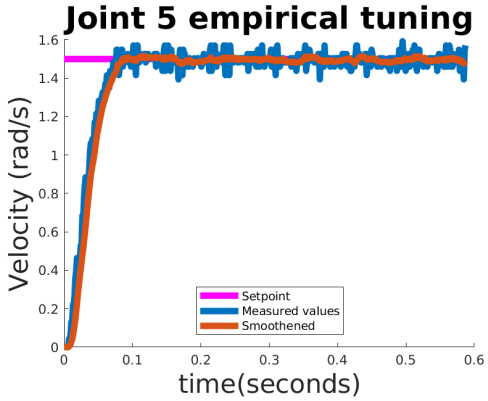


(a) Inner loop tuning

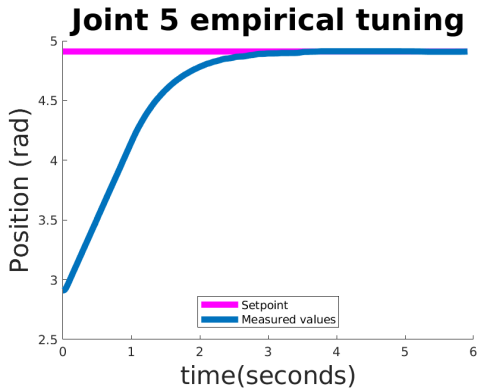


(b) Outerloop tuning

Figure D.1: Inner, outer loop tuning for joint 5 with the velocity, position set-points of 1.0 rad/s, 1.0 rad respectively



(a) Inner loop tuning



(b) Outerloop tuning

Figure D.2: Inner, outer loop tuning for joint 5 with the velocity, position set-points of 1.5 rad/s, 1.0 rad respectively