Design and Fabrication of Experiment for Dynamic Analysis of Mechanisms

submitted in partial fulfilment of the requirements

for the degree of

BACHELOR OF TECHNOLOGY

in

MECHANICAL ENGINEERING

by

AAKASH ME16B001

A. AKHIL ME16B003

Supervisor(s)

Dr. Sriram Sundar

DEPARTMENT OF MECHANICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY TIRUPATI

June 2020

DECLARATION

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Place: Tirupati
Date: 12-06-2020

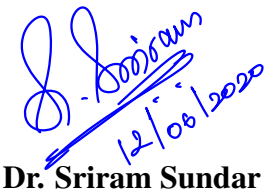
Signature
Aakash
ME16B001

Place: Tirupati
Date: 12-06-2020

Signature
A. Akhil
ME16B003

BONA FIDE CERTIFICATE

This is to certify that the thesis titled **Design and Fabrication of Experiment for Dynamic Analysis of Mechanisms**, submitted by **Aakash** and **A. Akhil**, to the Indian Institute of Technology, Tirupati, for the award of the degree of **Bachelor of Technology**, is a bona fide record of the research work done by them under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.



Place: Tirupati
Date: 12-06-2020

Dr. Sriram Sundar
Guide
Assistant Professor
Department of Mechanical Engineering
IIT Tirupati - 517501

ACKNOWLEDGEMENTS

Thanks to all those who helped us during the course of our B.Tech. project at IIT Tirupati. We are thankful to and fortunate enough to get constant encouragement, support and guidance from Dr. Sriram Sundar of Mechanical Engineering Department which helped us in successfully completing our project work. We would also like to extend our sincere esteems to all staff in laboratory for their timely support.

To our families, thank you for encouraging us in all of our pursuits and inspiring us to follow our dreams. We are especially grateful to our parents, who supported us emotionally and financially.

ABSTRACT

KEYWORDS: Slider crank mechanism; Four bar mechanism; Six-bar mechanism; Dynamic analysis; Kinematic analysis; Simulation.

This report explains the detailed process of design and analysis of an experimental setup, which consists of a 3-in-1 mechanism. The setup is designed to perform static, kinematic and dynamic analysis of mechanisms. The design and analysis were performed analytically and numerically by using various commercial software's. The results thus obtained were used to determine the link geometry of the mechanism and in selecting the suitable sensors which in turn will be used for fabrication. The instrumentation of the sensors and actuators with GUI has been done, also a standalone software 'OpenKDM' has been designed to carry out the analysis (project website: https://nimrobotics.github.io/OpenKDM/).

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

The following list describes the significance of various abbreviations and acronyms used throughout the report.

KDM	Kinematics and Dynamics of Machinery
PMSM	Permanent Magnet Synchronous Motor
CAD	Computer Aided Design
CSV	Comma Seperated Values
FBD	Free Body Diagram
GUI	Graphical User Interface
CLI	Command Line Interface
OpenKDM	Open-source Kinematics and Dynamics of Machinery

NOTATION

The following notations are used throughout the report.

$\vec{r}_{A/B}$	Position vector of point A with respect to point B, m
ω_{AB}	Angular velocity of link AB, $\frac{rad}{s}$
$\vec{V}_{A/B}$	Velocity vector of point A with respect to point B, $\frac{m}{s}$
$\vec{a}_{A/B}$	Acceleration vector of point A with respect to point B, $\frac{m}{s^2}$
α_{AB}	Angular acceleration of link AB, $\frac{rad}{s^2}$

CHAPTER 1

INTRODUCTION

1.1 Motivation and objective

The study of the *"Kinematics and Dynamics of Machinery" (IITT course code: ME2206)* lies at the very core of a mechanical engineering background. Although, little has changed in the way the subject is presented, our methodology brings the subject alive and current. We present the design and fabrication of a novel experimental setup for carrying out static, kinematic and dynamic analysis of three different mechanisms in a single setup . The mechanism is designed(Figure: 1.1) to be configurable to three different types of mechanisms namely - double crank, slider crank and a six bar mechanism depending on the use case as shown in Figure 1.2. The mechanism has retrofitted parts (different link lengths and sliders) to facilitate multiple experiments in the same setup.

The learner gets to "play" with the mechanism parameters and immediately understand their effects. This will enhance one's grasp of the concepts and the development of analytical skills. Hence greatly supplementing and reinforcing the theoretical understanding of the undergraduate students taking the course.

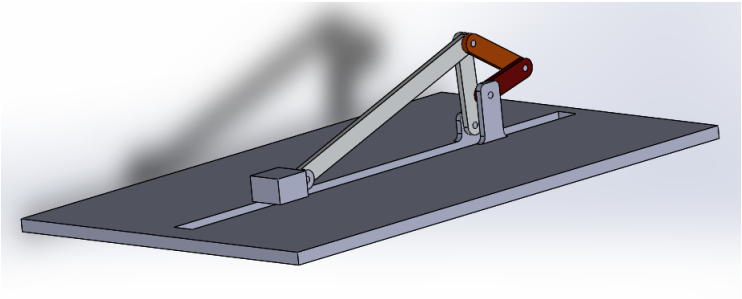


Figure 1.1: CAD model of the mechanism

1.2 Present method(s) and challenges

In the present scenario finding standard mechanisms integrated with different sensor measurement and data acquisition is hard to find. Almost all the mechanisms that are

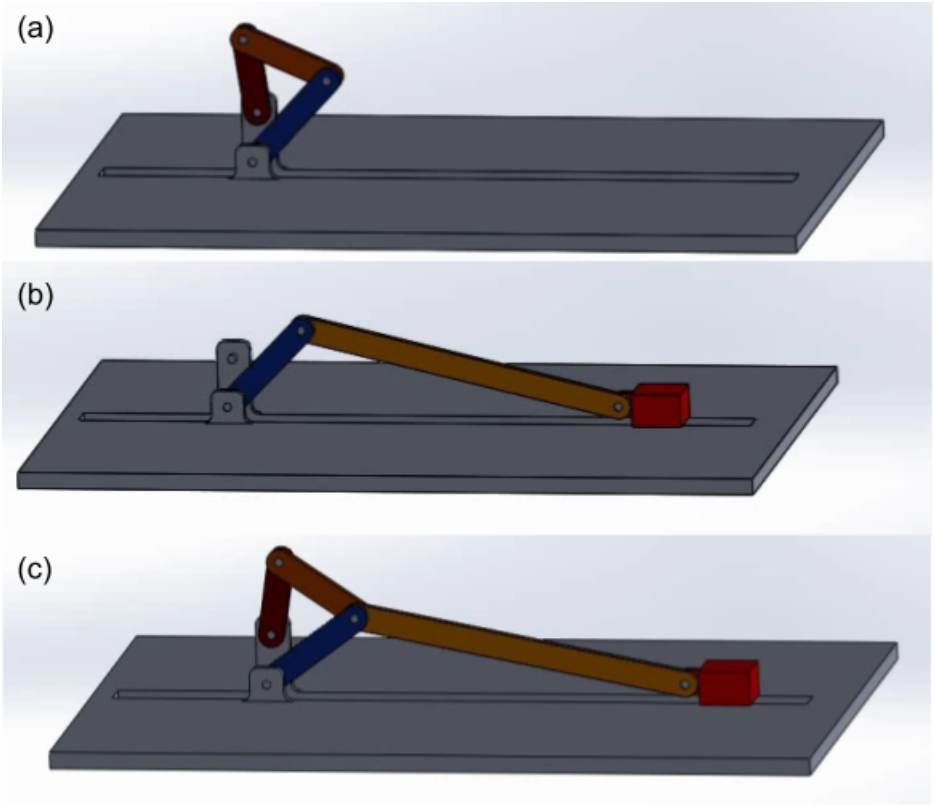


Figure 1.2: Three possible configurations of the mechanism: (a) Four bar, (b) Slider crank and (c) Six bar

available off-the-shelf do not have any data measurement capability and thus can only be used for qualitative kinematic demonstrations purpose and not as an experimental setup.

In chapter 2 we present thoroughly all the methodologies used in designing and modelling the mechanisms. This includes the analysis carried out by analytical as well as numerical methods. In the chapter that follows we bring out the various design considerations that has to be taken into account while manufacturing the product. The final two sections present the result(s) and the conclusion(s).

CHAPTER 2

ANALYSIS OF THE MECHANISMS

2.1 Kinematic analysis

This section presents the methodology employed in order to simulate the kinematics of the various mechanisms. In order to carry out the preliminary analysis of the mechanism, *GeoGebra* an interactive math and geometry software was used. This provided the coarse link lengths required for having the least deviant constant velocity mechanism. Once the coarse link lengths were obtained, developed an in-house Python code that can do iterations on the link lengths so as to minimise any fluctuations in the constant velocity mechanism. The visual output of the Python simulation can be seen as a graph while rest of the simulation data is store in a CSV file.

2.1.1 Mathematical formulation

This section brings out the equations of motion associated with the three different mechanisms (Norton, 2011; Uicker *et al.*, 2011). These equations are used to carry out the kinematic and dynamics method theoretically, which then in turn be compared with the numerical solutions obtained by other simulation tools.

Slider crank mechanism

The equations of motion pertaining to the slider crank mechanism (Figure 2.1) are presented in this subsection.

$$\vec{V}_{B/A} = \vec{V}_B - \vec{V}_A \tag{2.1}$$

For link *OA* we have

$$\vec{V}_{A/O} = \vec{\omega}_{AO} \times \vec{r}_{A/O} = \vec{V}_A \tag{2.2}$$

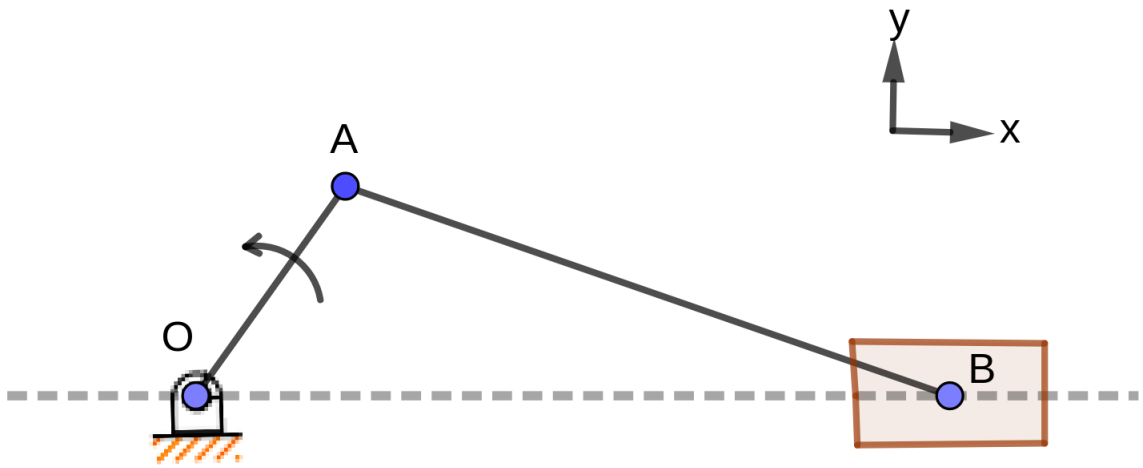


Figure 2.1: Slider crank mechanism

Since 'O' is ground joint we have $\vec{V}_O = 0 \implies \vec{V}_{A/O} = \vec{V}_A - \vec{V}_O = \vec{V}_A$. As point B lies on the slider and is moving leftwards it will have velocity along the negative x-direction i.e. $\vec{V}_B = -V_B\hat{i}$; because $\vec{\omega}_{AO} = \omega_{AO}\hat{k}$. For link AB we have

$$\vec{V}_{B/A} = \vec{\omega}_{BA} \times \vec{r}_{B/A} \tag{2.3}$$

Substituting equation 2.3 in equation 2.1 we obtain

$$\vec{\omega}_{BA} \times \vec{r}_{B/A} = \vec{V}_B - \vec{V}_A \tag{2.4}$$

Using equation 2.4 we can find ω_{BA} and V_B. For finding the accelerations we can write

$$\vec{a}_{A/O} = \vec{a}_A = \vec{a}_A^t + \vec{a}_A^n \tag{2.5a}$$

$$\vec{a}_A^t = \vec{\alpha_{A/O}} \times \vec{r_{A/O}} \tag{2.5b}$$

$$\vec{a}_A^n = \vec{\omega_{AO}} \times \vec{\omega_{AO}} \times \vec{r_{A/O}} \tag{2.5c}$$

where, $\vec{a}_A^t$ and $\vec{a}_A^n$ denote the tangential and normal acceleration of point A. As point B lies on the slider, we have $\vec{a}_B = a_B\hat{i}$.

$$\vec{a}_{B/A} = \vec{a}_{B/A}^t + \vec{a}_{B/A}^n = \vec{a}_B - \vec{a}_A \tag{2.6}$$

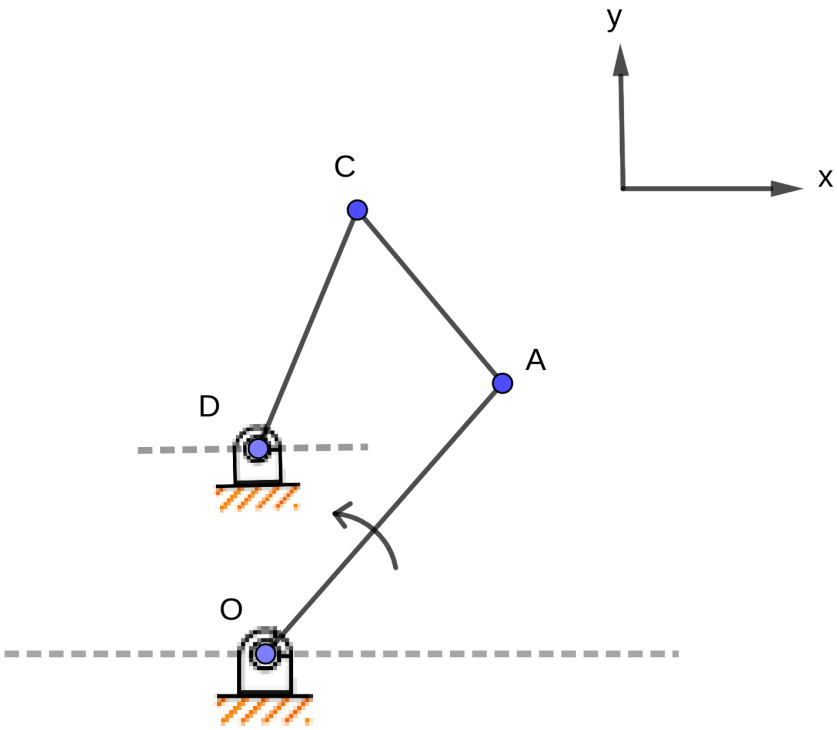


Figure 2.2: Four bar mechanism

Using the above equation we can find α_{BA} and $\vec{a}_B$.

Four bar mechanism (with four revolute joints)

This subsection lists the mathematical equations pertaining to the four bar mechanism (Figure 2.2).

$$\vec{V}_{A/O} = \vec{\omega}_{AO} \times \vec{r}_{A/O} = \vec{V}_A \tag{2.7a}$$

$$\vec{V}_{C/D} = \vec{\omega}_{CD} \times \vec{r}_{C/D} = \vec{V}_C \tag{2.7b}$$

$$\vec{V}_{A/C} = \vec{\omega}_{AC} \times \vec{r}_{A/C} = \vec{V}_A - \vec{V}_C = \vec{\omega}_{AO} \times \vec{r}_{A/O} - \vec{\omega}_{CD} \times \vec{r}_{C/D} \tag{2.8}$$

Using the above equation we can find ω_{CD} and ω_{AC}.

$$\vec{a}_{A/O} = \vec{a}_A = \vec{a}_A^t + \vec{a}_A^n \tag{2.9a}$$

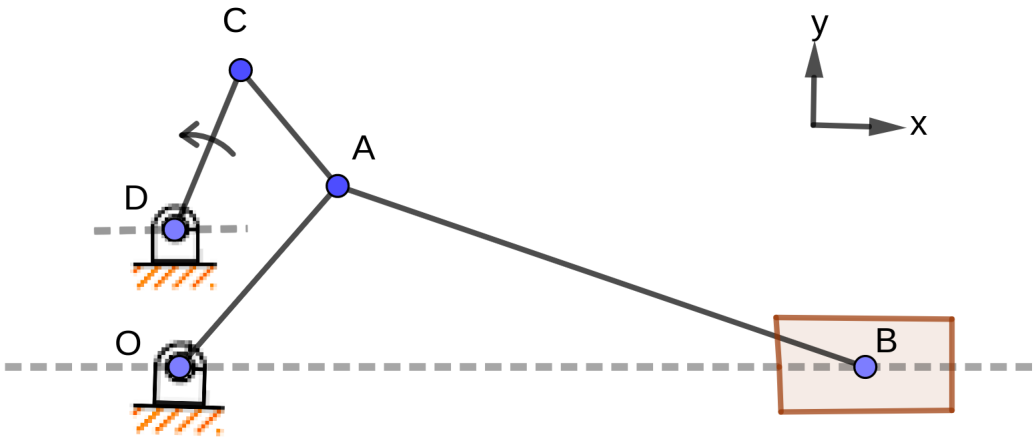


Figure 2.3: Six bar mechanism

$$\vec{a}_{C/D} = \vec{a}_C = \vec{a}_C^t + \vec{a}_C^n \tag{2.9b}$$

$$\vec{a}_{A/C} = \vec{a}_{A/C}^t + \vec{a}_{A/C}^n = \vec{a}_A - \vec{a}_C \tag{2.10}$$

Here $\vec{a}^t = \vec{\alpha} \times \vec{r}$ and $\vec{a}^n = \vec{\omega} \times \vec{\omega} \times \vec{r}$. Using the above equations we can find α_{AC} and α_{CD}

Six bar mechanism

This subsection presents the mathematical equations associated with the six bar mechanism (Figure 2.3).

$$\vec{V}_{C/D} = \vec{\omega}_{CD} \times \vec{r}_{C/D} = \vec{V}_C \tag{2.11}$$

$$\vec{V}_{A/C} = \vec{\omega}_{AC} \times \vec{r}_{A/C} = \vec{V}_A - \vec{V}_C = \vec{\omega}_{AO} \times \vec{r}_{A/O} - \vec{\omega}_{CD} \times \vec{r}_{C/D} \tag{2.12}$$

Using the above equation we can find ω_{OA} and ω_{AC}.

$$\vec{V}_{B/A} = \vec{V}_B - \vec{V}_A \tag{2.13}$$

For link OA we have

$$\vec{V}_{A/O} = \vec{\omega}_{AO} \times \vec{r}_{A/O} = \vec{V}_A \tag{2.14}$$

Since 'O' is ground joint we have $\vec{V}_O = 0 \implies \vec{V}_{A/O} = \vec{V}_A - \vec{V}_O = \vec{V}_A$. As point B lies on the slider and is moving leftwards it will have velocity along the negative x-direction i.e. $\vec{V}_B = -V_B\hat{i}$; because $\vec{\omega}_{AO} = \omega_{AO}\hat{k}$. For link AB we have

$$\vec{V}_{B/A} = \vec{\omega}_{BA} \times \vec{r}_{B/A} \tag{2.15}$$

Substituting equation 2.15 in equation 2.13 we obtain

$$\vec{\omega}_{BA} \times \vec{r}_{B/A} = \vec{V}_B - \vec{V}_A \tag{2.16}$$

Using equation 2.16 we can find ω_{BA} and V_B.

For finding the accelerations we can write

$$\vec{a}_{C/D} = \vec{a}_C = \vec{a}^t_C + \vec{a}^n_C \tag{2.17}$$

$$\vec{a}_{A/C} = \vec{a}^t_{A/C} + \vec{a}^n_{A/C} = \vec{a}_A - \vec{a}_C \tag{2.18}$$

Here $\vec{a}^t = \vec{\alpha} \times \vec{r}$ and $\vec{a}^n = \vec{\omega} \times \vec{\omega} \times \vec{r}$. Using the above equations we can find α_{AC} and α_{OA}

$$\vec{a}_{A/O} = \vec{a}_A = \vec{a}^t_A + \vec{a}^n_A \tag{2.19a}$$

$$\vec{a}^t_A = \vec{\alpha_{A/O}} \times \vec{r_{A/O}} \tag{2.19b}$$

$$\vec{a}^n_A = \vec{\omega_{AO}} \times \vec{\omega_{AO}} \times \vec{r_{A/O}} \tag{2.19c}$$

where, $\vec{a}^t_A$ and $\vec{a}^n_A$ denote the tangential and normal acceleration of point A. As point B lies on the slider, we have $\vec{a}_B = a_B\hat{i}$.

$$\vec{a}_{B/A} = \vec{a}^t_{B/A} + \vec{a}^n_{B/A} = \vec{a}_B - \vec{a}_A \tag{2.20}$$

Using the above equation we can find α_{BA} and $\vec{a}_B$.

2.1.2 GeoGebra modelling

This subsection provides an insight into the preliminary analysis done by using off-the-shelf software - GeoGebra for determining initial link lengths (Hohenwarter *et al.*, 2013). First the mechanism was created inside GeoGebra geometry with sliders for varying the link lengths manually. Later the slider velocity was plotted by using graphical analysis of the mechanism inside GeoGebra itself (as shown in Figure 2.4). In order to find the required link lengths yielding constant velocity mechanism, each of the link lengths were varied to get the constant velocity mechanism with least variation while keeping the others fixed. The manual iterations are shown in Table 2.1.

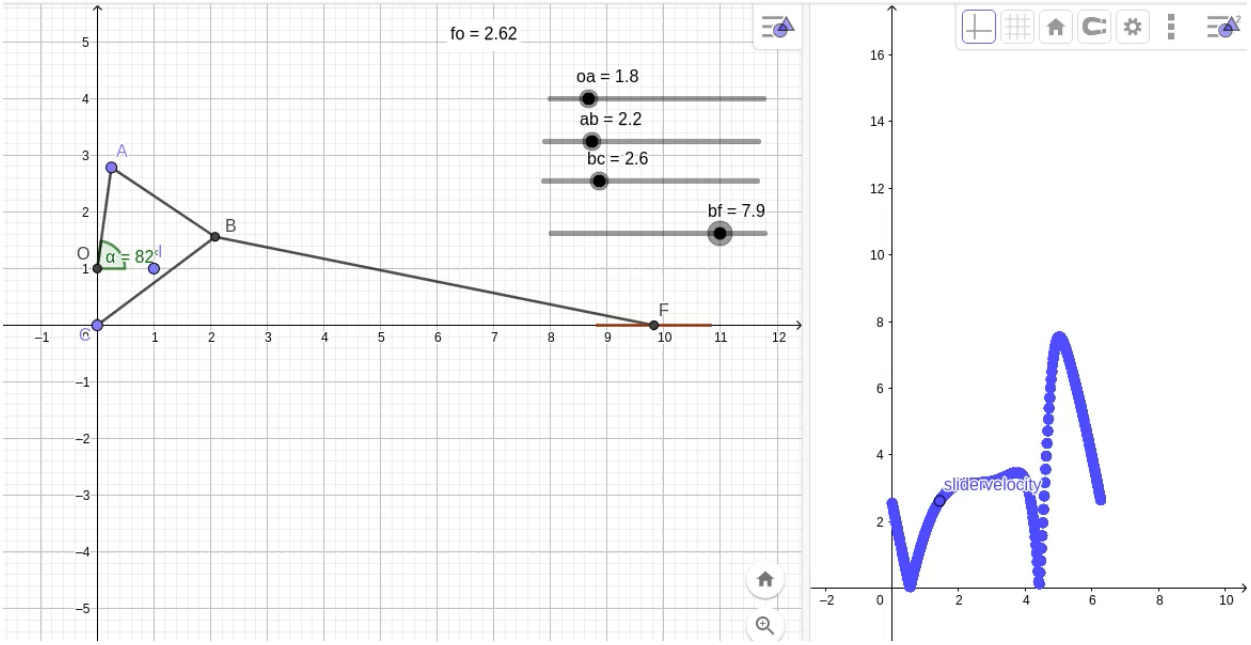


Figure 2.4: Snapshot from GeoGebra showing the mechanism (left) and the velocity profile (right)

2.1.3 Python simulation

This section deals with further refining the preliminary links lengths which were obtained from GeoGebra modelling. The initial lengths are fed to an in-house Python script (see appendix) to get optimum link lengths for nearly constant slider velocity (Python Core Team, 2018). The algorithm performs the iterations by varying each of the link by step size of $0.1 \times$ unit length. It also stores the peak to valley distance for each of the obtained velocity profiles which can be used to select the link lengths giving minimum velocity variation for the constant velocity range. Figure 2.5 shows the mechanism animation as

OC	OA	AB	BC	BF	Mean slider velocity	Peak to valley of slider velocity	ω of crank
1	2	1.6	2	6	2.25	0.5	2
1	2	1.4	2	6	2.28	0.81	2
1	2	1.2	2	6	2.37	1.24	2
1	2	2	2	6	2.385	0.77	2
1	2.2	1.6	2	6	2.31	0.83	2
1	1.8	1.6	2	6	2.375	0.75	2
1	2	1.6	2.2	6	2.545	0.37	2
1	2	1.6	1.8	6	1.92	0.76	2
1	2	1.6	2.4	6	2.87	0.26	2
1	2	1.6	2.6	6	3.2	0.25	2
1	2	1.6	2.8	6	3.52	0.36	2
1	2	1.6	2.6	7	3.135	0.15	2
1	2	1.6	2.6	7.5	3.2	0.12	2
1	2	1.6	2.6	8	3.24	0.09	2

All lengths are in *units*, velocity are in $units/s$, angular velocity ω is in rad/s.

Table 2.1: GeoGebra results

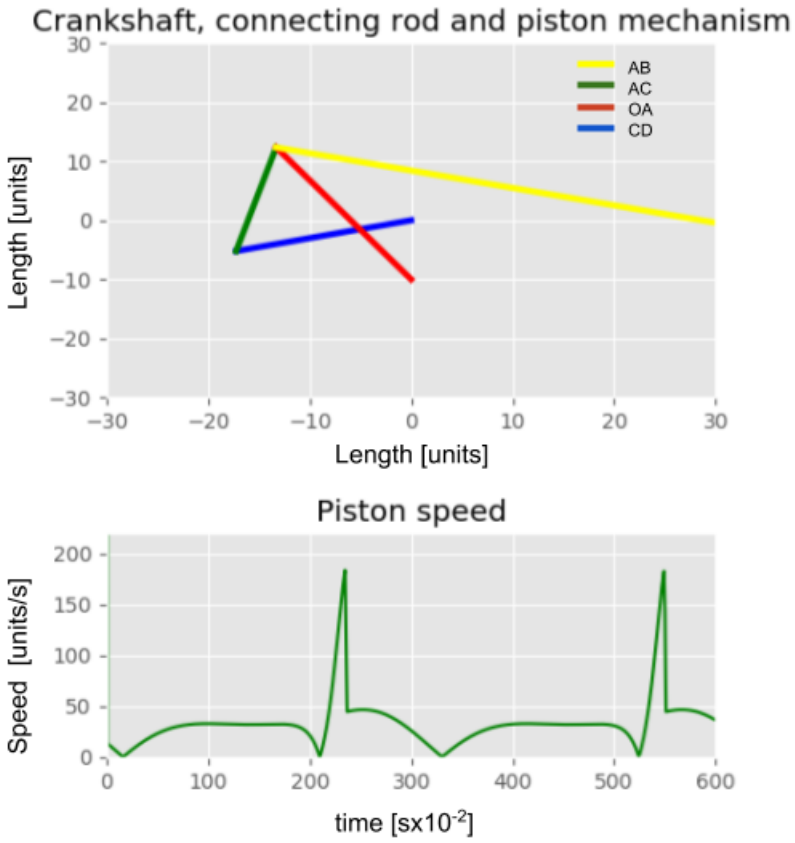


Figure 2.5: Output window from Python for kinematic simulation

well as the velocity plot having a near constant velocity for an interval.

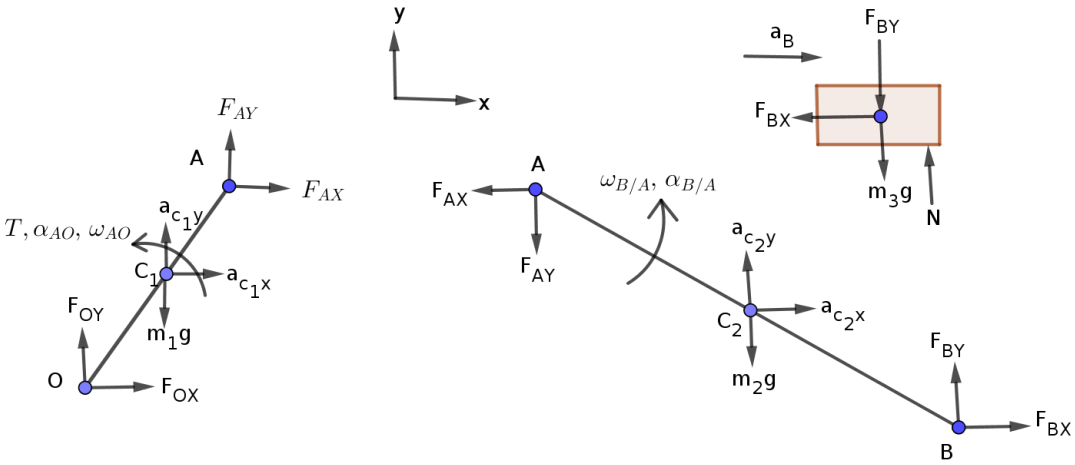


Figure 2.6: Free Body Diagram for Slider Crank Mechanism

2.2 Dynamic analysis

For performing the dynamic analysis of the mechanism we used MathWorks Simscape Mechanical package. A 3D model was created in the package itself and analysed the same for various forces, torques, acceleration and velocities by taking the material as aluminium. This analysis is necessary in order to ease the process of selection of actuator, material and the sensor system.

2.2.1 Mathematical formulation

This subsection brings out the mathematical equations for performing dynamic analysis of all the three possible mechanism (Ghosh and Mallik, 2002).

Slider crank mechanism

Force and Torque balance for link *OA* using Figure 2.6

$$F_{OX} + F_{AX} = m_1 a_{C_1 x} \tag{2.21a}$$

$$F_{OY} + F_{AY} = m_1 g + m_1 a_{C_1 y} \tag{2.21b}$$

$$I_{AO}\vec{\alpha}_{AO} + \vec{r}_{C_1/O} \times m_1 \vec{a}_{C_1} = \vec{r}_{C_1 O} \times m_1 g(-\hat{j}) + \vec{r}_{A/O} \times \vec{F}_A + \vec{T} \tag{2.21c}$$

where, $\vec{F}_A = \vec{F}_{AX}\hat{i} + \vec{F}_{AY}\hat{j}$. Similarly for link AB we have

$$-F_{AX} + F_{BX} = m_2 a_{C_2 x} \tag{2.22a}$$

$$F_{BY} = m_2 a_{C_2 y} + F_{AY} + m_2 g \tag{2.22b}$$

$$I_{BA}\vec{\alpha}_{B/A} + \vec{r}_{C_2/A} \times m_2 \vec{a}_{C_2} = \vec{r}_{C_2/A} \times m_2 g(-\hat{j}) + \vec{r}_{B/A} \times \vec{F}_B \tag{2.22c}$$

where, $\vec{F}_B = \vec{F}_{BX}\hat{i} + \vec{F}_{BY}\hat{j}$.

Force balance for the slider

$$F_{BY} + m_3 g = N \tag{2.23a}$$

$$m_3 a_B = -F_{BX} \tag{2.23b}$$

Four bar mechanism (with four revolute joints)

Force and Torque balance for link *OA* using Figure 2.7

$$\sum \vec{F}_X = m_1 \vec{a}_{c_1 x} \tag{2.24a}$$

$$\sum \vec{F}_Y = m_1 \vec{a}_{c_1 y} \tag{2.24b}$$

$$\sum \vec{T} = I_1 \vec{\alpha}_{AO} \tag{2.24c}$$

Force and Torque balance for link *AC* using Figure 2.7

$$\sum \vec{F}_X = m_4 \vec{a}_{c_4 x} \tag{2.25a}$$

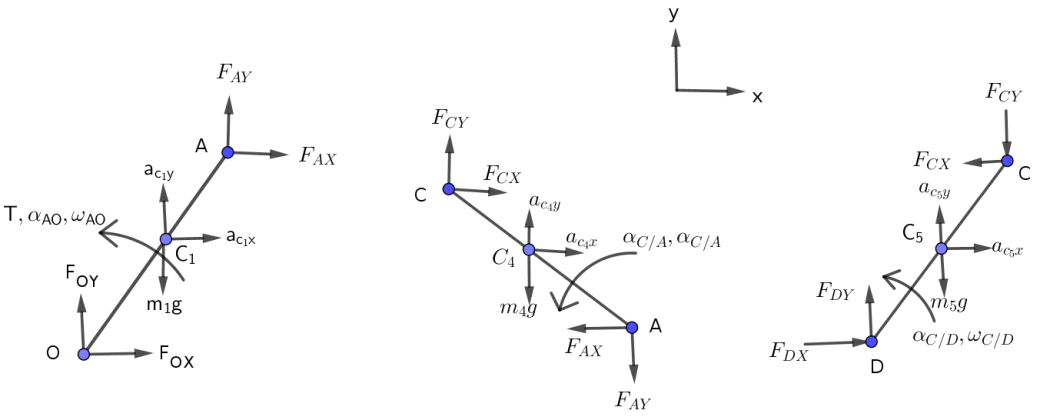


Figure 2.7: Free Body Diagram for various link of four-bar mechanism

$$\sum \vec{F}_Y = m_4 \vec{a}_{c_4y} \tag{2.25b}$$

$$\sum \vec{T} = I_4 \vec{\alpha}_{AC} \tag{2.25c}$$

Force and Torque balance for link CD using Figure 2.7

$$\sum \vec{F}_X = m_5 \vec{a}_{c_5x} \tag{2.26a}$$

$$\sum \vec{F}_Y = m_5 \vec{a}_{c_5y} \tag{2.26b}$$

$$\sum \vec{T} = I_5 \vec{\alpha}_{CD} \tag{2.26c}$$

Six bar mechanism

Force and Torque balance for all links using Figure 2.8

$$\sum \vec{F}_X = m\vec{a}_{cx}, \sum \vec{F}_Y = m\vec{a}_{cy}, \sum \vec{T} = I\vec{\alpha} \tag{2.27}$$

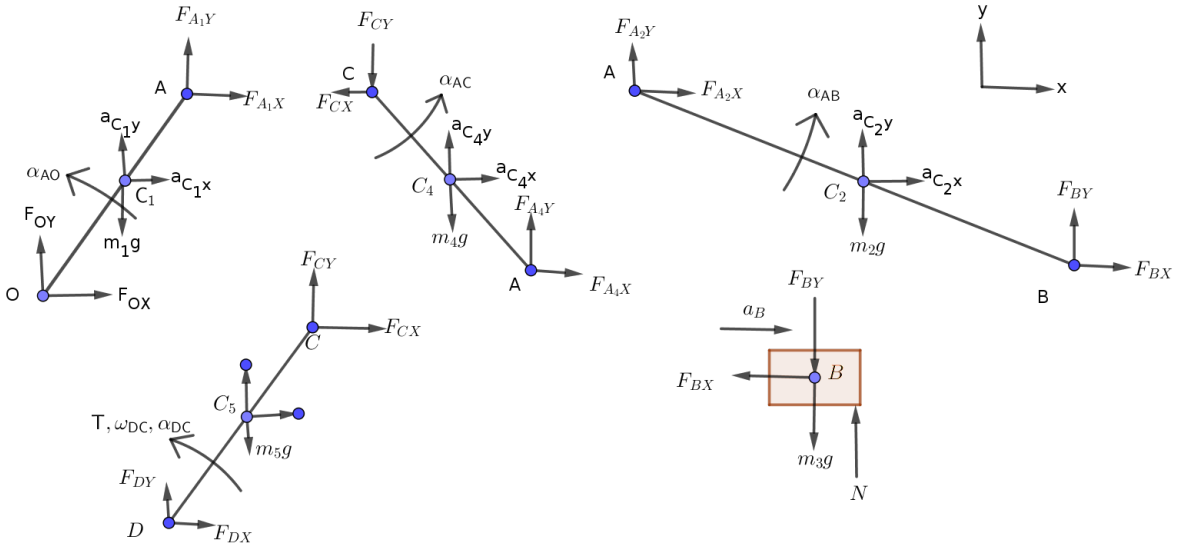


Figure 2.8: Free Body Diagram for various links of six-bar mechanism

$$F_{A_1X} + F_{A_2X} + F_{A_4X} = 0 \tag{2.28a}$$

$$F_{A_1Y} + F_{A_2Y} + F_{A_4Y} = 0 \tag{2.28b}$$

Force balance for Slider using Figure 2.8

$$F_{BY} + m_3 g = N \tag{2.29a}$$

$$m_3 a_B = -F_{BX} \tag{2.29b}$$

2.2.2 Using commercial software: MathWorks Simscape Multibody

This sections deals with the dynamic analysis of the mechanism using Mathworks Simscape multibody. A MathWorks Simscape model (Figure 2.9) was created based on the link lengths that were obtained from the kinematic analysis.The link width is taken as $3cm$ and link thickness as $1cm$ while taking aluminum as material for all links and slider. (MAT, 2017)

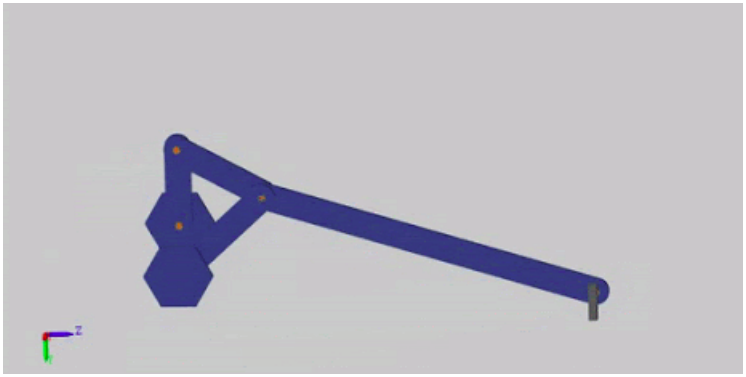
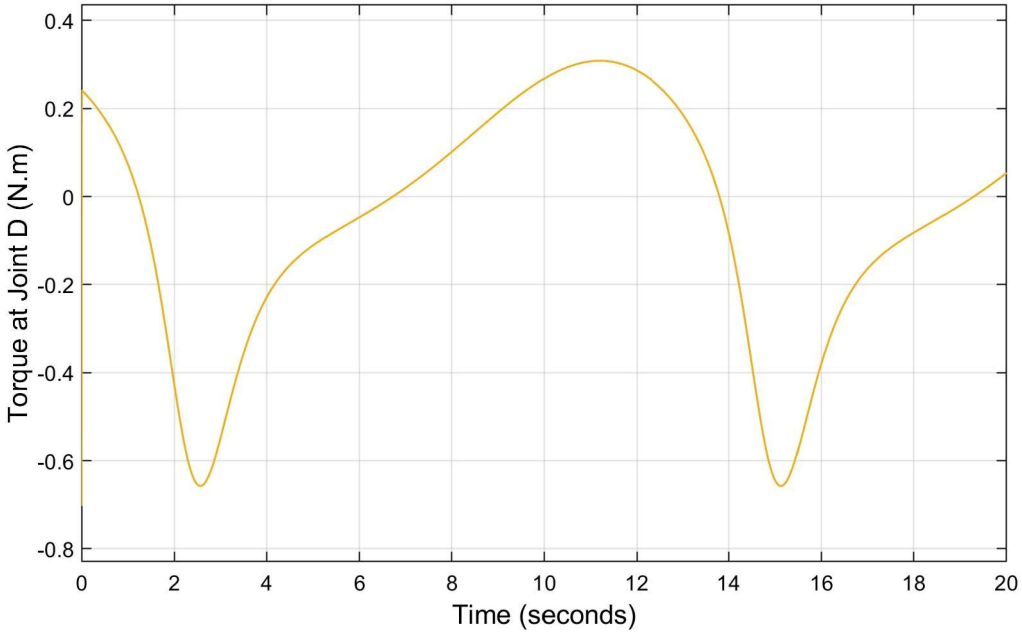
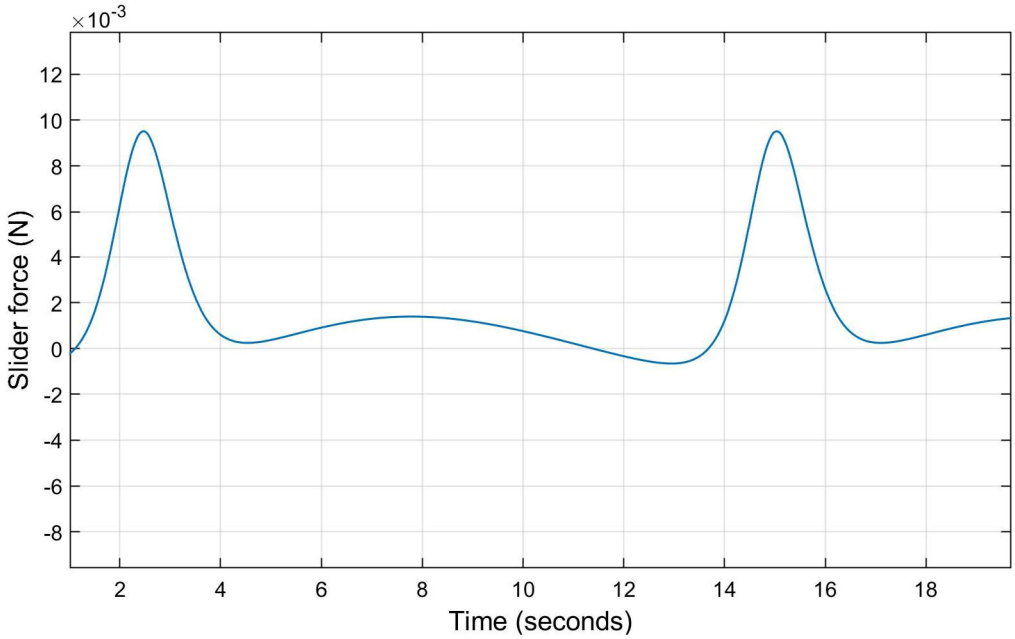


Figure 2.9: CAD model in MATLAB Simscape



(a) Input torque



(b) Slider force

Figure 2.10: Results obtained from Simscape

CHAPTER 3

DESIGN AND FABRICATION

3.1 Sensor selection

Based on the results of Kinematic, Dynamic analysis using MathWorks Simscape, Selected the 2-phase incremental optical rotary encoder to measure angular velocity and angular acceleration of links, tri-axial accelerometer is to measure linear acceleration and linear velocity of slider. 2-phase incremental optical rotary encoder has 600 pulse per revolution (PPR). Tri-axial accelerometer has range of $-16g$ to $16g$, with a sampling rate of 8kHz.

3.2 3D Modelling

3.2.1 Preliminary Design

Figure 3.1 is the preliminary design of experimental setup, where it has a flat plat with slot in it to provide a clearance for rotating links. Links are build with link lengths obtained from kinematic and dynamic analysis. This design is very basic as it not having any details about actuator and sensors.

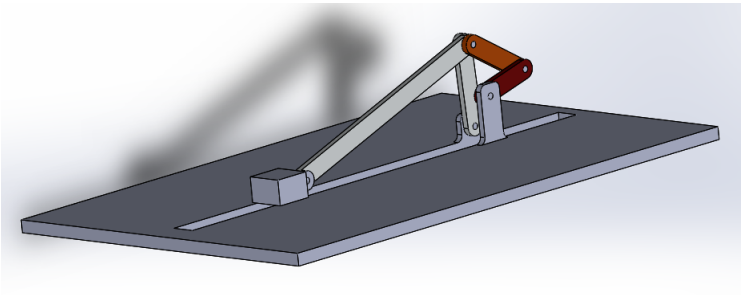


Figure 3.1: Preliminary CAD model of the mechanism

3.2.2 Second Design

Figure 3.2 is the second design for experimental setup. Where legs have been added to keep the experimental setup above the ground and to provide the clearance for rotating links.

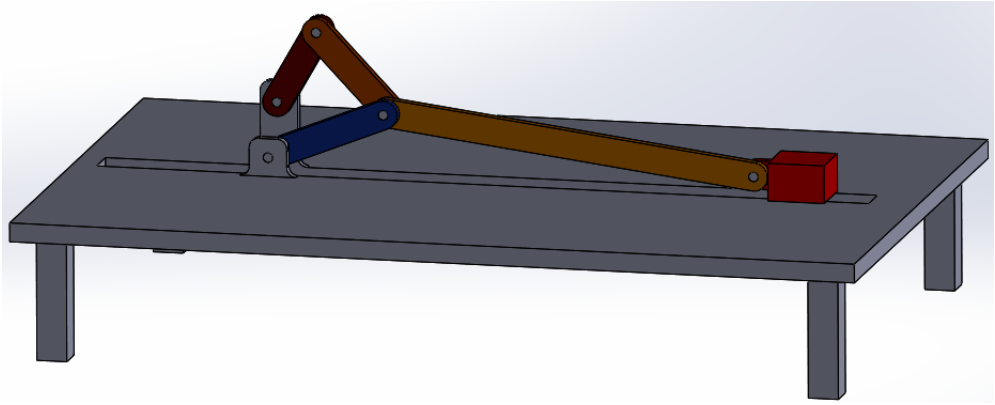


Figure 3.2: CAD Model of the experiment on (table)

3.2.3 Final Design

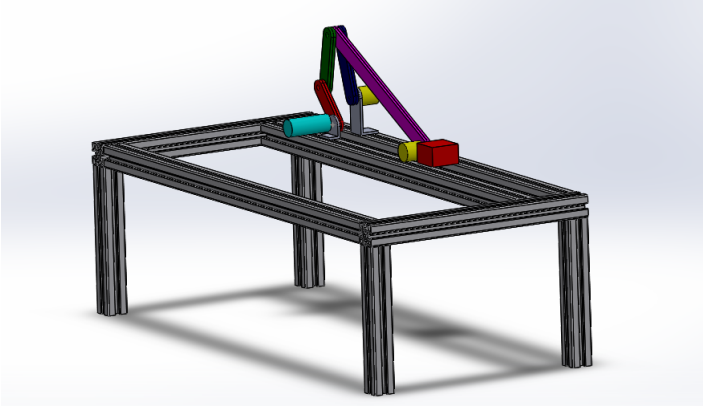


Figure 3.3: Final CAD Model of the experiment

Figure 3.3 is the final design in which is much better in terms of fabrication and operation. This design has an extruded aluminium to make the experimental setup light weight. Actuator, sensors, bearing are included in this design to make it more practical. In this design, links have a thickness of 12mm which is made with a two identical aluminium sheets of 6mm thickness welded to each other.

3.3 Manufacturing plan

This section focuses on the materials and fabrication processes that will be used for carrying out the manufacturing in the second phase of the project. Figure 3.4 gives complete insight on the various stages involved in manufacturing the experiment design.

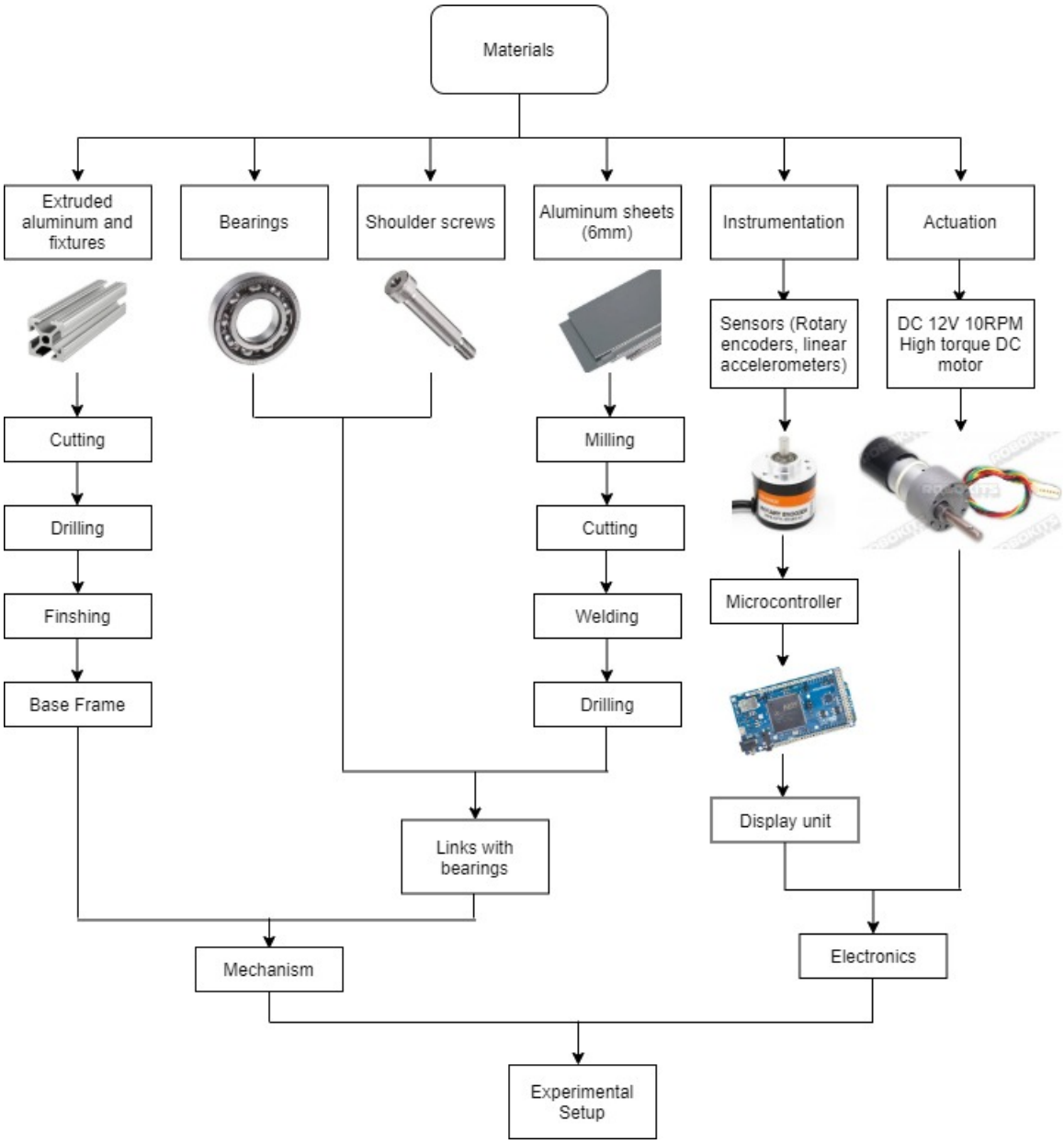


Figure 3.4: Flowchart showing manufacturing

Extruded Aluminum rods are cut into required length and assembled them to make a base frame for experimental setup as shown in Figure: 3.5. Aluminum plated with thickness

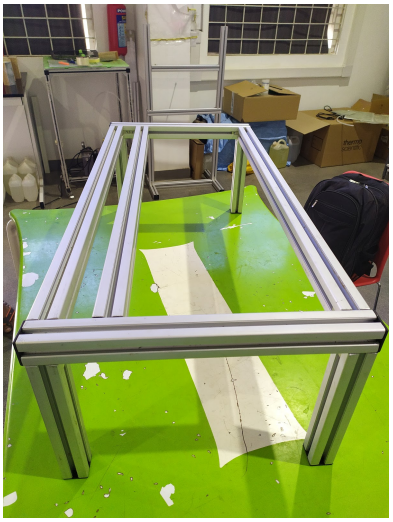


Figure 3.5: Fabricated Base frame

of 6mm is used to make links. To make 12mm thickness links, two flats are cut into the link shape by using jigsaw and then were welded together. Holes need to be drilled to

Figure 3.6: Fabricated links

insert bearing into links. Insert bearing into the link to decrease friction between link joints. Shoulder screw are used to joint the links. Electronics connection of actuator and sensor to micro controller has been completed, These actuators and sensors can be attached to mechanism to complete the experimental setup.

CHAPTER 4

Instrumentation and GUI

In this chapter we present our work on sensor integration and design of Graphical User Interface (GUI).

4.1 Instrumentation

This section provides detailed insight into sensor and actuator configuration for the experimental setup. The demo can be viewed at https://youtu.be/741drZK-4sk. The setup mainly includes three parts i.e. sensors, actuator, microcontroller and Raspberry Pi computer. The sensor setup includes three rotary encoders (only one shown in demo) for measuring the angular position of the link and one tri-axial accelerometer for measuring the acceleration of the slider. The sensors are connected to the Raspberry Pi via a microcontroller namely Arduino Uno board.

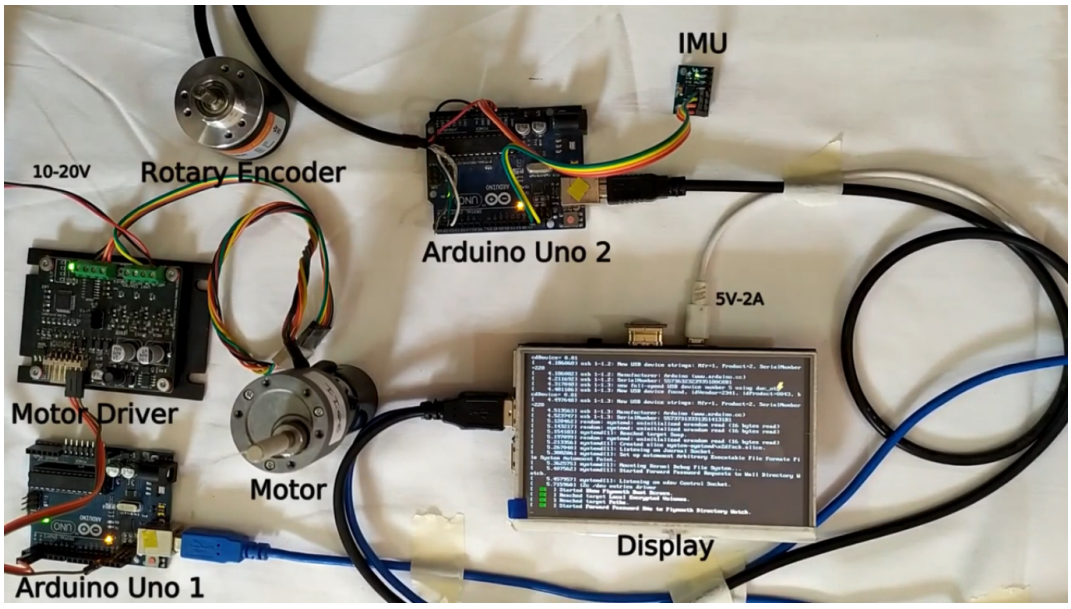


Figure 4.1: Instrumentation setup and integration

The actuation setup includes a high torque low RPM DC motor with inbuilt quadencoders for digital position control. The motor is connected to a motor controller

(RMCS2303) for performing control of the motor drive and delivering the required power in controlled manner. The motor controller is connected to another Arduino Uno and they both communicate via serial communication. The Arduino board is connected to the Raspberry Pi computer which can send the required speed command updates to the board. The complete setup can be seen in Fig. 4.1. The Raspberry Pi operates on a Debian based operating system(OS) known as Raspbian, the OS has been heavily modified to have all the required packages preinstalled and ready to use (available for download at project website). The software stack mainly makes use of Bash, Arduino-CLI, YAD and Python; the implementation details have been omitted for brevity, however some important portion of the code can be found in appendix. Fig. 4.2 show the 5-inch touchscreen display, HDMI connector and the Raspberry Pi display. The display comes with a stylus and can be easily operated by the user.

Figure 4.2: A closer look at the display and the Raspberry Pi

Once the power for the Raspberry Pi is turned on it boots up quickly to the screen as

shown in Fig. 4.3. Also when connected to the same network as of user the complete setup can be accessed and controlled remotely. The OpenKDM application can started by double-clicking the "OpenKDM.sh" file on the desktop. Once launched we can perform mainly operation namely setting the motor speed, recording the available sensor data and retrieving the recorded data as shown in Fig. 4.4. The GUI for the different operations can be seen in Fig. 4.5, 4.6 and 4.7.

Figure 4.3: Home screen after boot up

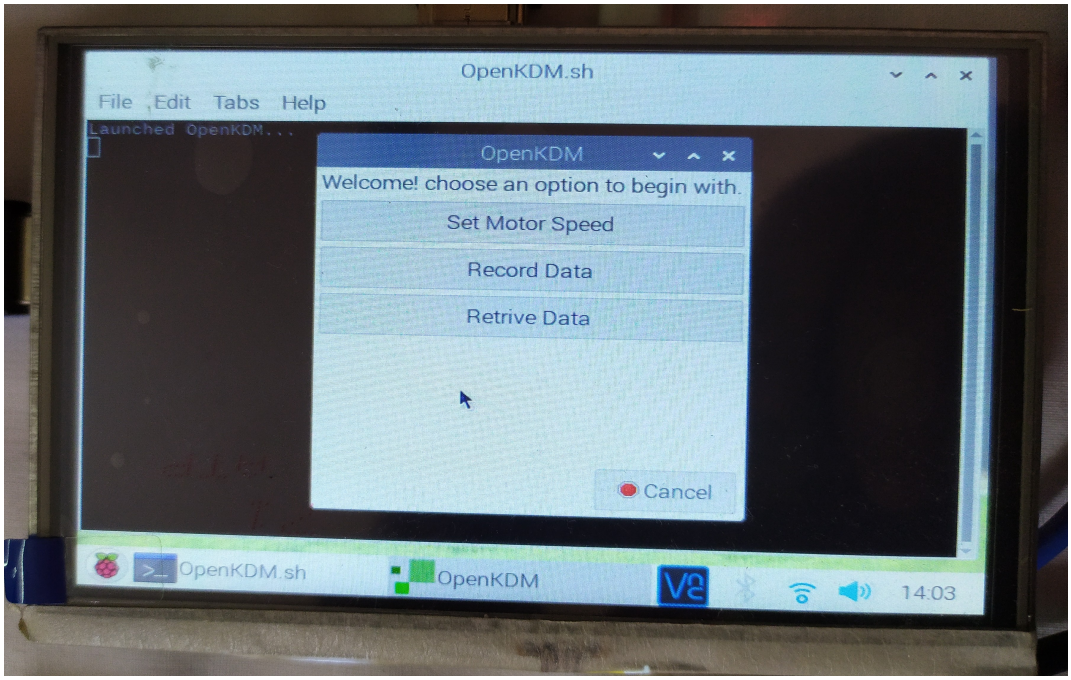


Figure 4.4: Main screen after launching the application

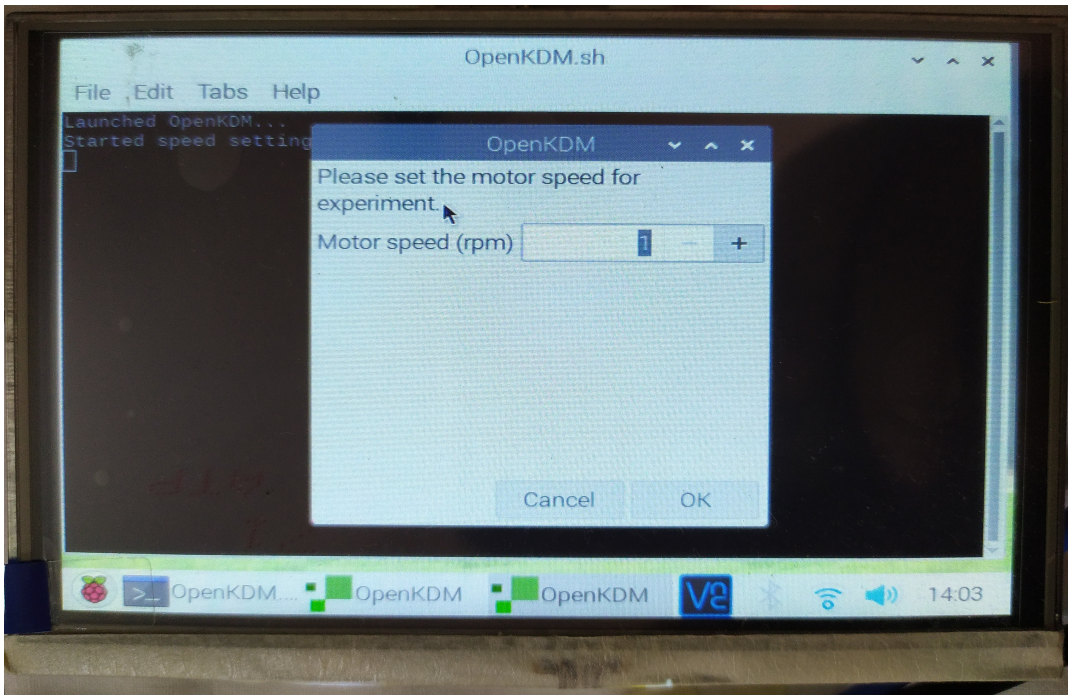


Figure 4.5: Speed setting screen

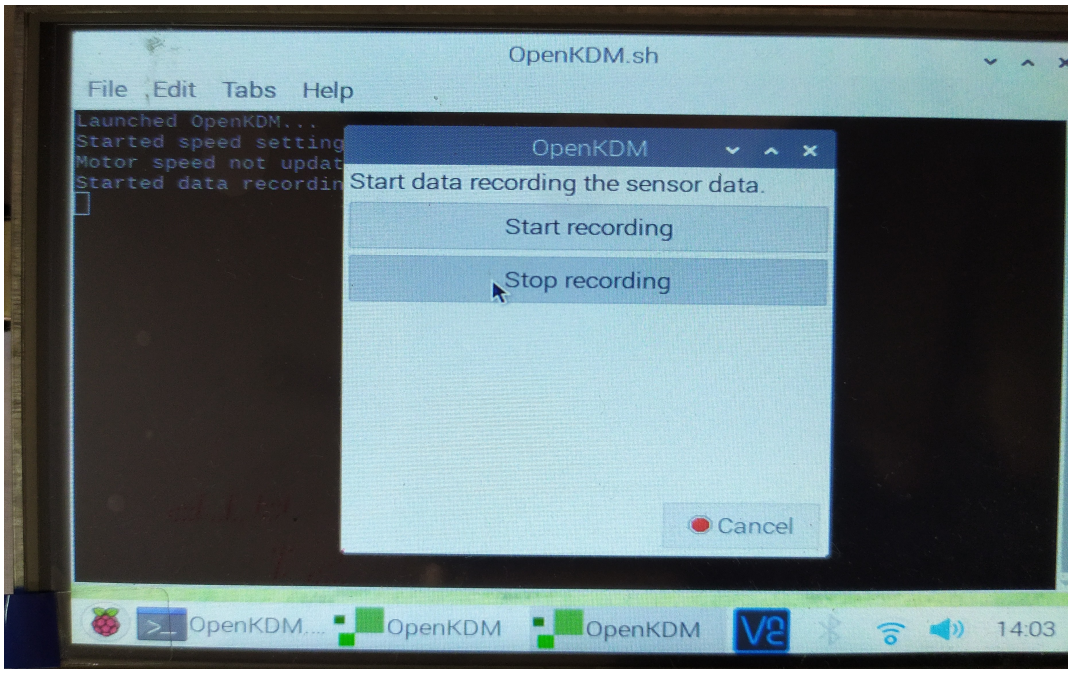


Figure 4.6: Data recording screen

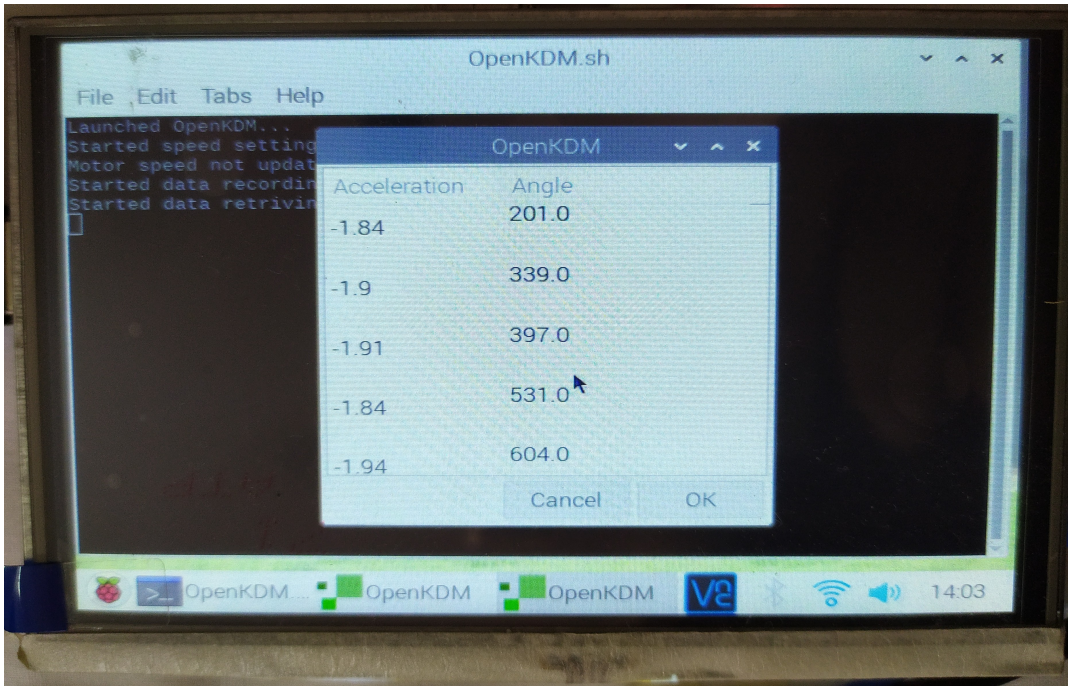


Figure 4.7: Data retrival screen

4.2 Software design

This section provides details of the OpenKDM software. The demo can be viewed at https://youtu.be/1nqF1lyesHM. This part of the work has been developed using Python and PyQt5.

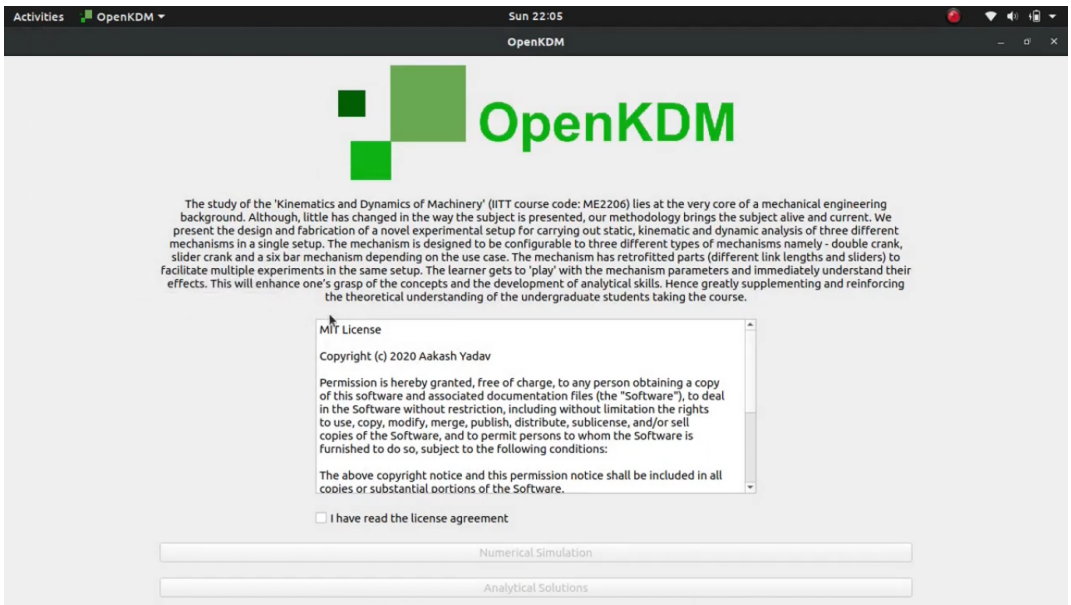


Figure 4.8: Main screen for the software package

The main screen for the software can be seen in Fig. 4.8 and provides two options:

Numerical solution and analytical solution which can be accessed by pressing the buttons at the bottom. The GUI for these two options can be seen in Fig. 4.9 and 4.10. The user can choose from a myriad of options to perform the simulation and obtain the results.

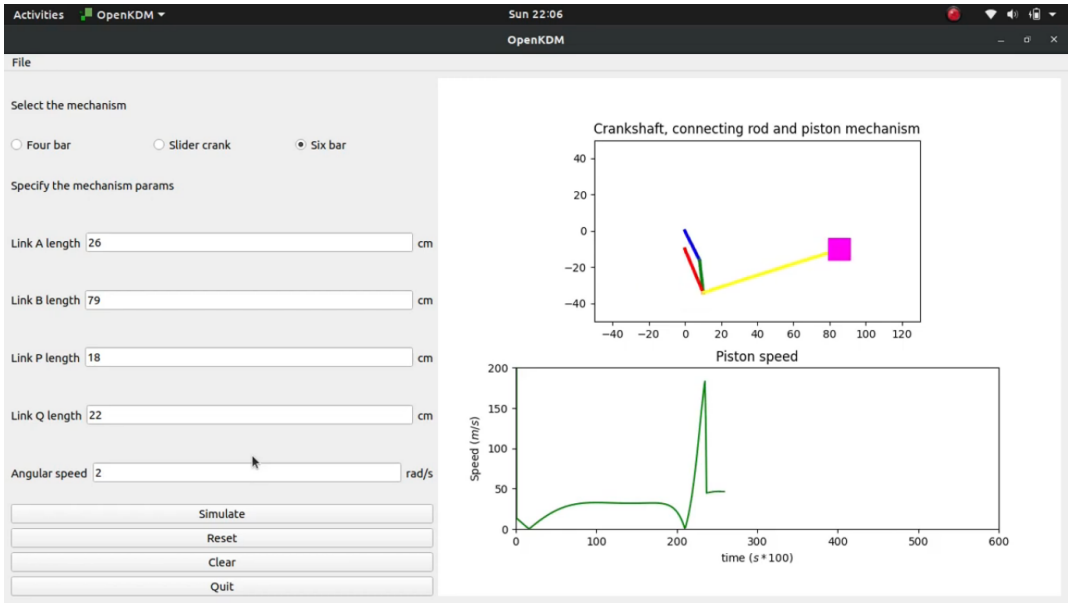


Figure 4.9: Numerical solutions

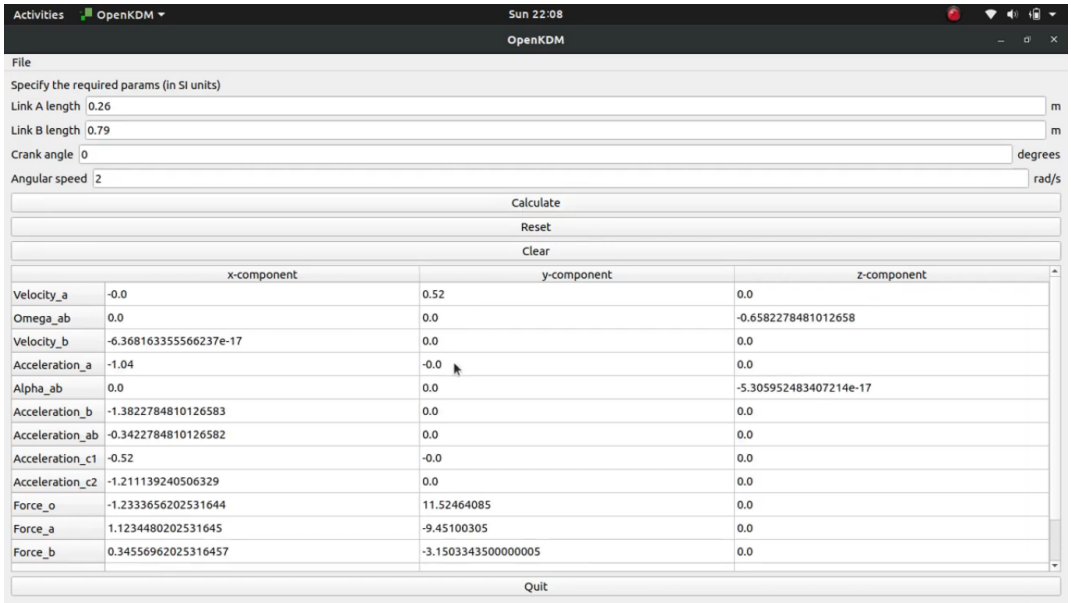


Figure 4.10: Analytical solutions

Generally its takes more time to compute velocities, accelerations, forces of the link in a mechanism on a paper. By using this software one can easily get those in very less time. This software had a option to give the input parameters, which allows to appreciate the effect of link length and crank velocity on link velocities, accelerations and forces.

CHAPTER 5

RESULTS

This final section presents obtained results. By utilizing the kinematic analysis we obtained the link lengths resulting in constant velocity mechanism as:

Link	Link length (cm)
OA	13
CD	9
AC	11
AB	39.5
DO	5

Table 5.1: Final link lengths

Furthermore plots for slider velocity and acceleration were obtained using Simscape.

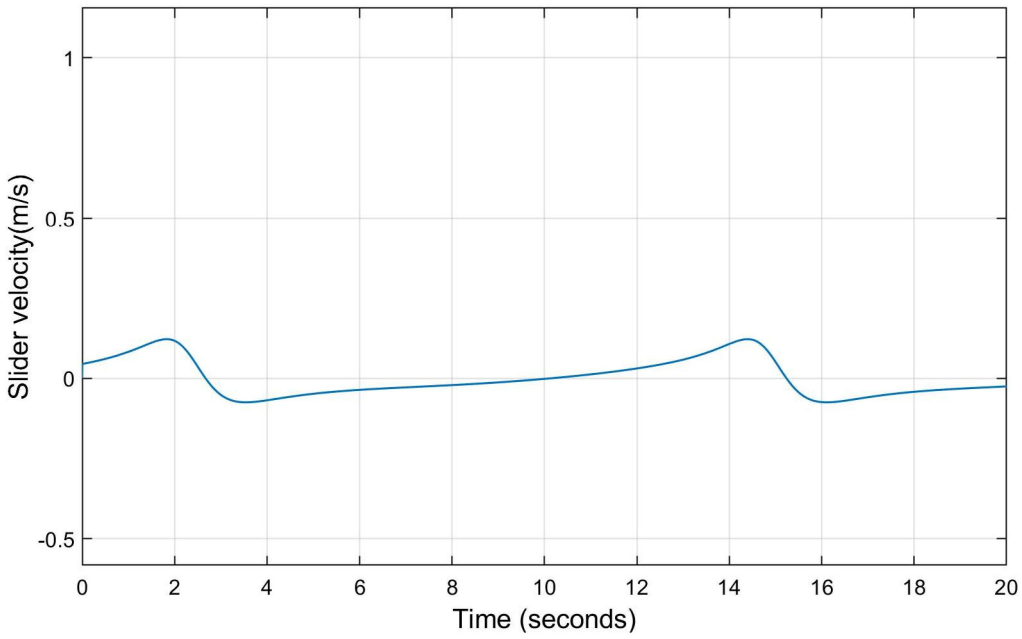


Figure 5.1: Slider Velocity

We have even created a software or a tool to get the analytical results of angular velocity, angular acceleration, slider velocity, slider acceleration, Link reaction forces for a any given link lengths and crank position with its speed as input.

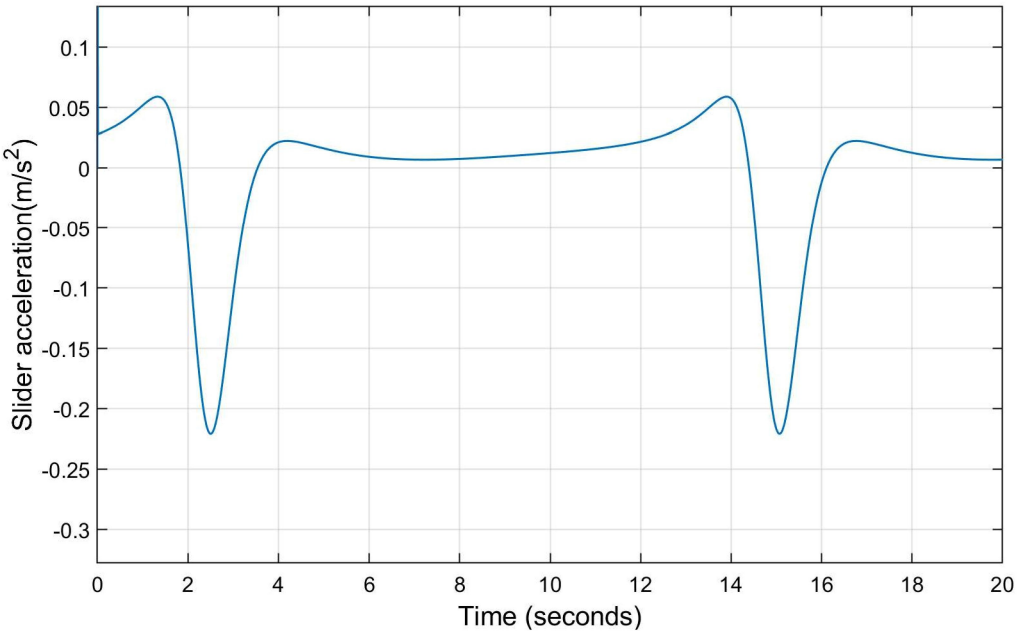


Figure 5.2: Slider acceleration

Coming to experimental setup we have completed entire instrumentation as shown in figure: 4.1 and base frame structure along with fabricated links as shown in figures: 3.5 and figure: 3.6.

CHAPTER 6

CONCLUSION AND FUTURE WORK

The complete analysis of the mechanism(s) was performed using applicable method and software. 3D CAD model of the mechanism was created to facilitate better visualisation. Various design considerations were addressed and manufacturing workflow and plan were accomplished as a prerequisite to manufacturing. Created a software to facilitated to get the velocities, acceleration and forces using it by using the analytical methods. Instrumentation for experimental setup has been completed along with its interface to get the sensors results.

In the future work left is to assembling of the base frame with links and connection of instrumentation.

APPENDIX A

Lab Manual

KINEMATIC AND DYNAMIC ANALYSIS OF MECHANISM

1 Aim

- Kinematic analysis of Slider crank, Four bar, Six bar mechanism.
- Dynamic analysis of Slider crank, Four bar, Six bar mechanism.

2 Apparatus Required

Experimental setup.

3 Theory

3.1 Kinematic analysis

3.1.1 Slider crank mechanism

The equations of motion pertaining to the slider crank mechanism (Figure 1) are given below.

$$\vec{V}_{B/A} = \vec{V}_B - \vec{V}_A \tag{1}$$

For link OA we have

$$\vec{V}_{A/O} = \vec{\omega}_{AO} \times \vec{r}_{A/O} = \vec{V}_A \tag{2}$$

Since 'O' is ground link we have $\vec{V}_O = 0 \implies \vec{V}_{A/O} = \vec{V}_A - \vec{V}_O = \vec{V}_A$. As point B lies on the slider and is moving leftwards it will have velocity along the negative x-direction i.e. $\vec{V}_B = -V_B\hat{i}$; because $\vec{\omega}_{AO} = \omega_{AO}\hat{k}$. For link AB we have

$$\vec{V}_{B/A} = \vec{\omega}_{BA} \times \vec{r}_{B/A} \tag{3}$$

Substituting equation 3 in equation 1 we obtain

$$\vec{\omega}_{BA} \times \vec{r}_{B/A} = \vec{V}_B - \vec{V}_A \tag{4}$$

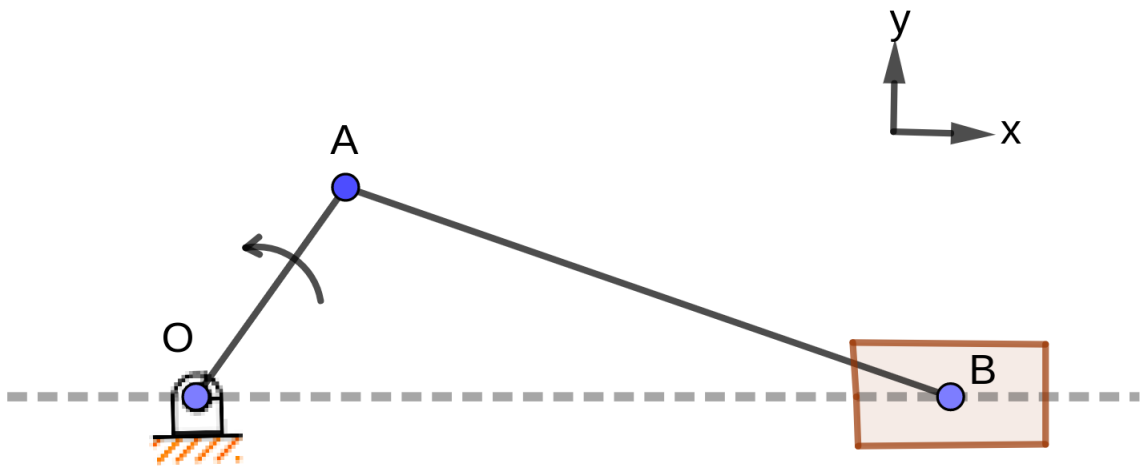


Figure 1: Slider crank mechanism

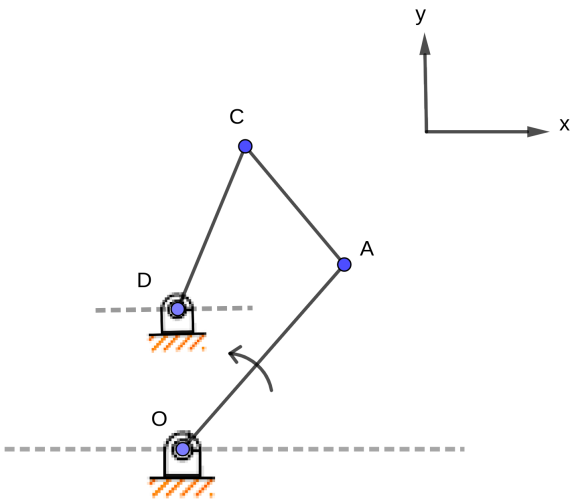


Figure 2: Four bar mechanism

Using equation 4 we can find ω_{BA} and V_B. For finding the accelerations we can write

$$\vec{a}_{A/O} = \vec{a}_A = \vec{a}_A^t + \vec{a}_A^n \tag{5a}$$

$$\vec{a}_A^t = \alpha_{\vec{A/O}} \times r_{\vec{A/O}} \tag{5b}$$

$$\vec{a}_A^n = \omega_{\vec{AO}} \times \omega_{\vec{AO}} \times r_{\vec{A/O}} \tag{5c}$$

where, $\vec{a}_A^t$ and $\vec{a}_A^n$ denote the tangential and normal acceleration of point A. As point B lies on the slider, we have $\vec{a}_B = a_B\hat{i}$.

$$\vec{a}_{B/A} = \vec{a}_{B/A}^t + \vec{a}_{B/A}^n = \vec{a}_B - \vec{a}_A \tag{6}$$

Using the above equation we can find α_{BA} and $\vec{a}_B$.

3.1.2 Four bar mechanism (with four revolute joints)

The mathematical equations pertaining to the four bar mechanism (Figure 2) are given below.

$$\vec{V}_{A/O} = \vec{\omega}_{AO} \times \vec{r}_{A/O} = \vec{V}_A \tag{7a}$$

$$\vec{V}_{C/D} = \vec{\omega}_{CD} \times \vec{r}_{C/D} = \vec{V}_C \tag{7b}$$

$$\vec{V}_{A/C} = \vec{\omega}_{AC} \times \vec{r}_{A/C} = \vec{V}_A - \vec{V}_C = \vec{\omega}_{AO} \times \vec{r}_{A/O} - \vec{\omega}_{CD} \times \vec{r}_{C/D} \tag{8}$$

Using the above equation we can find ω_{CD} and ω_{AC}.

$$\vec{a}_{A/O} = \vec{a}_A = \vec{a}_A^t + \vec{a}_A^n \tag{9a}$$

$$\vec{a}_{C/D} = \vec{a}_C = \vec{a}_C^t + \vec{a}_C^n \tag{9b}$$

$$\vec{a}_{A/C} = \vec{a}_{A/C}^t + \vec{a}_{A/C}^n = \vec{a}_A - \vec{a}_C \tag{10}$$

Here $\vec{a}^t = \vec{\alpha} \times \vec{r}$ and $\vec{a}^n = \vec{\omega} \times \vec{\omega} \times \vec{r}$. Using the above equations we can find α_{AC} and α_{CD}

3.1.3 Six bar mechanism

The mathematical equations associated with the six bar mechanism (Figure 3) are given below.

$$\vec{V}_{C/D} = \vec{\omega}_{CD} \times \vec{r}_{C/D} = \vec{V}_C \tag{11}$$

$$\vec{V}_{A/C} = \vec{\omega}_{AC} \times \vec{r}_{A/C} = \vec{V}_A - \vec{V}_C = \vec{\omega}_{AO} \times \vec{r}_{A/O} - \vec{\omega}_{CD} \times \vec{r}_{C/D} \tag{12}$$

Using the above equation we can find ω_{OA} and ω_{AC}.

$$\vec{V}_{B/A} = \vec{V}_B - \vec{V}_A \tag{13}$$

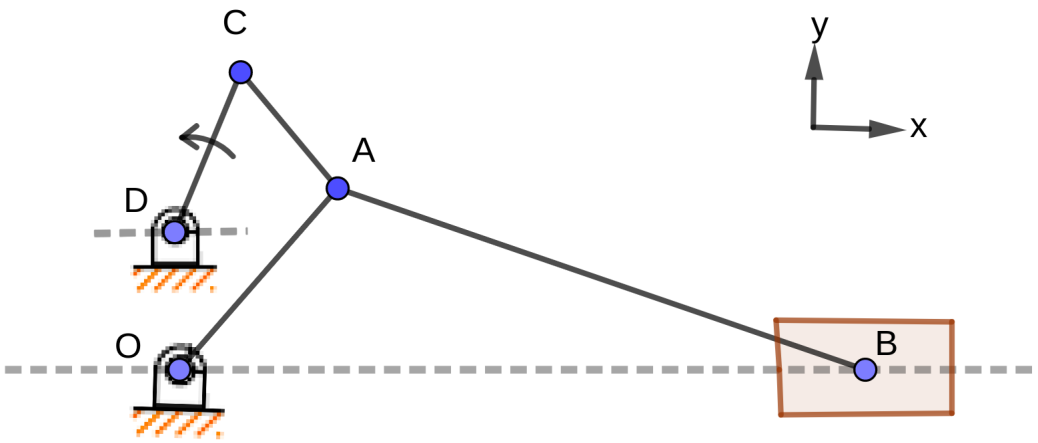


Figure 3: Six bar mechanism

For link OA we have

$$\vec{V}_{A/O} = \vec{\omega}_{AO} \times \vec{r}_{A/O} = \vec{V}_A \tag{14}$$

Since 'O' is ground link we have $\vec{V}_O = 0 \implies \vec{V}_{A/O} = \vec{V}_A - \vec{V}_O = \vec{V}_A$. As point B lies on the slider and is moving leftwards it will have velocity along the negative x-direction i.e. $\vec{V}_B = -V_B\hat{i}$; because $\vec{\omega}_{AO} = \omega_{AO}\hat{k}$. For link AB we have

$$\vec{V}_{B/A} = \vec{\omega}_{BA} \times \vec{r}_{B/A} \tag{15}$$

Substituting equation 15 in equation 13 we obtain

$$\vec{\omega}_{BA} \times \vec{r}_{B/A} = \vec{V}_B - \vec{V}_A \tag{16}$$

Using equation 16 we can find ω_{BA} and V_B.

For finding the accelerations we can write

$$\vec{a}_{C/D} = \vec{a}_C = \vec{a}_C^t + \vec{a}_C^n \tag{17}$$

$$\vec{a}_{A/C} = \vec{a}_{A/C}^t + \vec{a}_{A/C}^n = \vec{a}_A - \vec{a}_C \tag{18}$$

Here $\vec{a}^t = \vec{\alpha} \times \vec{r}$ and $\vec{a}^n = \vec{\omega} \times \vec{\omega} \times \vec{r}$. Using the above equations we can find α_{AC} and α_{OA}

$$\vec{a}_{A/O} = \vec{a}_A = \vec{a}_A^t + \vec{a}_A^n \tag{19a}$$

$$\vec{a}_A^t = \alpha_{\vec{A/O}} \times r_{\vec{A/O}} \tag{19b}$$

$$\vec{a}_A^n = \omega_{\vec{AO}} \times \omega_{\vec{AO}} \times r_{\vec{A/O}} \tag{19c}$$

where, $\vec{a}_A^t$ and $\vec{a}_A^n$ denote the tangential and normal acceleration of point A. As point B lies on the slider, we have $\vec{a}_B = a_B\hat{i}$.

$$\vec{a}_{B/A} = \vec{a}_{B/A}^t + \vec{a}_{B/A}^n = \vec{a}_B - \vec{a}_A \tag{20}$$

Using the above equation we can find α_{BA} and $\vec{a}_B$.

3.2 Dynamic analysis

3.2.1 Slider crank mechanism

Force and Torque balance for link OA using Figure 4

$$F_{OX} + F_{AX} + m_1 a_{C_1 x} = 0 \tag{21a}$$

$$F_{OY} + F_{AY} = m_1 g + m_1 a_{C_1 y} \tag{21b}$$

$$I_{AO}\vec{\alpha}_{AO} + \vec{r}_{C_1/O} \times m_1 \vec{a}_{C_1} = \vec{r}_{C_1 O} \times m_1 g(-\hat{j}) + \vec{r}_{A/O} \times \vec{F}_A + \vec{T} \tag{21c}$$

where, $\vec{F}_A = \vec{F}_{AX}\hat{i} + \vec{F}_{AY}\hat{j}$. Similarly for link AB we have

$$F_{AX} = F_{BX} + m_2 a_{C_2 x} \tag{22a}$$

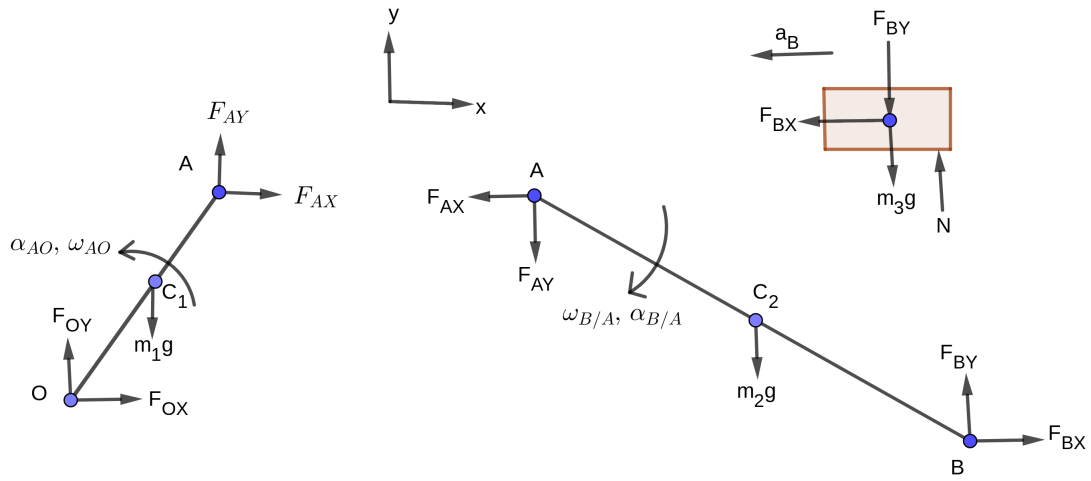


Figure 4: Free Body Diagram for Slider Crank Mechanism

$$F_{BY} + m_2 a_{C_2 y} = F_{AY} + m_2 g \tag{22b}$$

$$I_{BA}\vec{\alpha}_{B/A} + \vec{r}_{C_2/A} \times m_2 \vec{a}_{C_2} = \vec{r}_{C_2/A} \times m_2 g(-\hat{j}) + \vec{r}_{B/A} \times \vec{F}_B \tag{22c}$$

where, $\vec{F}_B = \vec{F}_{BX}\hat{i} + \vec{F}_{BY}\hat{j}$.
Force balance for the slider

$$F_{BY} + m_3 g = N \tag{23a}$$

$$m_3 a_B = F_{BX} \tag{23b}$$

3.2.2 Four bar mechanism (with four revolute joints)

Force and Torque balance for link OA using Figure 5

$$\sum \vec{F_X} = 0 \tag{24a}$$

$$\sum \vec{F_Y} = 0 \tag{24b}$$

$$\sum \vec{T} = I_1 \vec{\alpha}_{AO} \tag{24c}$$

Force and Torque balance for link AC using Figure 5

$$\sum \vec{F_X} = 0 \tag{25a}$$

$$\sum \vec{F_Y} = 0 \tag{25b}$$

$$\sum \vec{T} = I_4 \vec{\alpha}_{AC} \tag{25c}$$

Force and Torque balance for link CD using Figure 5

$$\sum \vec{F_X} = 0 \tag{26a}$$

$$\sum \vec{F_Y} = 0 \tag{26b}$$

$$\sum \vec{T} = I_5 \vec{\alpha}_{CD} \tag{26c}$$

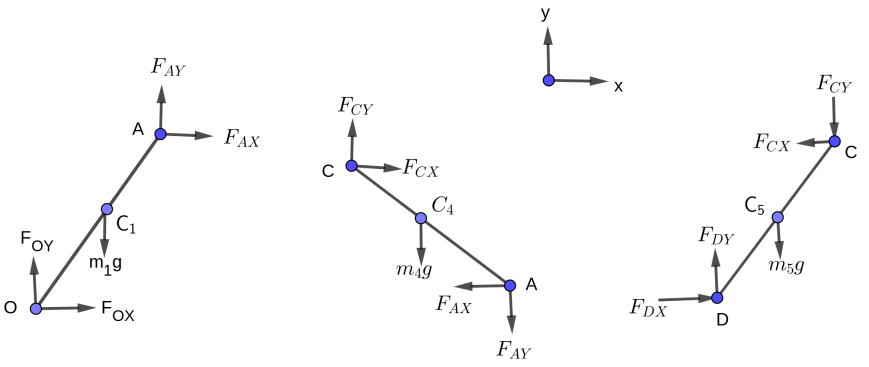


Figure 5: Free Body Diagram for four-bar mechanism

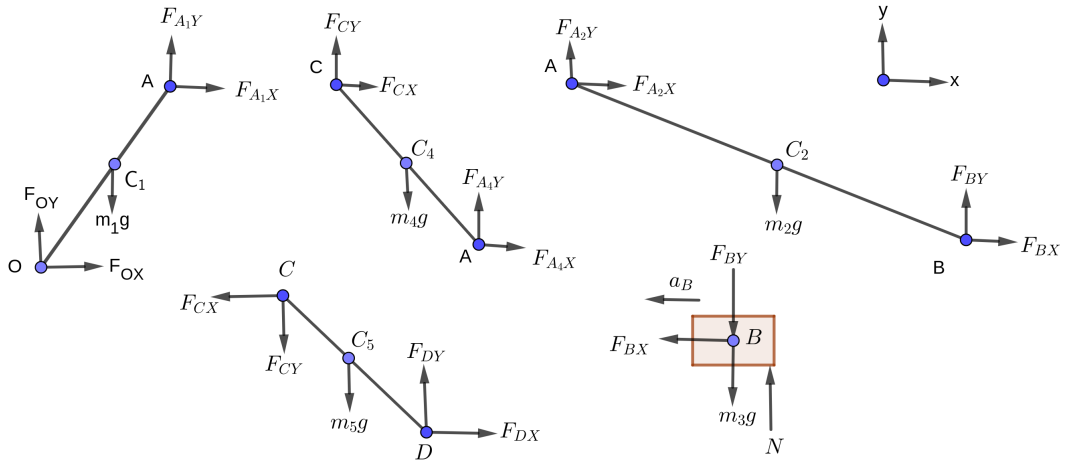


Figure 6: Free Body Diagram for six-bar mechanism

3.2.3 Six bar mechanism

Force and Torque balance for all links using Figure 6

$$\sum \vec{F_X} = 0, \sum \vec{F_Y} = 0, \sum \vec{T} = I\vec{\alpha} \tag{27}$$

$$F_{A_1X} + F_{A_2X} + F_{A_4X} = 0 \tag{28a}$$

$$F_{A_1Y} + F_{A_2Y} + F_{A_4Y} = 0 \tag{28b}$$

Force balance for Slider using Figure 6

$$F_{BY} + m_3g = N \tag{29a}$$

$$m_3a_B = F_{BX} \tag{29b}$$

4 Experimental Procedure

- Turn on the experimental setup.
- Run the mechanism for some time to get the data about positions, Velocities, Accelerations of links and sliders.
- Turn off the actuator.
- Now give the angular position of crank as input in the interface provided with setup to get the experimental data for that particular instance.

5 Observation

Compare the experimental reading with theoretical data calculated from formulas.

	Link Position		Link velocity		Link acceleration	
Link Name	Theoretical	Experimental	Theoretical	Experimental	Theoretical	Experimental

Table 1: Kinematic analysis

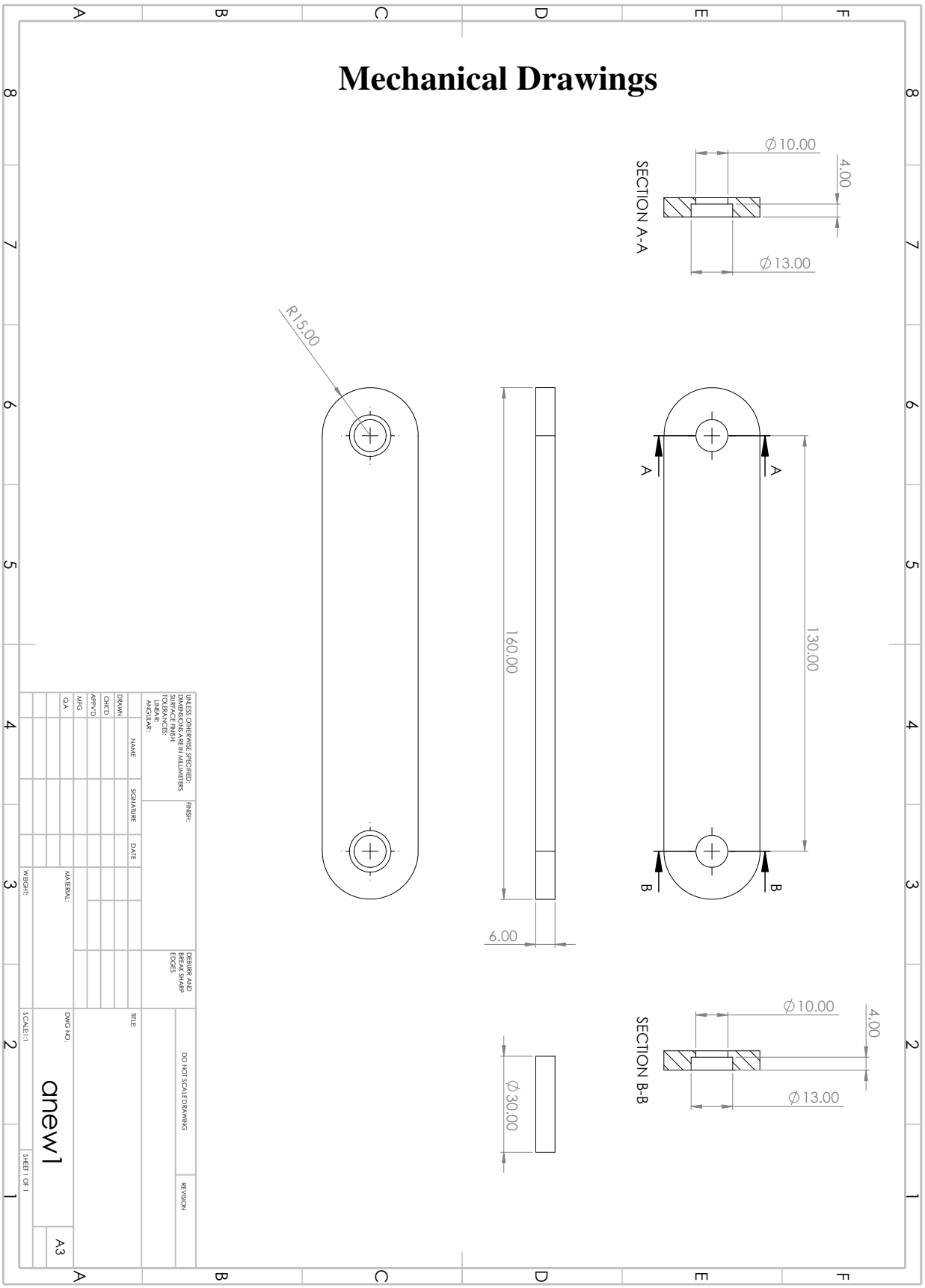
Joint	Theoretical F_x	Experimental F_x	Theoretical F_y	Experimental F_y

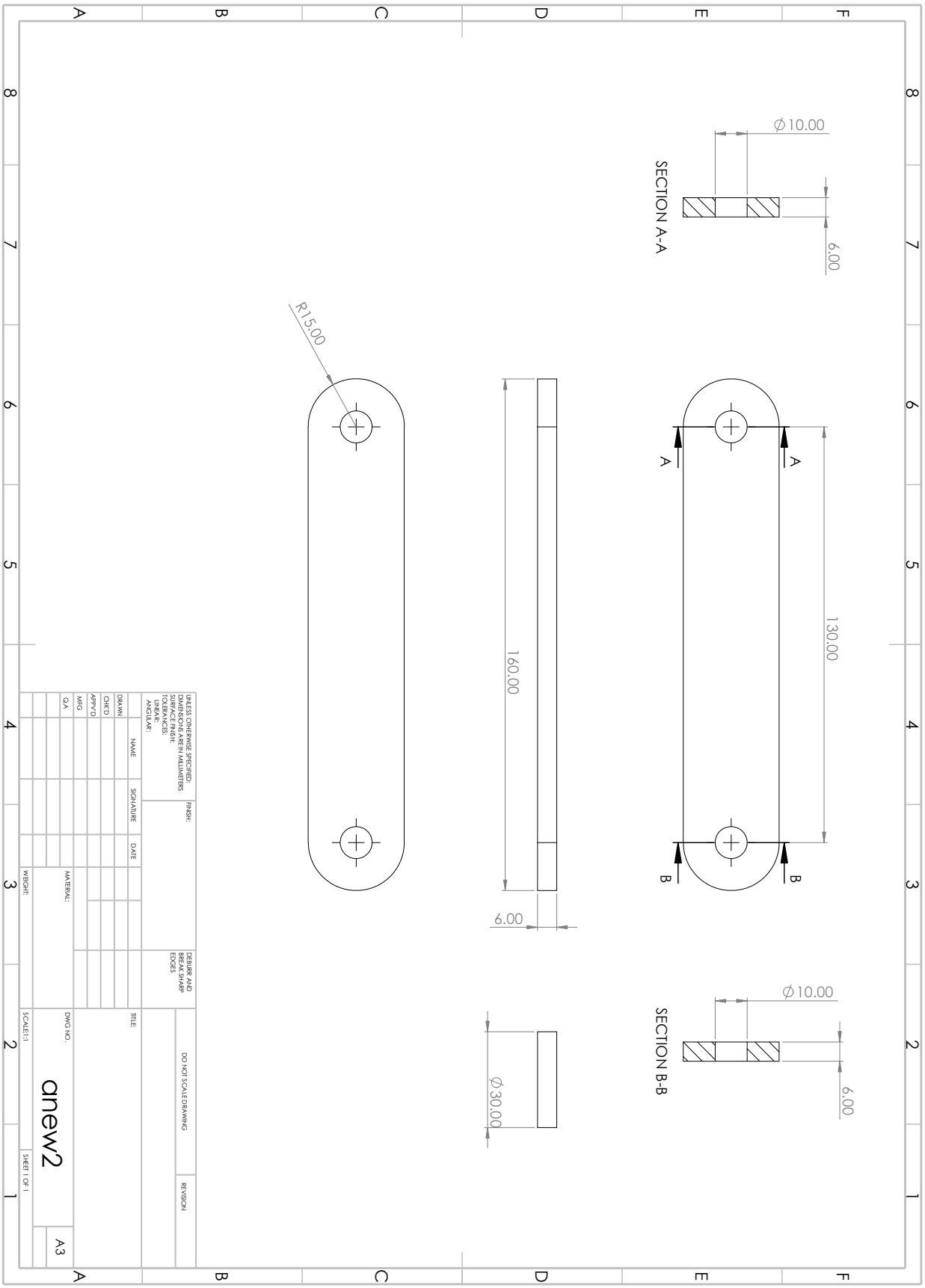
Table 2: Dynamic analysis

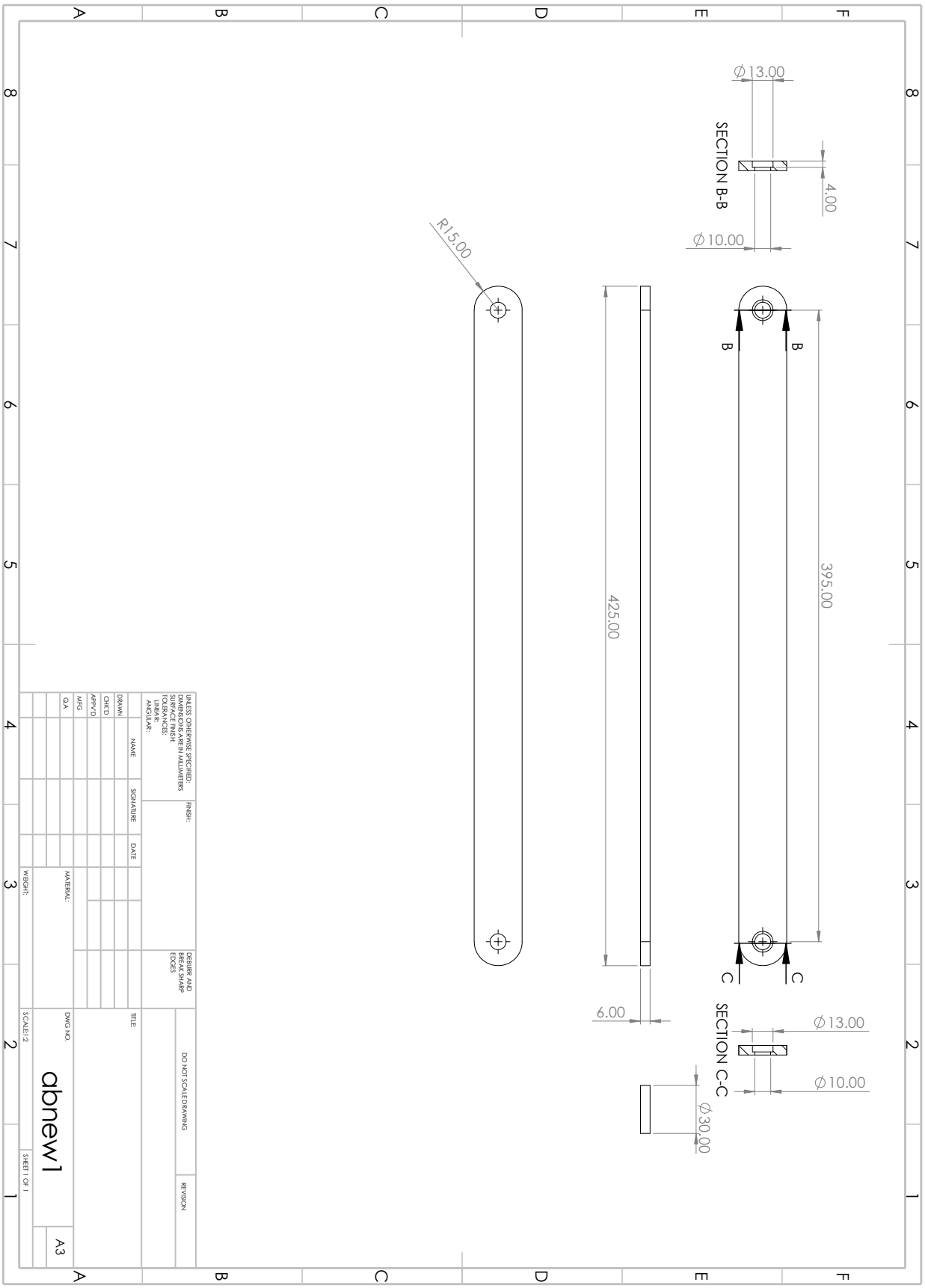
6 Results

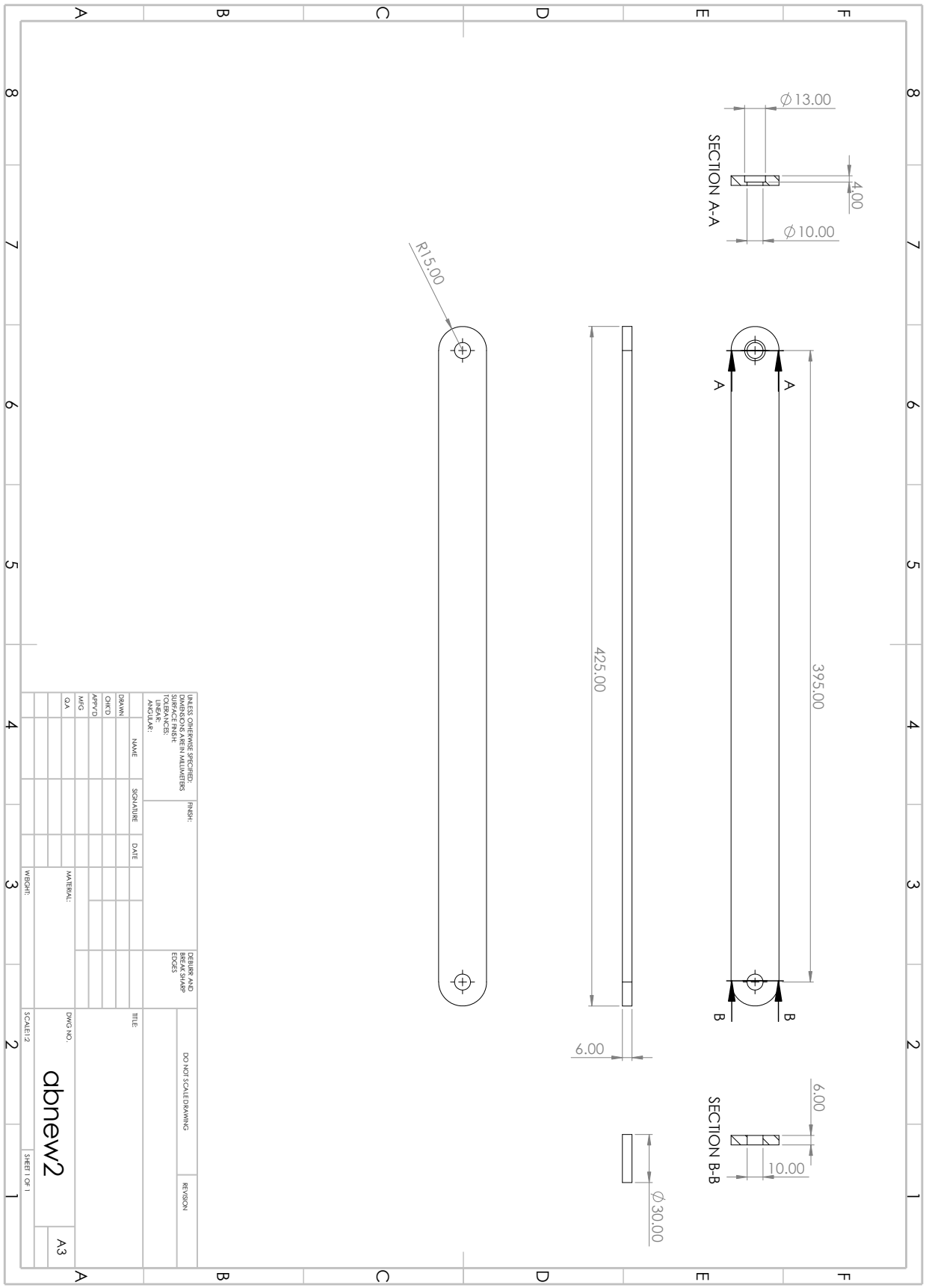
What is the error between experimental reading and theoretical values?

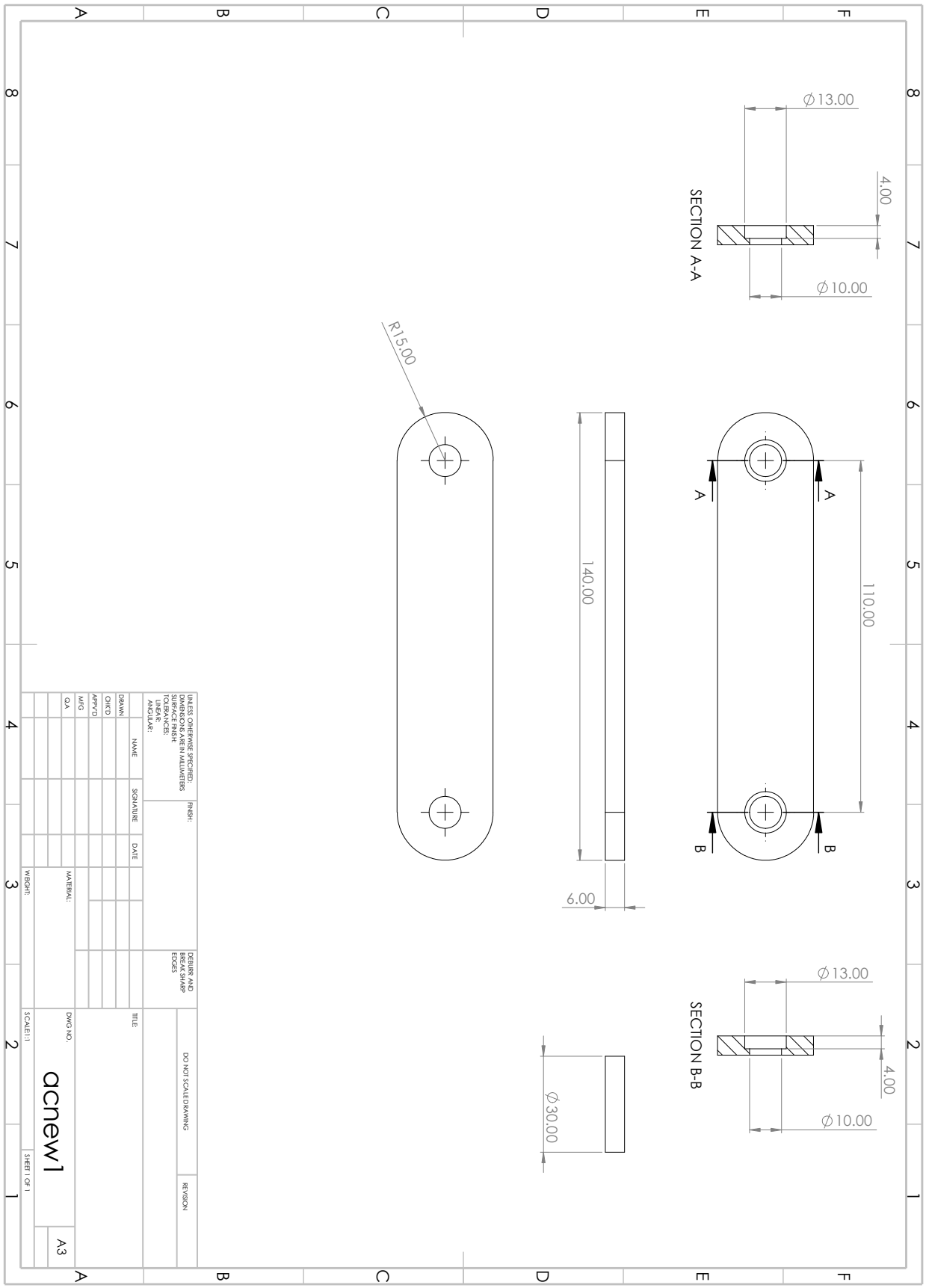
APPENDIX B

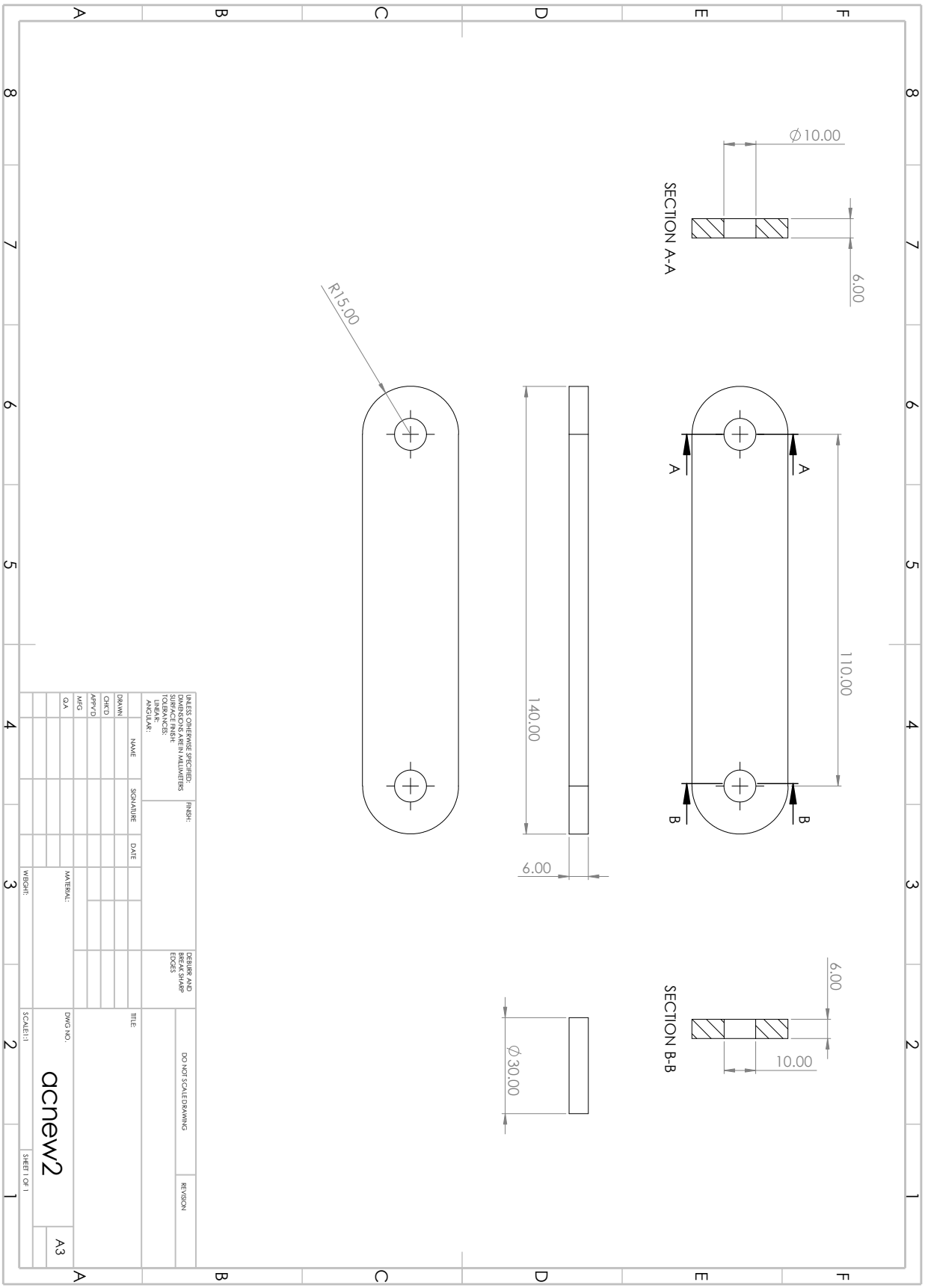

Mechanical Drawings
SECTION A-A
∅10.00
4.00
∅13.00
R15.00
160.00
6.00
130.00
A
A
B
B
SECTION B-B
∅10.00
4.00
∅13.00
∅30.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
FINISH:
DEBURR AND BREAK SHARP EDGES
MATERIAL:
WEIGHT:
TITLE:
DWG NO.
anew1
A3
SCALE:1:1
SHEET 1 OF 1
DO NOT SCALE DRAWING
REVISION

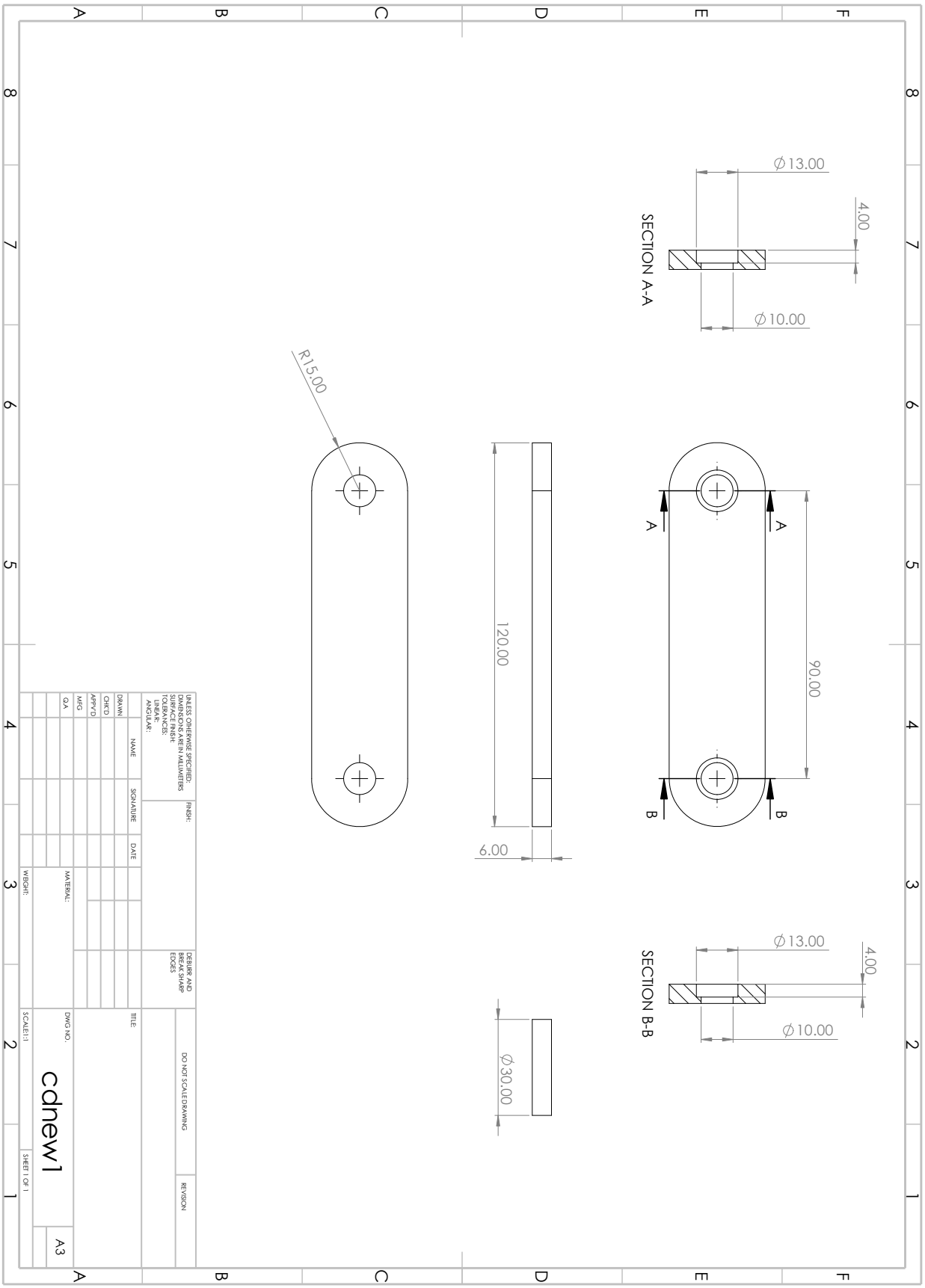

SECTION A-A
∅10.00
6.00
R15.00
160.00
6.00
∅30.00
A
A
130.00
B
B
SECTION B-B
∅10.00
6.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
FINISH:
DEBURR AND BREAK SHARP EDGES
MATERIAL:
WEIGHT:
DWG NO.
anew2
TITLE:
DO NOT SCALE DRAWING
REVISION
SCALE:1:1
A3
SHEET 1 OF 1

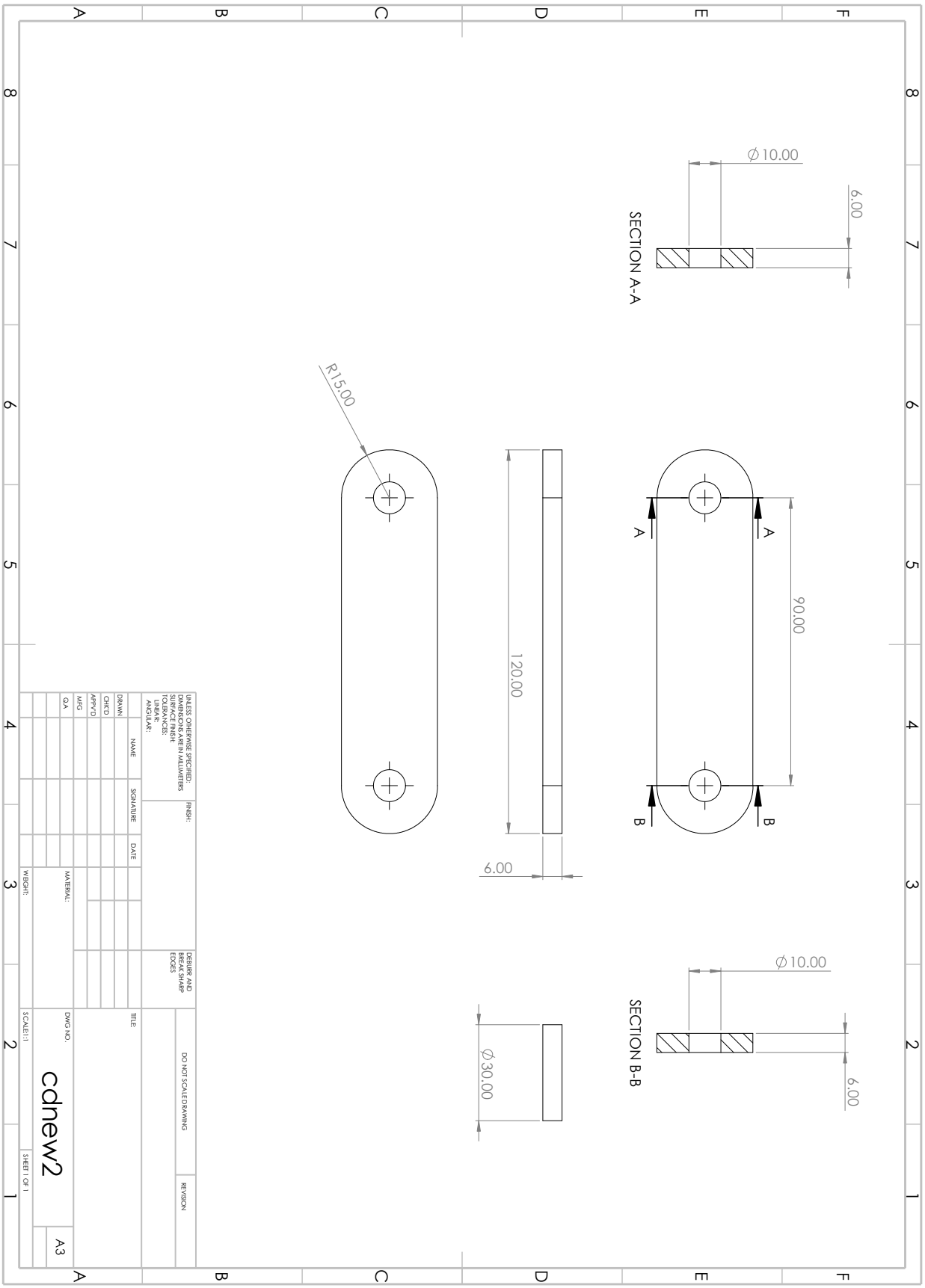

∅13.00
SECTION B-B
4.00
∅10.00
R15.00
B
B
395.00
425.00
C
C
6.00
∅13.00
SECTION C-C
∅10.00
∅30.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
FINISH:
DEBURR AND BREAK SHARP EDGES
MATERIAL:
WEIGHT:
TITLE:
DWG NO.
abnew1
A3
SCALE:1:2
SHEET 1 OF 1
DO NOT SCALE DRAWING
REVISION


∅13.00
4.00
∅10.00
SECTION A-A
R15.00
425.00
395.00
6.00
A
A
B
B
6.00
10.00
SECTION B-B
∅30.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
FINISH:
DEBURR AND BREAK SHARP EDGES
MATERIAL:
WEIGHT:
SCALE:1:2
DWG NO.
abnew2
A3
TITLE:
DO NOT SCALE DRAWING
REVISION
SHEET 1 OF 1


SECTION A-A
∅13.00
4.00
∅10.00
R15.00
140.00
6.00
110.00
A
A
B
B
SECTION B-B
∅13.00
4.00
∅10.00
∅30.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
FINISH:
DEBURR AND BREAK SHARP EDGES
MATERIAL:
WEIGHT:
TITLE:
DWG NO.
acnew1
A3
SCALE:1:1
SHEET 1 OF 1
DO NOT SCALE DRAWING
REVISION


SECTION A-A
∅10.00
6.00
R15.00
140.00
6.00
A
A
110.00
B
B
SECTION B-B
6.00
10.00
∅30.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
FINISH:
DEBURR AND BREAK SHARP EDGES
MATERIAL:
WEIGHT:
TITLE:
DWG NO.
acnew2
A3
SCALE:1:1
SHEET 1 OF 1
DO NOT SCALE DRAWING
REVISION


SECTION A-A
∅13.00
4.00
∅10.00
R15.00
120.00
6.00
A
A
90.00
B
B
SECTION B-B
∅13.00
4.00
∅10.00
∅30.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
FINISH:
DEBURR AND BREAK SHARP EDGES
MATERIAL:
WEIGHT:
SCALE:1:1
DWG NO.
cdnew1
A3
TITLE:
DO NOT SCALE DRAWING
REVISION
SHEET 1 OF 1


SECTION A-A
∅10.00
6.00
R15.00
120.00
90.00
6.00
A
A
B
B
SECTION B-B
∅10.00
6.00
∅30.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
WEIGHT:
FINISH:
DEBURR AND BREAK SHARP EDGES
TITLE:
DWG NO.
cdnew2
A3
SCALE:1:1
SHEET 1 OF 1
DO NOT SCALE DRAWING
REVISION

APPENDIX C

CODE LISTING

The code used for carrying out this work is presented here. Due due to the high volume of the code involved in this project, it is also hosted on GitHub for brevity at https://github.com/nimrobotics/OpenKDM with complete documentation.

C.1 Six-bar mechanism (Python)

```
"""
Python class for six bar mechanism. Performs speed computations at various
time steps and stores data

@nimrobotics
"""

import numpy as np
import math
import csv

# ground for link a and p
link_a_pivot = (0,-10)
link_p_pivot = (0,0)

class My_mechanism(object):
    def __init__(self,a,b,p,q,omega):
        self.a = a                      # Rod a
        self.b = b                      # Rod b
        self.p = p                      # Rod p
        self.q = q                      # Rod q
        self.omega = omega              # Angular speed of rod a in rad/s
        self.theta0 = 0                 # Initial angular position of rod a theta()
```

```
24          self.set0 = (0,-100)            # for selecting one of the two
    solutions
25          self.k = 0                      # Initial time for animation
26          self.c_position = []            # Piston position for animation
27          self.c_speed = []               # storing piston speed
28          self.c_time = []                # storing the time intervals
29          self.roi_speed=[]               # store roi speed
30          self.roi_time=[]                # store roi time
31          self.pos_old = 0                # old position of the piston
    for c_dot calculation
32
33      # Angular position of rod a as a function of time
34      def theta(self,t):
35          theta = self.theta0 + self.omega*t
36          return theta
37
38      # position of end point of rod p
39      def rod_p_position(self,t):
40          p_y = self.p*np.sin(self.theta(t))
41          p_x = self.p*np.cos(self.theta(t))
42          return p_x,p_y
43
44      # position of end point of rod q
45      def rod_q_position(self,t):
46          px,py = self.rod_p_position(t)
47          ax,ay = link_a_pivot
48          c = ((self.p**2-px**2-py**2) - (self.a**2-ax**2-ay**2)) /
    (2*(ax-px))
49          d = (py-ay)/(ax-px)
50          D = (d*(px-c)+py)**2 - (1+d**2)*(py**2 + (px-c)**2 - self.p
    **2)
51          if D<0:
52              """
53              any length combination resulting in complex i.e.
    mechanism breaks
54              will be skipped
55              """
56              raise Exception('complex')
57          D2 = D**0.5
58          y0 = (d*(px-c)+py+D2)/(1+d**2)
```

```
59          y1 = (d*(px-c)+py-D2)/(1+d**2)
60          x0 = c+d*y0
61          x1 = c+d*y1
62          set1=(x0,y0)
63          set2=(x1,y1)
64
65          dist1 = math.hypot(set1[0] - self.set0[0], set1[1] - self.
    set0[1])
66          dist2 = math.hypot(set2[0] - self.set0[0], set2[1] - self.
    set0[1])
67          if dist1<dist2:
68              self.set0=set1
69              return set1[0],set1[1]
70          else:
71              self.set0=set2
72              return set2[0],set2[1]
73
74      # position of piston end (i.e. slider)
75      def piston_position(self,t):
76          q_x,q_y = self.rod_q_position(t)
77          h0 = q_x+(self.b**2 - (q_y-link_a_pivot[1])**2)**0.5
78          return h0
79
80      # Piston speed
81      def c_dot(self,t):
82          c_x = self.piston_position(t)
83          c_dot = abs(c_x-self.pos_old)/0.01  # dt = 0.01
84          self.pos_old = c_x
85          return c_dot
86
87      # stores and computes velocity data
88      def velocity_params(self,data_stored):
89          """
90          Here the value time="3.5" is a hyperparameter. It denote the
    time taken
91          for the mechanism to complete one cycle. To find it, first
    set it to some
92          value say "10" and plot the slider speed vs time curve using
    'slider_speed.csv'
93          and note the time taken for one cycle from the curve
```

```
94          """
95          while self.k<3.5:
96              speed = self.c_dot(self.k)
97
98              self.c_speed.append(speed)
99              self.c_time.append(self.k)
100
101             # update time step
102             self.k+=0.01
103
104         # collect data (time,slider_speed) for a single set of params
     (lengths) using "slider_speed.py"
105         if data_stored=="slider_speed":
106             print("Computing time series data for slider speed")
107             for itr in range(len(self.c_time)):
108                 with open('slider_speed.csv', mode='a') as label_file
    :
109                     label_writer = csv.writer(label_file, delimiter=',
    ', quotechar='"', quoting=csv.QUOTE_MINIMAL)
110                     label_writer.writerow([self.c_time[itr],self.
    c_speed[itr]])
111
112
113         elif data_stored=="iterations":
114             print("Computing slider speed for various link length
    combinations")
115             for itr in range(len(self.c_time)):
116                 # store the Region of Interest (RoI) i.e. plateau
117                 if self.c_time[itr]>0.4 and self.c_time[itr]<2.1:
118                     self.roi_speed.append(self.c_speed[itr])
119                     self.roi_time.append(self.c_time[itr])
120
121             maxi = []
122             mini = []
123
124             for i in range(1,len(self.roi_speed)-1):
125
126                 # local maxima
127                 if self.roi_speed[i]>self.roi_speed[i+1] and self.
    roi_speed[i]>self.roi_speed[i-1]:
```

```
                    maxi.append(self.roi_speed[i])

                # local minima
                if self.roi_speed[i]<self.roi_speed[i+1] and self.
    roi_speed[i]<self.roi_speed[i-1]:
                    mini.append(self.roi_speed[i])

            with open('length_iterations.csv', mode='a') as
    label_file:
                label_writer = csv.writer(label_file, delimiter=',',
    quotechar='"', quoting=csv.QUOTE_MINIMAL)
                label_writer.writerow([self.a,self.b,self.p,self.q,max
    (maxi),min(mini)])

            mini.clear()
            maxi.clear()
            self.c_speed.clear()
            self.k.clear()
            self.roi_speed.clear()
            self.roi_time.clear()
```

```
"""
this script generates time series data for the slider speed
data stored @ slider_speed.csv

@nimrobotics
"""

from mechanism import *
import time

initial_time = time.time()
m = My_mechanism(26,79,18,22,2)

m.velocity_params(data_stored="slider_speed")
print("Processing time : ",time.time()-initial_time,"s")
```

```
"""
this script carries out length_iterations for find the best constant
    velocity
```

```
curve and stores the data @ 'length_iterations.csv'

@nimrobotics
"""

from mechanism import *
import time

initial_time = time.time()

for a in range(20,35):
    for b in range(70,80):
        for p in range(15,25):
            for q in range(15,30):
                m = My_mechanism(a,b,p,q,2)      #a,b,p,q,omega
                try:
                    m.velocity_params(data_stored="iterations")
                except Exception:
                    continue

print("Processing time : ",time.time()-initial_time,"s")
```

C.2 GUI software package core (Python, PyQT5)

```
"""
Contains main class for calculating and visualizing the mechanism

@nimrobotics
"""

import numpy as np
import math

# ground for link a and p
link_a_pivot = (0,-10)
link_p_pivot = (0,0)


```

```
15 class My_mechanism(object):
16
17     # The init function
18     def __init__(self,a,b,p,q,omega):
19         self.a = a                                  # Rod a
20         self.b = b                                  # Rod b
21         self.p = p                                  # Rod p
22         self.q = q                                  # Rod q
23         self.omega = omega                          # Angular speed of rod
    p in rad/s
24         self.theta0 = 0                             # Initial angular
    position of rod a
25         self.set0 = (0,-100)                        # for selecting one of
    the two solutions
26         self.k=0                                    # Initial time for
    animation
27         self.conn_rod_angular_speed = []            # Classes for the
    graphs animation
28         self.c_speed = []                           # storing piston speed
29         self.c_time = []                            # storing the time
    intervals
30         self.pos_old = 0                            # old position of the
    piston for c_dot calculation
31
32     # Angular position of rod p as a function of time
33     def theta(self,t):
34         theta = self.theta0 + self.omega*t
35         return theta
36
37     # position of end point of rod p
38     def rod_p_position(self,t):
39         p_x = self.p*np.cos(self.theta(t))
40         p_y = self.p*np.sin(self.theta(t))
41         return p_x,p_y
42
43     # position of end point of rod q
44     def rod_q_position(self,t):
45         px,py = self.rod_p_position(t)
46         ax,ay = link_a_pivot
```

```
47          c = ((self.p**2-px**2-py**2) - (self.a**2-ax**2-ay**2)) /
    (2*(ax-px))
48          d = (py-ay)/(ax-px)
49          D = (d*(px-c)+py)**2 - (1+d**2)*(py**2 + (px-c)**2 - self.p
    **2)
50          if D<0:
51              """
52              any length combination resulting in complex i.e.
    mechanism breaks
53              will be skipped
54              """
55              raise Exception('complex')
56          D2 = D**0.5
57          y0 = (d*(px-c)+py+D2)/(1+d**2)
58          y1 = (d*(px-c)+py-D2)/(1+d**2)
59          x0 = c+d*y0
60          x1 = c+d*y1
61          set1=(x0,y0)
62          set2=(x1,y1)
63
64          dist1 = math.hypot(set1[0] - self.set0[0], set1[1] - self.
    set0[1])
65          dist2 = math.hypot(set2[0] - self.set0[0], set2[1] - self.
    set0[1])
66          if dist1<dist2:
67              self.set0=set1
68              return set1[0],set1[1]
69          else:
70              self.set0=set2
71              return set2[0],set2[1]
72
73      # position of piston end (i.e. slider)
74      def piston_position(self,t):
75          q_x,q_y = self.rod_q_position(t)
76          h0 = q_x+(self.b**2 - (q_y-link_a_pivot[1])**2)**0.5
77          return h0
78
79      # Piston speed
80      def c_dot(self,t):
81          c_x = self.piston_position(t)
```

```
82          c_dot = abs(c_x-self.pos_old)/0.01  # dt = 0.01
83          self.pos_old = c_x
84          return c_dot
85
86  class My_mechanism_slider(object):
87
88      # The init function
89      def __init__(self,a,b,omega):
90          self.a = a                              # Rod a
91          self.b = b                              # Rod b
92          self.omega = omega                      # Angular speed of rod
    a in rad/s
93          self.theta0 = 0                         # Initial angular
    position of rod a
94          self.set0 = (0,-100)                    # for selecting one of
    the two solutions
95          self.k=0                                # Initial time for
    animation
96          self.c_position = []                    # Piston position for
    animation
97          self.c_speed = []                       # storing piston speed
98          self.c_time = []                        # storing the time
    intervals
99          self.pos_old = 0                        # old position of the
    piston for c_dot calculation
100
101     # Angular position of rod p as a function of time
102     def theta(self,t):
103         theta = self.theta0 + self.omega*t
104         return theta
105
106     # position of end point of rod a
107     def rod_a_position(self,t):
108         return self.a*np.cos(self.theta(t)),self.a*np.sin(self.theta(
    t))
109
110     # position of piston end (i.e. slider)
111     def piston_position(self,t):
112         a_x,a_y = self.rod_a_position(t)
113         return a_x+(self.b**2 - (a_y-link_a_pivot[1])**2)**0.5
```

```
114
115     # Piston speed
116     def c_dot(self,t):
117         c_x = self.piston_position(t)
118         c_dot = abs(c_x-self.pos_old)/0.01  # dt = 0.01
119         self.pos_old = c_x
120         return c_dot
```

```
1  """
2  Functionality to finds analytical solution for kinetaic and dynamic
3  analysis of slider crank mechanism
4
5  @nimrobotics for OpenKDM, 2020
6  """
7
8  import numpy as np
9
10 class link(object):
11   """finds parameters like mass, inertia of each link"""
12   def __init__(self, dim, rho):
13     self.l = dim[0]
14     self.b = dim[1]
15     self.h = dim[2]
16     self.rho = rho
17
18   def mass(self):
19     """
20     finds mass of a link
21     """
22     return self.l*self.b*self.h*self.rho
23
24   def inertia(self):
25     """
26     Calculates inertia of a link by neglecting the rounded edges
27     """
28     return (self.mass()/12)*(self.h**2 + self.l**2) + (self.mass()*
       self.l**2)/4
29
30
31 class Vectors:
```

```
32      """Stores i,j component of a vector"""
33      def __init__(self, i,j,k):
34          self.i = i
35          self.j = j
36          self.k = k
37
38
39  def sliderCrank(link1, link2, m3, Wao, theta, rho, g):
40    Lao=link(link1,rho)
41    Lab=link(link2,rho)
42
43    # print(np.deg2rad(theta),theta)
44    theta=np.deg2rad(theta)
45    if theta<=np.pi:
46      theta3=np.pi-abs(np.arcsin((Lao.l*np.sin(theta))/Lab.l))
47    else:
48      theta3=np.pi+abs(np.arcsin((Lao.l*np.sin(theta))/Lab.l))
49    # print(theta3,np.rad2deg(theta3))
50
51    Va = Vectors(-Lao.l*Wao*np.sin(theta),Lao.l*Wao*np.cos(theta),0)
52    Wba = Vectors(0,0,(Lao.l*Wao*np.cos(theta))/(Lab.l*np.cos(theta3)))
53    Vb = Vectors(Lab.l*Wba.k*np.sin(theta3) - Lao.l*Wao*np.sin(theta)
       ,0,0)
54    Aa = Vectors(-Wao**2*Lao.l*np.cos(theta), -Wao**2*Lao.l*np.sin(
       theta),0)
55    ALPHAba = Vectors(0,0,(Wba.k**2*Lab.l*np.sin(theta3) - Wao**2*Lao.l
       *np.sin(theta))/(Lab.l*np.cos(theta3)))
56    Ab = Vectors((ALPHAba.k*np.sin(theta3) + Wba.k**2*np.cos(theta3))*
       Lab.l - Lao.l*Wao**2*np.cos(theta),0,0)
57    Aba = Vectors((np.sin(theta3)*ALPHAba.k + Wba.k**2*np.cos(theta3))*
       Lab.l,(-np.cos(theta3)*ALPHAba.k + Wba.k**2*np.sin(theta3))*Lab.l
       ,0)
58    Ac1 = Vectors(Aa.i/2,Aa.j/2,Aa.k/2)
59    Ac2 = Vectors(Aa.i + Aba.i/2, Aa.j + Aba.j/2, Aa.k + Aba.k/2)
60
61    A = np.array([[1,0,1,0,0,0,0,0],
62          [0,1,0,1,0,0,0,0],
63          [0,0,-1,0,1,0,0,0],
64          [0,0,0,-1,0,1,0,0],
65          [0,0,0,0,1,0,0,0],
```

```
66          [0,0,0,0,0,-1,1,0],
67          [0,0,-Lao.l*np.sin(theta),Lao.l*np.cos(theta),0,0,0,1],
68          [0,0,0,0,Lab.l*np.sin(theta3),Lab.l*np.cos(theta3),0,0]])
69
70    B = np.array([[Lao.mass()*Ac1.i],
71            [Lao.mass()*Ac1.j + Lao.mass()*g],
72            [Lab.mass()*Ac2.i],
73            [Lab.mass()*Ac2.j + Lab.mass()*g],
74            [-m3*Ab.i],
75            [m3*g],
76            [Lao.l*0.5*np.cos(theta)*Lao.mass()*(Ac1.j+g)] - Lao.l*0.5*
       np.sin(theta)*Lao.mass()*Ac1.i,
77            [Lab.inertia()*ALPHAba.k + Lab.l*0.5*np.sin(theta3)*Lab.
       mass()*Ac2.i - Lab.l*0.5*np.cos(theta3)*Lab.mass()*(Ac2.j+g)]])
78
79    # Fox, Foy, Fax, Fay, Fbx, Fby, N, T
80    result = np.dot(np.linalg.inv(A),B)
81    Fo = Vectors(result[0][0],result[1][0],0)
82    Fa = Vectors(result[2][0],result[3][0],0)
83    Fb = Vectors(result[4][0],result[5][0],0)
84    N = Vectors(result[6][0],0,0)
85    T = Vectors(0,0,result[7][0])
86
87    # print(result)
88
89    resultArr = np.zeros((14,3))
90    values = [Va,Wba,Vb,Aa,ALPHAba,Ab,Aba,Ac1,Ac2,Fo,Fa,Fb,N,T]
91    for u,value in enumerate(values):
92      resultArr[u]=[value.i,value.j,value.k]
93      # print(value.i,value.j,value.k,"\n")
94
95    # print(resultArr)
96    return(resultArr)
97
98
99  #-------------------------------- INIT
        ------------------------------------------
100 # density of aluminum 2,710kg/m3 or 2.7 g/cm
101 # Wao is the input angular velcoity
102 # theta is the position of crank
```

```
# [l,b,h] in meters
sliderCrank([0.13,0.03,0.01],[0.395,0.03,0.01],m3=0.25,Wao=2, theta
    =90,rho=2710,g=9.81)
```

```
from fbs_runtime.application_context.PyQt5 import ApplicationContext
# https://gist.github.com/MalloyDelacroix/2
    c509d6bcad35c7e35b1851dfc32d161

import sys
from PyQt5.QtCore import QObject, pyqtSlot, Qt, pyqtSignal
from PyQt5.QtGui import QPainter, QFont, QPixmap
from PyQt5.QtWidgets import (QAction, QApplication, QHeaderView,
    QHBoxLayout, QLabel, QLineEdit,
                                QMainWindow, QPushButton, QRadioButton
    , QTableWidget, QTableWidgetItem,
                                QVBoxLayout, QWidget,  QMessageBox,
    QSizePolicy, QGridLayout, QPlainTextEdit, QCheckBox)

from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as
    FigureCanvas
from matplotlib.figure import Figure
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from mechanismClass import *
#  for analytical solutions
from analytical import sliderCrank


class plot_figure(FigureCanvas):

    def __init__(self, width=50, height=50, dpi=100, parent=None,):
        self.fig = Figure(figsize=(width, height), dpi=dpi)
        FigureCanvas.__init__(self, self.fig)
        self.setParent(parent)
        FigureCanvas.setSizePolicy(self, QSizePolicy.Expanding,
    QSizePolicy.Expanding)
        FigureCanvas.updateGeometry(self)

    def draw_graph_fourbar(self, a, b, p, q, omega):
        self.ax1 = self.fig.add_subplot(2,1,1)
```

```
31          # self.ax2 = self.fig.add_subplot(2,1,2)
32          self.m = My_mechanism(a,b,p,q,omega)
33          self.anim = animation.FuncAnimation(self.fig, self.
    animate_loop_fourbar, interval=10)
34          self.draw()
35
36      def animate_loop_fourbar(self,i):
37          p_x,p_y = self.m.rod_p_position(self.m.k)
38          q_x,q_y = self.m.rod_q_position(self.m.k)
39          # c_x = self.m.piston_position(self.m.k)
40          self.ax1.clear()
41          # self.ax2.clear()
42          # rod P
43          self.ax1.plot([link_p_pivot[0],p_x],[link_p_pivot[1],p_y],
    linewidth=3,color='blue')
44          # rod Q
45          self.ax1.plot([p_x,q_x],[p_y,q_y],linewidth=3,color='green')
46          # rod A
47          self.ax1.plot([q_x,link_a_pivot[0]],[q_y,link_a_pivot[1]],
    linewidth=3,color='red')
48          # rod B
49          # self.ax1.plot([q_x,c_x],[q_y,link_a_pivot[1]],linewidth=3,
    color='yellow')
50          # Piston (c)
51          # self.ax1.plot(c_x,link_a_pivot[1],'s',markersize=20,color='
    magenta')
52          self.ax1.set_xlim(-50,50)
53          self.ax1.set_ylim(-50,50)
54          self.ax1.set_title('Crankshaft, connecting rod and piston
    mechanism')
55          # Piston speed
56          # self.m.c_speed.append(self.m.c_dot(self.m.k))
57          # self.m.c_time.append(100*self.m.k)
58          # self.ax2.plot(self.m.c_time,self.m.c_speed,color='green')
59          # self.ax2.set_xlim(0,600)
60          # self.ax2.set_ylim(0,200)
61          # self.ax2.set_ylabel("Speed $(m/s)$")
62          # self.ax2.set_xlabel("time $(s*100)$")
63          # self.ax2.set_title('Piston speed')
64          self.ax1.set_aspect("equal")
```

```
65          # self.ax2.set_aspect("equal")
66          self.m.k += 0.01
67
68
69      def draw_graph_sixbar(self, a, b, p, q, omega):
70          self.ax1 = self.fig.add_subplot(2,1,1)
71          self.ax2 = self.fig.add_subplot(2,1,2)
72          self.m = My_mechanism(a,b,p,q,omega)
73          self.anim = animation.FuncAnimation(self.fig, self.
    animate_loop_sixbar, interval=10)
74          self.draw()
75
76      def animate_loop_sixbar(self,i):
77          p_x,p_y = self.m.rod_p_position(self.m.k)
78          q_x,q_y = self.m.rod_q_position(self.m.k)
79          c_x = self.m.piston_position(self.m.k)
80          self.ax1.clear()
81          self.ax2.clear()
82          # rod P
83          self.ax1.plot([link_p_pivot[0],p_x],[link_p_pivot[1],p_y],
    linewidth=3,color='blue')
84          # rod Q
85          self.ax1.plot([p_x,q_x],[p_y,q_y],linewidth=3,color='green')
86          # rod A
87          self.ax1.plot([q_x,link_a_pivot[0]],[q_y,link_a_pivot[1]],
    linewidth=3,color='red')
88          # rod B
89          self.ax1.plot([q_x,c_x],[q_y,link_a_pivot[1]],linewidth=3,
    color='yellow')
90          # Piston (c)
91          self.ax1.plot(c_x,link_a_pivot[1],'s',markersize=20,color='
    magenta')
92          self.ax1.set_xlim(-50,130)
93          self.ax1.set_ylim(-50,50)
94          self.ax1.set_title('Crankshaft, connecting rod and piston
    mechanism')
95          # Piston speed
96          self.m.c_speed.append(self.m.c_dot(self.m.k))
97          self.m.c_time.append(100*self.m.k)
98          self.ax2.plot(self.m.c_time,self.m.c_speed,color='green')
```

```
99          self.ax2.set_xlim(0,600)
100         self.ax2.set_ylim(0,200)
101         self.ax2.set_ylabel("Speed $(m/s)$")
102         self.ax2.set_xlabel("time $(s*100)$")
103         self.ax2.set_title('Piston speed')
104         self.ax1.set_aspect("equal")
105         self.ax2.set_aspect("equal")
106         self.m.k += 0.01
107
108
109     def draw_graph_slider(self, a, b, omega):
110         self.ax1 = self.fig.add_subplot(2,1,1)
111         self.ax2 = self.fig.add_subplot(2,1,2)
112         self.m = My_mechanism_slider(a,b,omega)
113         self.anim = animation.FuncAnimation(self.fig, self.
    animate_loop_slider, interval=10)
114         self.draw()
115
116     def animate_loop_slider(self,i):
117         # p_x,p_y = self.m.rod_p_position(self.m.k)
118         a_x,a_y = self.m.rod_a_position(self.m.k)
119         c_x = self.m.piston_position(self.m.k)
120         self.ax1.clear()
121         self.ax2.clear()
122         # rod P
123         # self.ax1.plot([link_p_pivot[0],p_x],[link_p_pivot[1],p_y],
    linewidth=3,color='blue')
124         # rod Q
125         # self.ax1.plot([p_x,q_x],[p_y,q_y],linewidth=3,color='green
    ')
126         # rod A
127         self.ax1.plot([a_x,link_a_pivot[0]],[a_y,link_a_pivot[1]],
    linewidth=3,color='red')
128         # rod B
129         self.ax1.plot([a_x,c_x],[a_y,link_a_pivot[1]],linewidth=3,
    color='yellow')
130         # Piston (c)
131         self.ax1.plot(c_x,link_a_pivot[1],'s',markersize=20,color='
    magenta')
132         self.ax1.set_xlim(-50,130)
```

```
133         self.ax1.set_ylim(-50,50)
134         self.ax1.set_title('Crankshaft, connecting rod and piston
    mechanism')
135         # Piston speed
136         self.m.c_speed.append(self.m.c_dot(self.m.k))
137         self.m.c_time.append(100*self.m.k)
138         # self.ax2.plot(self.m.c_time,self.m.c_time)
139         self.ax2.plot(self.m.c_time,self.m.c_speed,color='green')
140         self.ax2.set_xlim(0,600)
141         self.ax2.set_ylim(0,200)
142         self.ax2.set_ylabel("Speed $(m/s)$")
143         self.ax2.set_xlabel("time $(s*100)$")
144         self.ax2.set_title('Piston speed')
145         self.ax1.set_aspect("equal")
146         self.ax2.set_aspect("equal")
147         self.m.k += 0.01
148
149
150     def clear_plot(self):
151         self.anim.event_source.stop()
152         if self.ax1 is not None:
153             self.ax1.clear()
154         if self.ax2 is not None:
155             self.ax2.clear()
156         self.draw()
157
158
159 class Widget(QWidget):
160     def __init__(self):
161         QWidget.__init__(self)
162         self.a=26
163         self.b=76
164         self.p=18
165         self.q=22
166         self.omega=2
167
168         # Right
169         # a=26,b=79,p=18,q=22,omega=2
170         self.plot_figure = plot_figure(width=8, height=4)
171         self.right = QVBoxLayout()
```

```
172         self.right.addWidget(self.plot_figure)
173
174         # Left: selectMechanism radio buttons
175         self.fourbar = QRadioButton("Four bar")
176         self.slidercrank = QRadioButton("Slider crank")
177         self.sixbar = QRadioButton("Six bar")
178
179         self.selectMechanism = QHBoxLayout()
180         self.selectMechanism.addWidget(self.fourbar)
181         self.selectMechanism.addWidget(self.slidercrank)
182         self.selectMechanism.addWidget(self.sixbar)
183
184         self.fourbar.setChecked(True)
185         self.fourbar.toggled.connect(self.four_bar)
186         self.slidercrank.toggled.connect(self.slider_crank)
187         self.sixbar.toggled.connect(self.six_bar)
188
189         # left: link lengths input
190         self.lenA = QLineEdit()
191         self.lenAbox = QHBoxLayout()
192         self.lenAbox.addWidget(QLabel("Link A length"))
193         self.lenAbox.addWidget(self.lenA)
194         self.lenAbox.addWidget(QLabel("cm"))
195
196         self.lenB = QLineEdit()
197         self.lenB.setEnabled(False)
198         self.lenBbox = QHBoxLayout()
199         self.lenBbox.addWidget(QLabel("Link B length"))
200         self.lenBbox.addWidget(self.lenB)
201         self.lenBbox.addWidget(QLabel("cm"))
202
203         self.lenP = QLineEdit()
204         self.lenPbox = QHBoxLayout()
205         self.lenPbox.addWidget(QLabel("Link P length"))
206         self.lenPbox.addWidget(self.lenP)
207         self.lenPbox.addWidget(QLabel("cm"))
208
209         self.lenQ = QLineEdit()
210         self.lenQbox = QHBoxLayout()
211         self.lenQbox.addWidget(QLabel("Link Q length"))
```

```
212         self.lenQbox.addWidget(self.lenQ)
213         self.lenQbox.addWidget(QLabel("cm"))
214 
215         self.omega = QLineEdit()
216         self.omegabox = QHBoxLayout()
217         self.omegabox.addWidget(QLabel("Angular speed"))
218         self.omegabox.addWidget(self.omega)
219         self.omegabox.addWidget(QLabel("rad/s"))
220 
221 
222         # left
223         self.description = QLineEdit()
224         self.price = QLineEdit()
225         self.add = QPushButton("Simulate")
226         # Disabling 'Add' button
227         self.add.setEnabled(False)
228         self.reset = QPushButton("Reset")
229         self.clear = QPushButton("Clear")
230         self.quit = QPushButton("Quit")
231 
232         self.left = QVBoxLayout()
233         self.left.addWidget(QLabel("Select the mechanism"))
234         self.left.addLayout(self.selectMechanism)
235         self.left.addWidget(QLabel("Specify the mechanism params"))
236         self.left.addLayout(self.lenAbox)
237         self.left.addLayout(self.lenBbox)
238         self.left.addLayout(self.lenPbox)
239         self.left.addLayout(self.lenQbox)
240         self.left.addLayout(self.omegabox)
241         self.left.addWidget(self.add)
242         self.left.addWidget(self.reset)
243         self.left.addWidget(self.clear)
244         self.left.addWidget(self.quit)
245 
246         # Signals and pyqtSlots
247         self.add.clicked.connect(self.add_element)
248         self.quit.clicked.connect(self.quit_application)
249         self.reset.clicked.connect(self.reset_values)
250         self.clear.clicked.connect(self.clear_inputs)
251 
```

```
252
253         # check when to enable Simulate button
254         self.lenA.textChanged[str].connect(self.check_disable)
255         self.lenB.textChanged[str].connect(self.check_disable)
256         self.lenP.textChanged[str].connect(self.check_disable)
257         self.lenQ.textChanged[str].connect(self.check_disable)
258         self.omega.textChanged[str].connect(self.check_disable)
259         # self.fourbar.
260
261
262         # QWidget Layout
263         self.layout = QHBoxLayout()
264
265         # self.layout.addWidget(self.table)
266         self.layout.addLayout(self.left)
267         self.layout.addLayout(self.right)
268
269         # Set the layout to the QWidget
270         self.setLayout(self.layout)
271
272
273     # onClick for fourbar radio
274     @pyqtSlot()
275     def four_bar(self):
276         if self.sender().isChecked():
277             self.lenP.setEnabled(True)
278             self.lenQ.setEnabled(True)
279             self.lenA.setEnabled(True)
280             self.lenB.setEnabled(False)
281         self.check_disable()
282
283     # onClick for slidercrank radio
284     @pyqtSlot()
285     def slider_crank(self):
286         if self.sender().isChecked():
287             self.lenP.setEnabled(False)
288             self.lenQ.setEnabled(False)
289             self.lenA.setEnabled(True)
290             self.lenB.setEnabled(True)
291         self.check_disable()
```

```
292
293     # onClick for sixbar radio
294     @pyqtSlot()
295     def six_bar(self):
296         if self.sender().isChecked():
297             self.lenP.setEnabled(True)
298             self.lenQ.setEnabled(True)
299             self.lenA.setEnabled(True)
300             self.lenB.setEnabled(True)
301         self.check_disable()
302
303     # main Simulate button
304     @pyqtSlot()
305     def add_element(self):
306         if self.fourbar.isChecked():
307             self.plot_figure.draw_graph_fourbar(int(self.lenA.text())
    , 0,
308                                                 int(self.lenP.text()), int(
    self.lenQ.text()), int(self.omega.text()))
309             print(self.lenA.text(),self.lenP.text(),self.lenQ.text(),
    self.omega.text())
310
311         if self.slidercrank.isChecked():
312             self.plot_figure.draw_graph_slider(int(self.lenA.text()),
     int(self.lenB.text()),int(self.omega.text()))
313             print(self.lenA.text(),self.lenB.text(),self.omega.text()
    )
314
315         if self.sixbar.isChecked():
316             self.plot_figure.draw_graph_sixbar(int(self.lenA.text()),
     int(self.lenB.text()),
317                                                 int(self.lenP.text()), int(
    self.lenQ.text()), int(self.omega.text()))
318             print(self.lenA.text(),self.lenB.text(),self.lenP.text(),
    self.lenQ.text(),self.omega.text())
319
320     # enable add button after required inputs are met
321     @pyqtSlot()
322     def check_disable(self):
323         if self.fourbar.isChecked():
```

```
324             if not self.lenA.text() or not self.lenP.text() or not
    self.lenQ.text() or not self.omega.text():
325                 self.add.setEnabled(False)
326             else:
327                 self.add.setEnabled(True)
328 
329         if self.slidercrank.isChecked():
330             if not self.lenA.text() or not self.lenB.text() or not
    self.omega.text():
331                 self.add.setEnabled(False)
332             else:
333                 self.add.setEnabled(True)
334 
335         if self.sixbar.isChecked():
336             if not self.lenA.text() or not self.lenB.text() or not
    self.lenP.text() or not self.lenQ.text() or not self.omega.text():
337                 self.add.setEnabled(False)
338             else:
339                 self.add.setEnabled(True)
340 
341     @pyqtSlot()
342     def quit_application(self):
343         QApplication.quit()
344 
345     # sets the default values i.e.six bar constant velocity
346     @pyqtSlot()
347     def reset_values(self):
348         # a=26,b=79,p=18,q=22,omega=2
349         self.lenA.setText("26")
350         self.lenB.setText("79")
351         self.lenP.setText("18")
352         self.lenQ.setText("22")
353         self.omega.setText("2")
354         self.sixbar.setChecked(True)
355 
356     # clears all input fields
357     @pyqtSlot()
358     def clear_inputs(self):
359         self.lenA.setText("")
360         self.lenB.setText("")
```

```
361         self.lenP.setText("")
362         self.lenQ.setText("")
363         self.omega.setText("")
364         self.fourbar.setChecked(True)
365         self.plot_figure.clear_plot()
366 
367 
368 class analWidget(QWidget):
369     def __init__(self):
370         QWidget.__init__(self)
371         self.a=26
372         self.b=76
373         self.p=18
374         self.q=22
375         self.omega=2
376 
377         # Right
378         # self.plot_figure = plot_figure(width=8, height=4)
379         # self.right = QVBoxLayout()
380         # self.right.addWidget(self.plot_figure)
381         self.tableWidget = QTableWidget()
382         self.tableWidget.setRowCount(14)
383         self.tableWidget.setColumnCount(3)
384         self.tableWidget.setHorizontalHeaderLabels(["x-component", "y
    -component","z-component"])
385         # Va,Wba,Vb,Aa,ALPHAba,Ab,Aba,Ac1,Ac2,Fo,Fa,Fb,N,T
386         self.tableWidget.setVerticalHeaderLabels(["Velocity_a", "
    Omega_ab","Velocity_b","Acceleration_a","Alpha_ab","Acceleration_b
    ","Acceleration_ab","Acceleration_c1","Acceleration_c2","Force_o",
    "Force_a","Force_b","N","T"])
387         # self.tableWidget.setItem(0,0, QTableWidgetItem("as"))
388         # self.tableWidget.setItem(0,1, QTableWidgetItem("Ceddll
    (1,2)"))
389         # self.tableWidget.setItem(0,2, QTableWidgetItem("Ceddll
    (1,2)"))
390         # for i in range(10):
391         #     self.tableWidget.setItem(i,0, QTableWidgetItem("as"))
392         #     self.tableWidget.setItem(i,1, QTableWidgetItem("as"))
393         #     self.tableWidget.setItem(i,2, QTableWidgetItem("as"))
394         self.tableWidget.horizontalHeader().setSectionResizeMode(
```

```
        QHeaderView.Stretch)
395
396         # # Left: selectMechanism radio buttons
397         # self.fourbar = QRadioButton("Four bar")
398         # self.slidercrank = QRadioButton("Slider crank")
399         # self.sixbar = QRadioButton("Six bar")
400
401         # self.selectMechanism = QHBoxLayout()
402         # self.selectMechanism.addWidget(self.fourbar)
403         # self.selectMechanism.addWidget(self.slidercrank)
404         # self.selectMechanism.addWidget(self.sixbar)
405
406         # self.fourbar.setChecked(True)
407         # self.fourbar.toggled.connect(self.four_bar)
408         # self.slidercrank.toggled.connect(self.slider_crank)
409         # self.sixbar.toggled.connect(self.six_bar)
410
411         # left: link lengths input
412         self.lenA = QLineEdit()
413         self.lenAbox = QHBoxLayout()
414         self.lenAbox.addWidget(QLabel("Link A length"))
415         self.lenAbox.addWidget(self.lenA)
416         self.lenAbox.addWidget(QLabel("m"))
417
418         self.lenB = QLineEdit()
419         self.lenBbox = QHBoxLayout()
420         self.lenBbox.addWidget(QLabel("Link B length"))
421         self.lenBbox.addWidget(self.lenB)
422         self.lenBbox.addWidget(QLabel("m"))
423
424         self.crankA = QLineEdit()
425         self.crankAngle = QHBoxLayout()
426         self.crankAngle.addWidget(QLabel("Crank angle"))
427         self.crankAngle.addWidget(self.crankA)
428         self.crankAngle.addWidget(QLabel("degrees"))
429
430         # self.lenP = QLineEdit()
431         # self.lenPbox = QHBoxLayout()
432         # self.lenPbox.addWidget(QLabel("Link P length"))
433         # self.lenPbox.addWidget(self.lenP)
```

```
434         # self.lenPbox.addWidget(QLabel("cm"))
435 
436         # self.lenQ = QLineEdit()
437         # self.lenQbox = QHBoxLayout()
438         # self.lenQbox.addWidget(QLabel("Link Q length"))
439         # self.lenQbox.addWidget(self.lenQ)
440         # self.lenQbox.addWidget(QLabel("cm"))
441 
442         self.omega = QLineEdit()
443         self.omegabox = QHBoxLayout()
444         self.omegabox.addWidget(QLabel("Angular speed"))
445         self.omegabox.addWidget(self.omega)
446         self.omegabox.addWidget(QLabel("rad/s"))
447 
448 
449         # left
450         self.description = QLineEdit()
451         self.price = QLineEdit()
452         self.add = QPushButton("Calculate")
453         # Disabling 'Add' button
454         # self.add.setEnabled(False)
455         self.reset = QPushButton("Reset")
456         self.clear = QPushButton("Clear")
457         self.quit = QPushButton("Quit")
458 
459         self.left = QVBoxLayout()
460         # self.left.addWidget(QLabel("Select the mechanism"))
461         # self.left.addLayout(self.selectMechanism)
462         self.left.addWidget(QLabel("Specify the required params (in
    SI units)"))
463         self.left.addLayout(self.lenAbox)
464         self.left.addLayout(self.lenBbox)
465         self.left.addLayout(self.crankAngle)
466         # self.left.addLayout(self.lenPbox)
467         # self.left.addLayout(self.lenQbox)
468         self.left.addLayout(self.omegabox)
469         self.left.addWidget(self.add)
470         self.left.addWidget(self.reset)
471         self.left.addWidget(self.clear)
472         self.left.addWidget(self.tableWidget)
```

```
473         self.left.addWidget(self.quit)
474 
475         # Signals and pyqtSlots
476         self.add.clicked.connect(self.add_element)
477         self.quit.clicked.connect(self.quit_application)
478         self.reset.clicked.connect(self.reset_values)
479         self.clear.clicked.connect(self.clear_inputs)
480 
481 
482         # # check when to enable Simulate button
483         # self.lenA.textChanged[str].connect(self.check_disable)
484         # self.lenB.textChanged[str].connect(self.check_disable)
485         # # self.lenP.textChanged[str].connect(self.check_disable)
486         # # self.lenQ.textChanged[str].connect(self.check_disable)
487         # self.omega.textChanged[str].connect(self.check_disable)
488         # # self.fourbar.
489 
490 
491         # QWidget Layout
492         self.layout = QHBoxLayout()
493 
494         # self.layout.addWidget(self.table)
495         self.layout.addLayout(self.left)
496         # self.layout.addLayout(self.right)
497 
498         # Set the layout to the QWidget
499         self.setLayout(self.layout)
500 
501 
502     # # onClick for fourbar radio
503     # @pyqtSlot()
504     # def four_bar(self):
505     #     if self.sender().isChecked():
506     #         self.lenP.setEnabled(True)
507     #         self.lenQ.setEnabled(True)
508     #         self.lenA.setEnabled(True)
509     #         self.lenB.setEnabled(False)
510     #     self.check_disable()
511 
512     # # onClick for slidercrank radio
```

```
513     # @pyqtSlot()
514     # def slider_crank(self):
515     #     if self.sender().isChecked():
516     #         self.lenP.setEnabled(False)
517     #         self.lenQ.setEnabled(False)
518     #         self.lenA.setEnabled(True)
519     #         self.lenB.setEnabled(True)
520     #     self.check_disable()
521
522     # # onClick for sixbar radio
523     # @pyqtSlot()
524     # def six_bar(self):
525     #     if self.sender().isChecked():
526     #         self.lenP.setEnabled(True)
527     #         self.lenQ.setEnabled(True)
528     #         self.lenA.setEnabled(True)
529     #         self.lenB.setEnabled(True)
530     #     self.check_disable()
531
532     # main Simulate button
533     @pyqtSlot()
534     def add_element(self):
535         # print(self.lenA.text(),self.crankA.text())
536         paramValues = sliderCrank([float(self.lenA.text())
    ,0.03,0.01],[float(self.lenB.text()),0.03,0.01],m3=0.25,Wao=float(
    self.omega.text()), theta=float(self.crankA.text()),rho=2710,g
    =9.81)
537         # print(paramValues)
538         # print(paramValues.shape)
539         for i in range(paramValues.shape[0]):
540             # print(paramValues[i][0])
541             self.tableWidget.setItem(i,0, QTableWidgetItem(str(
    paramValues[i][0])))
542             self.tableWidget.setItem(i,1, QTableWidgetItem(str(
    paramValues[i][1])))
543             self.tableWidget.setItem(i,2, QTableWidgetItem(str(
    paramValues[i][2])))
544             # self.tableWidget.setItem(i,0, QTableWidgetItem("agvhvs
    "))
545             # self.tableWidget.setItem(i,1, QTableWidgetItem("agvhvs
```

```
"))
546             # self.tableWidget.setItem(i,2, QTableWidgetItem("agvhvs
"))
547         self.tableWidget.horizontalHeader().setSectionResizeMode(
QHeaderView.Stretch)
548 
549         # if self.fourbar.isChecked():
550         #     self.plot_figure.draw_graph_fourbar(int(self.lenA.text
()), 0,
551         #                                         int(self.lenP.text()), int(
self.lenQ.text()), int(self.omega.text()))
552         #     print(self.lenA.text(),self.lenP.text(),self.lenQ.text
(),self.omega.text())
553 
554         # if self.slidercrank.isChecked():
555         #     self.plot_figure.draw_graph_slider(int(self.lenA.text()
), int(self.lenB.text()),int(self.omega.text()))
556         #     print(self.lenA.text(),self.lenB.text(),self.omega.text
())
557 
558         # if self.sixbar.isChecked():
559         #     self.plot_figure.draw_graph_sixbar(int(self.lenA.text()
), int(self.lenB.text()),
560         #                                         int(self.lenP.text()), int(
self.lenQ.text()), int(self.omega.text()))
561         #     print(self.lenA.text(),self.lenB.text(),self.lenP.text
(),self.lenQ.text(),self.omega.text())
562 
563     # # enable add button after required inputs are met
564     # @pyqtSlot()
565     # def check_disable(self):
566     #     if self.fourbar.isChecked():
567     #         if not self.lenA.text() or not self.lenP.text() or not
self.lenQ.text() or not self.omega.text():
568     #             self.add.setEnabled(False)
569     #         else:
570     #             self.add.setEnabled(True)
571 
572     #     if self.slidercrank.isChecked():
573     #         if not self.lenA.text() or not self.lenB.text() or not
```

```
self.omega.text():
#             self.add.setEnabled(False)
#         else:
#             self.add.setEnabled(True)

#     if self.sixbar.isChecked():
#         if not self.lenA.text() or not self.lenB.text() or not
self.lenP.text() or not self.lenQ.text() or not self.omega.text():
#             self.add.setEnabled(False)
#         else:
#             self.add.setEnabled(True)

@pyqtSlot()
def quit_application(self):
    QApplication.quit()

# sets the default values i.e.six bar constant velocity
@pyqtSlot()
def reset_values(self):
    # a=26,b=79,p=18,q=22,omega=2
    self.lenA.setText("0.26")
    self.lenB.setText("0.79")
    self.crankA.setText("0")
    # self.lenP.setText("18")
    # self.lenQ.setText("22")
    self.omega.setText("2")
    # self.sixbar.setChecked(True)

# clears all input fields
@pyqtSlot()
def clear_inputs(self):
    self.lenA.setText("")
    self.lenB.setText("")
    self.crankA.setText("")
    # self.lenP.setText("")
    # self.lenQ.setText("")
    self.omega.setText("")
    # self.fourbar.setChecked(True)
    # self.plot_figure.clear_plot()
```

```
612
613 class num_win(QMainWindow):
614     switch_window = pyqtSignal()
615     def __init__(self, widget):
616         QMainWindow.__init__(self)
617         self.setWindowTitle("OpenKDM")
618
619         # Menu
620         self.menu = self.menuBar()
621         self.file_menu = self.menu.addMenu("File")
622
623         # Exit QAction
624         exit_action = QAction("Exit", self)
625         exit_action.setShortcut("Ctrl+Q")
626         exit_action.triggered.connect(self.exit_app)
627
628         back_action = QAction("Back", self)
629         back_action.setShortcut("Ctrl+B")
630         back_action.triggered.connect(self.back)
631
632         self.file_menu.addAction(exit_action)
633         self.file_menu.addAction(back_action)
634         self.setCentralWidget(widget)
635
636     @pyqtSlot()
637     def exit_app(self):
638         QApplication.quit()
639
640     @pyqtSlot()
641     def back(self):
642         self.switch_window.emit()
643
644
645 class anal_win(QMainWindow):
646     switch_window = pyqtSignal()
647     def __init__(self, widget):
648         QMainWindow.__init__(self)
649         self.setWindowTitle("OpenKDM")
650
651         # Menu
```

```
652         self.menu = self.menuBar()
653         self.file_menu = self.menu.addMenu("File")
654 
655         # Exit QAction
656         exit_action = QAction("Exit", self)
657         exit_action.setShortcut("Ctrl+Q")
658         exit_action.triggered.connect(self.exit_app)
659 
660         back_action = QAction("Back", self)
661         back_action.setShortcut("Ctrl+B")
662         back_action.triggered.connect(self.back)
663 
664         self.file_menu.addAction(exit_action)
665         self.file_menu.addAction(back_action)
666         self.setCentralWidget(widget)
667 
668     @pyqtSlot()
669     def exit_app(self):
670         QApplication.quit()
671 
672     @pyqtSlot()
673     def back(self):
674         self.switch_window.emit()
675 
676 
677 # class anal_win(QMainWindow):
678 #     switch_window = pyqtSignal()
679 #     def __init__(self):
680 #         QMainWindow.__init__(self)
681 #         self.setWindowTitle('Window Two')
682 
683 #         layout = QGridLayout()
684 
685 #         self.button = QPushButton('Back')
686 #         self.button.clicked.connect(self.back)
687 
688 #         layout.addWidget(self.button)
689 
690 #         self.setLayout(layout)
691 
```

```
692 #     @pyqtSlot()
693 #     def back(self):
694 #         self.switch_window.emit()
695
696
697 class main_win(QWidget):
698
699     switch_window = pyqtSignal()
700     switch_window2 = pyqtSignal()
701
702     def __init__(self):
703         QWidget.__init__(self)
704         self.setWindowTitle('OpenKDM')
705
706         layout = QVBoxLayout()
707
708         self.labelImage = QLabel(self)
709         self.pixmap = QPixmap(appctxt.get_resource("logo_banner.png")
    )
710         self.pixmap=self.pixmap.scaledToWidth(512)  # image size
711         self.labelImage.setPixmap(self.pixmap)
712         self.labelImage.setAlignment(Qt.AlignCenter)
713
714         self.button = QPushButton('Numerical Simulation')
715         self.button.clicked.connect(self.login1)
716
717         self.button2 = QPushButton('Analytical Solutions')
718         self.button2.clicked.connect(self.login2)
719         self.button.resize(100,32)
720         # self.button.setSizePolicy(QSizePolicy.Preferred,QSizePolicy.
    Expanding)
721         # self.button2.setSizePolicy(QSizePolicy.Preferred,
    QSizePolicy.Expanding)
722
723         self.button.setEnabled(False)
724         self.button2.setEnabled(False)
725
726         self.intro_text = """The study of the 'Kinematics and
    Dynamics of Machinery' (IITT course code: ME2206) lies at the very
     core of a mechanical engineering background. Although, little has
```

```
        changed in the way the subject is presented, our methodology
       brings the subject alive and current. We present the design and
       fabrication of a novel experimental setup for carrying out static,
        kinematic and dynamic analysis of three different mechanisms in a
        single setup. The mechanism is designed to be configurable to
       three different types of mechanisms namely - double crank, slider
       crank and a six bar mechanism depending on the use case. The
       mechanism has retrofitted parts (different link lengths and
       sliders) to facilitate multiple experiments in the same setup. The
        learner gets to 'play' with the mechanism parameters and
       immediately understand their effects. This will enhance ones grasp
        of the concepts and the development of analytical skills. Hence
       greatly supplementing and reinforcing the theoretical
       understanding of the undergraduate students taking the course."""

        self.license_text = """MIT License

Copyright (c) 2020 Aakash Yadav

Permission is hereby granted, free of charge, to any person obtaining
     a copy
of this software and associated documentation files (the "Software"),
     to deal
in the Software without restriction, including without limitation the
     rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or
     sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be
     included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
     EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
      MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
     NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
     HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
     WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
```

```
 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN THE SOFTWARE. """

        self.text = QLabel()
        self.text.setText(self.intro_text)
        self.text.setWordWrap(True)
        self.text.setAlignment(Qt.AlignCenter)

        self.license_layout = QVBoxLayout()
        self.license = QPlainTextEdit(self.license_text)
        self.license.setReadOnly(True)
        self.license_layout.addWidget(self.license)
        self.license_layout.setContentsMargins(200, 0, 200, 0)

        self.agree_box = QCheckBox("I have read the license agreement
")
        # self.agree_box.setChecked(True)
        self.agree_box.stateChanged.connect(self.license_agree)
        self.license_layout.addWidget(self.agree_box)


        layout.addWidget(self.labelImage)
        layout.addWidget(self.text)
        layout.addLayout(self.license_layout)
        layout.addWidget(self.button)
        layout.addWidget(self.button2)
        layout.setSpacing(20)
        layout.setContentsMargins(200, 10, 200, 10) #left, top, right
, bottom


        self.setLayout(layout)

    @pyqtSlot()
    def login1(self):
        self.switch_window.emit()

    @pyqtSlot()
    def login2(self):
        self.switch_window2.emit()
```

```
779
780     @pyqtSlot()
781     def license_agree(self):
782         if self.sender().isChecked():
783             self.button.setEnabled(True)
784             self.button2.setEnabled(True)
785         else:
786             self.button.setEnabled(False)
787             self.button2.setEnabled(False)
788
789
790 class Controller:
791
792     def __init__(self):
793         pass
794
795     def show_main(self):
796         self.main_win = main_win()
797         widget = Widget()
798         self.num_win = num_win(widget)
799         analwidget = analWidget()
800         self.anal_win = anal_win(analwidget)
801         self.main_win.switch_window.connect(self.show_numerical)
802         self.main_win.switch_window2.connect(self.show_analytical)
803         self.num_win.close()
804         self.anal_win.close()
805         self.main_win.showMaximized()
806
807     def show_numerical(self):
808         self.num_win.switch_window.connect(self.show_main)
809         self.main_win.close()
810         self.num_win.showMaximized()
811
812     def show_analytical(self):
813         self.anal_win.switch_window.connect(self.show_main)
814         self.main_win.close()
815         self.anal_win.showMaximized()
816
817
818 if __name__ == '__main__':
```

```
819     appctxt = ApplicationContext()       # 1. Instantiate
    ApplicationContext
820     app = QApplication(sys.argv)
821     controller = Controller()
822     controller.show_main()
823     exit_code = appctxt.app.exec_()      # 2. Invoke appctxt.app.
    exec_()
824     sys.exit(exit_code)
```

C.3 Code for instrumentation and it's GUI (Arduino CLI, Bash, Python, YAD)

```
1 #!/bin/sh
2
3 echo 'Launched OpenKDM...'
4
5 yad --form --title "OpenKDM" --window-icon=128.png --width=300 --
    height=300 --center \
6   --text="Welcome! choose an option to begin with." \
7   --field="Set Motor Speed":fbtn "bash speed.sh" \
8   --field="Record Data":fbtn "bash record.sh" \
9   --field="Retrive Data":fbtn "bash retrive.sh" \
10  --button=gtk-cancel:1
```

```
1 #!/bin/sh
2
3 echo 'Started speed setting window...'
4
5 n=4
6
7 ANSWER=`yad --title "OpenKDM" --window-icon=128.png --width=300 --
    height=300 --center \
8   --text="Please set the motor speed for experiment." \
9   --form --separator=' ' --field="Motor speed (rpm):NUM"
    1\!1..8\!1\!0 `
10
11 exval=$?
```

```
# echo $exval
case $exval in
   0)
  echo '[1/'$n'] Updating motor speed (rpm)...'
  sed "/int speed_rpm/c\int speed_rpm=$ANSWER;" speed_control/
    speed_control.ino -i

  echo '[2/'$n'] Compiling arduino sketch...'
  arduino-cli compile --fqbn arduino:avr:uno /home/pi/Desktop/OpenKDM
    /speed_control/

  echo '[3/'$n'] Uploading arduino sketch (uno)...'
  arduino-cli upload --port /dev/ttyACM0 --fqbn arduino:avr:uno /home
    /pi/Desktop/OpenKDM/speed_control/
  echo '[4/'$n'] Process finshed. Check traceback for errors...'
  ;;
   *) echo "Motor speed not updated...";;
esac



# sed '/int speed_rpm/c\int speed_rpm=4;' speed_control/speed_control.
    ino -i

# echo '[2/'$n'] Compiling arduino sketch...'
# arduino-cli compile --fqbn arduino:avr:uno /home/pi/Desktop/OpenKDM
    /speed_control/

# echo '[1/'$n'] Uploading arduino sketch (uno)...'
# arduino-cli upload --port /dev/ttyACM0 --fqbn arduino:avr:uno /home
    /pi/Desktop/OpenKDM/speed_control/
```

```
#!/bin/sh

echo 'Started data recording window...'

# python3 record_data.py &
# sleep 10
# PID=$!
```

```
# kill $PID


# PID=$
# echo "killing $PID"
# kill -9 $(ps aux | grep -v grep | grep "record_data.py" | awk '{
    print $2}')


yad --form --title "OpenKDM" --window-icon=128.png --width=300 --
    height=300 --center \
  --text="Start data recording the sensor data." \
  --field="Start recording":fbtn "python3 record_data.py &" \
  --field="Stop recording":fbtn "bash kill.sh" \
  --button=gtk-cancel:1
```

```
#!/bin/sh

echo 'Started data retriving window...'

[ -f yad_data.list ] && rm yad_data.list

while IFS=, read -r field1 field2
do
    echo $field1 >> yad_data.list
    echo $field2 >> yad_data.list

done < sensor_data.csv

yad --title "OpenKDM" --window-icon=128.png --width=300 --height=300
    --center --list --column=Angle --column=Acceleration < yad_data.
    list


# yad --height=300 --list --column=Data --column=Daa < sensor_data.
    csv


# yad --form --title "OpenKDM" --window-icon=128.png --width=300 --
    height=300 --center \
```

```
#    --text="Start data recording the sensor data." \
#    --field="Start recording":fbtn "python3 record_data.py &" \
#    --field="Stop recording":fbtn "bash kill.sh" \
#    --button=gtk-cancel:1
```

```
/*
Connections of Drive and Arduino
Serial Port 0 is not used to connect to drive because its connected
    to USB-Serial and used to show information on console.

For Arduino Uno Software serial needs to be used as there is only one
     hardware serial port and its connected to USB-Serial.
    Drive to Arduino UNO/Nano connections
    GND          -      GND
    RXD          -      D3
    TXD          -      D2

For arduino mega and other arduinos with multiple hardware serial
    port, any port other than 0 can be selected to connect the drive.

    Drive to Arduino Mega2560 connections
    GND          -      GND
    RXD          -      Tx1/Tx2/Tx3
    TXD          -      Rx1/Rx2/Rx3

* This mode can be used when multiple motors are to be used to run at
     exactly the same RPM and same torque even though the voltage
    supply might be different.
* Also in this mode the direction of the motor can be controlled
    digitally via modbus ASCII commands to run the dc servo motor in
    both directions

* For more information see : https://robokits.co.in/motor-drives-
    drivers/encoder-dc-servo/rhino-dc-servo-driver-50w-compatible-with
    -modbus-uart-ascii-for-encoder-dc-servo-motor
*
* modified @nimrobotics for OpenKDM
*/

#include<RMCS2303drive.h>
```

```
27
28 RMCS2303 rmcs;                    //object for class RMCS2303
29
30 SoftwareSerial myserial(2,3);     //Software Serial port For Arduino
       Uno. Comment out if using Mega.
31
32 int speed_rpm=3 ;
33
34 //parameter Settings "Refer datasheet for details"
35 byte slave_id=7;
36 int INP_CONTROL_MODE=257;
37 int PP_gain=32;
38 int PI_gain=16;
39 int VF_gain=32;
40 int LPR=334;    //lines per rotation
41 int acceleration=0;
42 int speed=speed_rpm*LPR;
43
44
45 long int Current_position;
46 long int Current_Speed;
47
48 void setup()
49 {
50    rmcs.Serial_selection(1);       //Serial port selection:0-Hardware
        serial,1-Software serial
51    rmcs.Serial0(9600);             //Set baudrate for usb serial to
       monitor data on serial monitor
52    Serial.println("RMCS-2303 Speed control mode demo\r\n\r\n");
53
54    //rmcs.begin(&Serial1,9600);    //Uncomment if using hardware
       serial port for mega2560:Serial1,Serial2,Serial3 and set baudrate.
        Comment this line if Software serial port is in use
55    rmcs.begin(&myserial,9600);     //Uncomment if using software
       serial port. Comment this line if using hardware serial.
56    rmcs.WRITE_PARAMETER(slave_id,INP_CONTROL_MODE,PP_gain,PI_gain,
       VF_gain,LPR,acceleration,speed);    //Uncomment to write
       parameters to drive. Comment to ignore.
57    rmcs.READ_PARAMETER(slave_id);
58
```

```
}

void loop(void)
{
   Serial.print("Sending speed command (RPM):");
   Serial.println(speed_rpm);
   rmcs.Speed(slave_id,speed);                    //Set speed within
    range of 0-65535 or 0-(maximum speed of base motor)
   rmcs.Enable_Digital_Mode(slave_id,0);         //To enable motor in
    digital speed control mode. 0-fwd,1-reverse direction.

//   delay(3000);
   Current_Speed=rmcs.Speed_Feedback(slave_id);
   Serial.print("Current Speed feedback (RPM): ");
   Serial.println(Current_Speed/LPR+0.0);

//   delay(5000);
//   Serial.println("Break Motor");
//   rmcs.Brake_Motor(slave_id,0);                  //Brake motor. 0-
    fwd,1-reverse direction.
//
//   delay(2000);
//   Serial.println("Changing direction");
//   rmcs.Enable_Digital_Mode(slave_id,1);           //To enable
    motor in digital speed control mode. 0-fwd,1-reverse direction.

//   delay(3000);
//   Current_Speed=rmcs.Speed_Feedback(slave_id);
//   Serial.print("Current Speed feedback : ");
//   Serial.println(Current_Speed);


//   delay(3000);
//   Serial.println("Disable Motor");
//   rmcs.Disable_Digital_Mode(slave_id,1);         //Disable motor in
     digital speed control mode. 0-fwd,1-reverse direction.
//   delay(3000);
}
```

REFERENCES

1. (2017). *MATLAB version 9.3.0.713579 (R2017b)*. The Mathworks, Inc., Natick, Massachusetts.

2. **A. Ghosh** and **A. K. Mallik**, *Theory of mechanisms and machines*. Affiliated East-West Press Private Limited, 2002.

3. **M. Hohenwarter**, **M. Borcherds**, **G. Ancsin**, **B. Bencze**, **M. Blossier**, **A. Delobelle**, **C. Denizet**, **J. Éliás**, **A. Fekete**, **L. Gál**, **Z. Konečný**, **Z. Kovács**, **S. Lizelfelner**, **B. Parisse**, and **G. Sturr** (2013). GeoGebra 4.4. http://www.geogebra.org.

4. **R. L. Norton**, *Kinematics and dynamics of machinery*. McGraw-Hill Higher Education, 2011.

5. **Python Core Team** (2018). *Python: A dynamic, open source programming language*. Python Software Foundation. URL https://www.python.org/.

6. **J. J. Uicker**, **G. R. Pennock**, **J. E. Shigley**, *et al.*, *Theory of machines and mechanisms*, volume 1. Oxford University Press New York, NY, 2011.