Small Campaign,Outfield and Uneven Terrain(S.C.O.U.T) Walking Robot

"A four legged robot for motion on different terrains"

Submitted by

Amad Ud Din Gakkhar	*2012053*
Muhammad Saad Najib	*2012327*
Syed Zeeshan Ahmed Zaidi	*2012389*
Alina Moin	*2012902*

Advisor: Dr. Zia Ul Haq Abbas

Co- Advisor: Mr. Mazhar Javed

Faculty of Electrical Engineering

GIK Institute of Engineering Sciences and Technology

Topi, District Swabi, Khyber Pakhtunkhwa, Pakistan.www.giki.edu.pk

April 2016

“Behold! In the creation of the heavens and the earth, and the alternation of Night and Day- there are indeed signs for men of understanding”

[Qur’an 3:190]

Certificate of Ownership

It is certified that the work contained in Final Year Project Report "*Small Campaign Outfield Uneven Terrain Walking Robot"* has been carried out through the collective efforts of Amad Ud Din Gakkhar, Muhammad Saad Najib , Syed Zeeshan Ahmed Zaidi and Alina Moin under the supervision of Dr Zia ul Haq Abbas and Mr. Mazhar Javed ,through extensive research ,internet , library and diverse expert opinion.

Advisor: Dr. Zia ul Haq Abbas

Co-Advisor: Mr. Mazhar Javed

Dean FEE: Dr. Nisar Ahmed

Preface

This Final Year Project Report is the detailed explanation of designing a four legged robot that is capable of moving on an uneven terrain.

It comprises of portions that elaborate the creation of mechanical body, development of electronic circuits, interfacing of sensors and complete software developments that was carried out in process of designing the robot. Furthermore, it includes practical applications that legged robots are best suited for. All possible future enhancements that could be done to revamp the present version are also mentioned.

Abstract

We have designed a Four Legged Robot with each leg having three degrees of freedom. Legged Robotics is inspired from nature for fast, efficient and versatile movement. We have achieved it through coordinated movement of joints, which enables adaptation to environment and improved mobility. Our Objective was to replicate the animal motion and increase mobility as compared to wheeled robotics.

.

Acknowledgements

Any project of such a vast scope cannot possibly be done by four people. We would like to thank all those who have put in their efforts in order to make the project reach the point of completion.

Firstly, we are thankful to Dr. Nisar Ahmed , Dean Faculty of Electronic Engineering GIK institute for granting us approval. Next we would thank our adviser Dr Ziaul haq Abbas and co-adviser Mr. Mazhar Javed for their guidance and constant support without which we could not have gone so far.

Furthermore, we would like to thank Mr. Khurshid , incharge computer numerical control (CNC) lab for the fabrication of mechanical structure.

Last but not the least, we wish to acknowledge the efforts of Mr. Talha Altaf , senior year student Faculty of Mechanical Engineering for the software development of the mechanical design and Mr. Anas bin Zubair, senior year student Faculty of Electronic Engineering for benefiting the project through his programming expertise.

Table of Contents

Chapter 1: Introduction

1.1 Objective

- Our objective was to design a four legged robot which is capable of moving on an uneven terrain.
- The mechanical design of the robot should resemble the ***Pronograde Posture*** (walking with the body parallel to ground) of an animal. The servo motors attached to the joints in all four legs will help in achieving ***Quadrupedal mode of locomotion*.**
- To develop a gait pattern that could maintain the stability of the robot with each deliberated movement.
- A camera should be mounted at the top to display live video during locomotion.
- Condition monitoring i.e. displaying the outputs of accelerometer, temperature sensor and current sensor on GUI and Wireless Communication.

1.2 Overview of the study

The FYP report carries detailed description of mechanical design & its fabrication, electronic circuitry and all hardware and software developments. Besides these, functionalities of all components used in the making of robot are elaborated along with schematics & relevant figures. It gives reader an idea about the series of sequential steps that were taken in order to meet the objective in the limited span of time. All algorithms / codes are provided in appendix.

1.3 Significance of the study

Legged Robots are capable of moving on irregular terrains because of their ability to vary their legs configuration. Therefore legs are inherently adequate systems for locomotion on uneven terrains. The area of contact between the foot and the surface on which the robot is moving can be made in such a way that the ground support pressure is significantly small as compared to wheeled robots. Multiple Degrees of Freedom in the leg joints legged makes legged robot capable of changing the direction of heading without any slippage. The height of robot body can be varied. Damping and decoupling effect between terrain irregularities and the body can be introduced. As a result of which legged robots exhibit superior mobility on naturally occurring terrains.

Chapter 2: Review of Literature

Legged robotics is a relatively new innovation in the field of robotics. Their main advantage over wheeled robotics is the level of mobility they provide. Some of the early attempts were made in this field by MIT leg laboratory. Boston Dynamics is another research center which has done an extensive research and experimental work in this field.

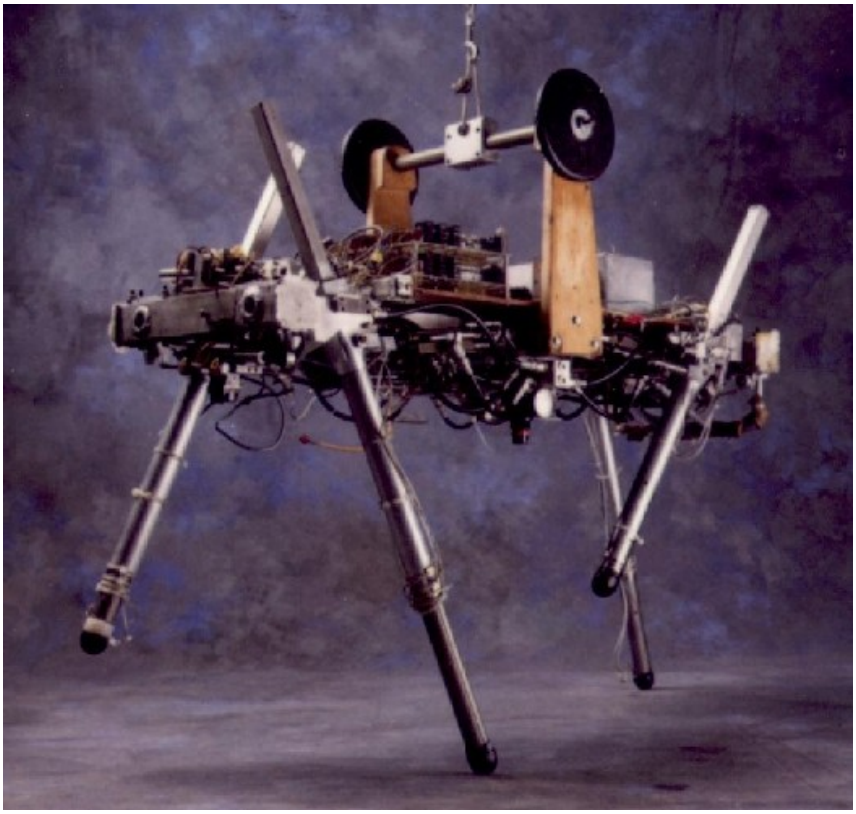


Fig.2.1 Quadruped (1984-1987) - **MIT Leg Laboratory**

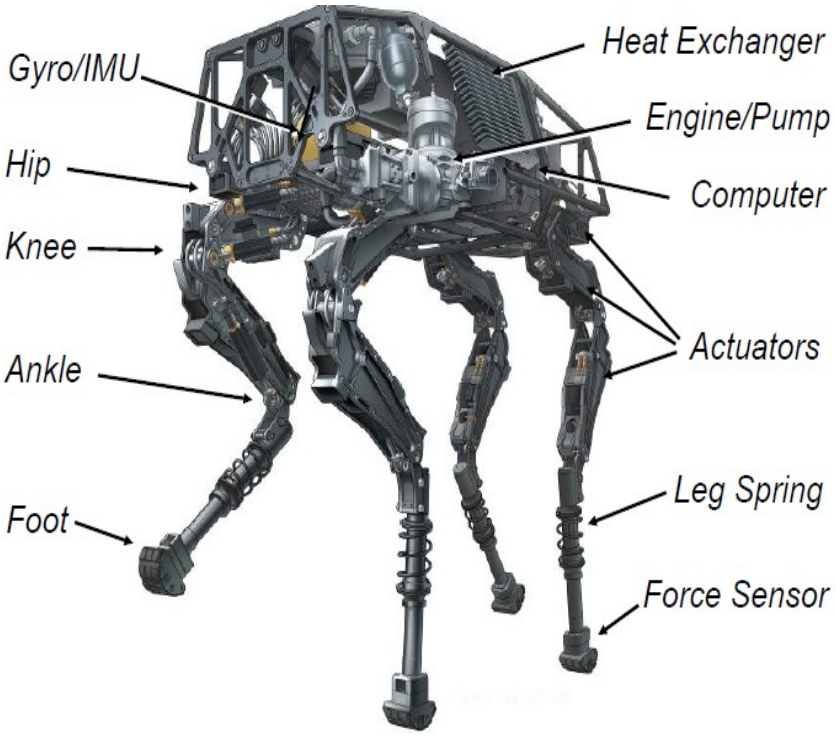


Fig.2.2 **BigDog** - Boston Dynamics and US Military

Robots having four legs are preferred because of the inherent stability associated with four legs and less complexity as compared to five or six legged robots. This is the main reason we have opted to design a four legged robot. Our robot is a prototype four legged robot and is different from most of the quadruped robots in the selection of actuators. We have used servos instead of linear actuators because of their low cost and in-built feedback system.

2.1 Background

Co-ordinated Movements of joints constitutes the gait patterns of robot. There can be two types of gait patterns [1].

- Static Stable Gait pattern
- Dynamic Stable Gait pattern

In static Stable gait pattern there must always be at least a three point contact with ground. Only one leg is in the air at a time. This type of Gait pattern is very slow but provides an easy implementation.

In dynamic stable gait pattern 2 or more legs move at a time. This type of gait pattern is very efficient and fast as compared to static stable gait pattern. Most mammals use this type of gait pattern. Trot and Pace are two types of leg movements which exhibit dynamic stable gait pattern [1-2].

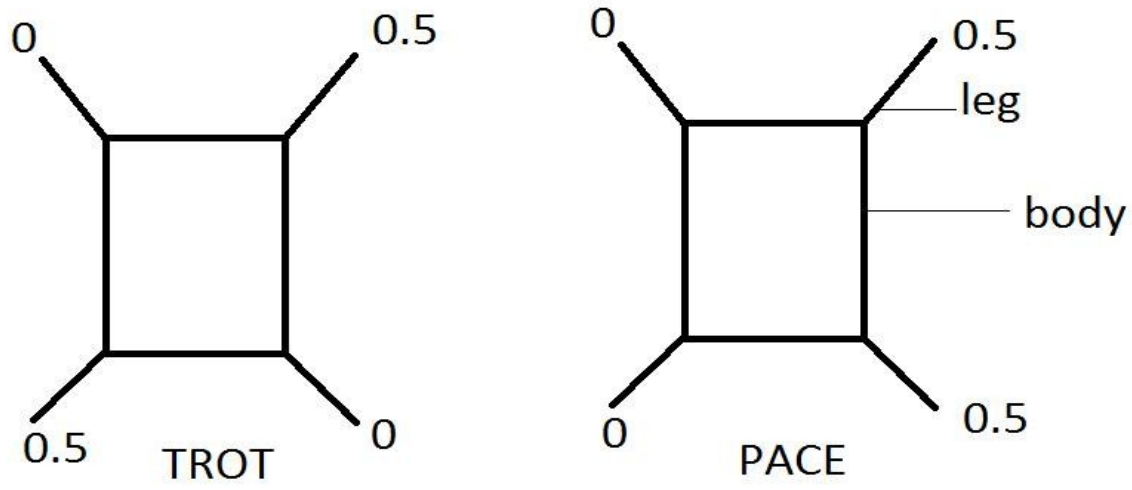


Fig.2.3 Gait Patterns

Stability admitting lines (SALs)[3] is another important concept in legged robotics. Lines that connect leg 1 and 3 and 2 and 4 are termed as stability admitting lines. They divide the plane into four quadrants. Position of COG defines the leg movement.

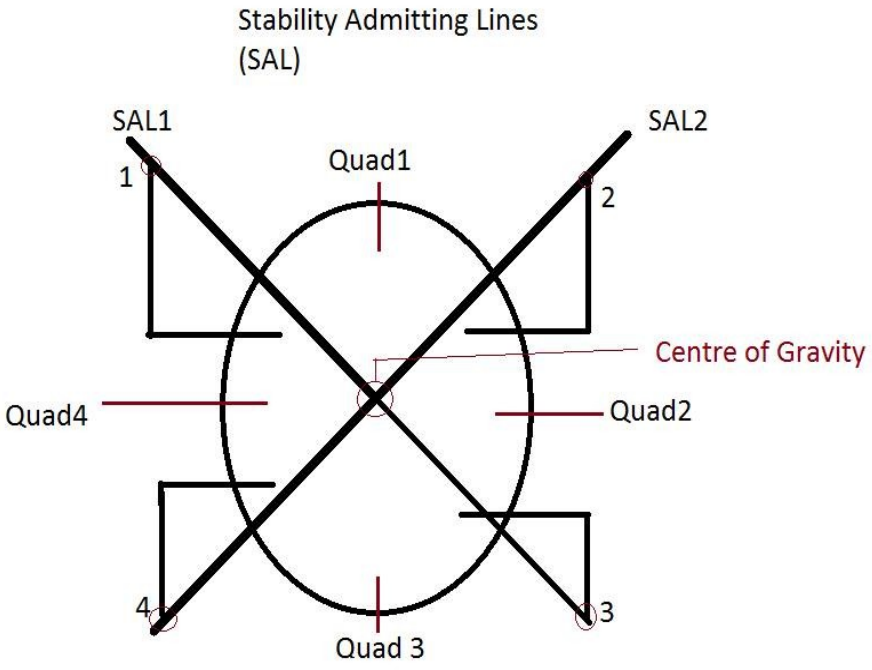


Fig.2.4 Stability Admitting Lines

2.2 Detailed Description of Components

2.2.1 Temperature Sensor (LM 35)

LM35 is a very easily available and cheap temperature sensor. It is used in project for condition monitoring of the robot as the circuits tend to heat up during long lasting operation.

It is precision integrated-circuit temperature sensor. The output voltage varies linearly with temperature on centigrade scale. Its advantage over linear temperature sensors calibrated in Kelvin is primarily because subtracting a large value of constant voltage from the output is not required in the process of extracting centigrade scaling. Furthermore, LM 35 does not need any kind of external calibration to give accuracies of ± 0.25 °C at room temperature and ±0.75 °C on a full −55°C to 150°C range.

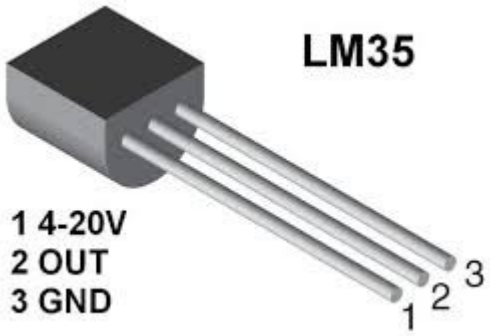


Fig.2.5 LM35 - Temperature Sensor

2.2.2 Servo Motors

We have used servo motors as actuators in this project. Normally the actuators used in such kind of a robot are linear actuators with much more load capacity. As ours was a prototype project, we preferred servo motors because of our budget limitations and their inherent feedback system. We have used Tower Pro-MG996 for lower joints and Power HD 1501-MG for upper joints because of different torque requirements.

Servo motors are assembled with following four things

- DC motor
- Gearing set
- Control circuit
- Position Sensor

The position in case of a servo motor can be controlled with more precision as compared to those of standard DC motors. Usually they have three wires

- Power
- Ground
- Control

The angle of rotation is confined in the range of 0 to 180 Degrees. The servo motors receive a control signal which indicates the output position and keeps supplying power to the DC motor until the shaft has rotated to the required position. It is sensed by the position sensor. Pulse width modulation is used for the control signal of servo motors. Once the servo shaft has reached the final position it shall stop immediately irrespective of the resistive forces acting on it and the maximum intensity of resistive force the servo can exert is defined as the torque rating.

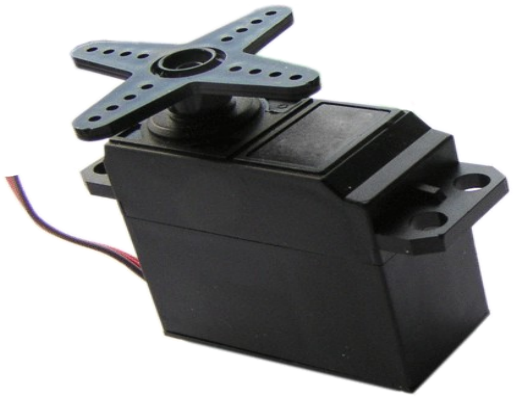


Fig.2.6 Servo motor

2.2.3 Arduino

We have use d Arduino for real time sensor conditioning purpose. As it provides an easy way to take sensor readings using an inbuilt ADC, sensor readings can be easily processed and provided to Raspberry Pi via serial communication.

Arduino is open-source hardware, open-source software, and microcontroller-based kits used for creating digital devices and interactive objects. It is primarily based on microcontroller board. There are sets of digital and analog I/O pins that can be conveniently interfaced with various expansion boards and many other circuits. Arduino board bears serial communication interfaces which may include Universal Serial Bus (USB) in order to load different programs from PCs. To program the microcontrollers, the Arduino boards provide the user with an integrated

development environment (IDE) based on a programming language called Processing. It supports both C and C++ languages.

2.2.4 Raspberry Pi

Raspberry Pi has been selected for control algorithm implementation primarily because of its high speed performance. It provide an ideal platform for future enhancements like image processing etc.

Raspberry Pi is basically a series of credit card–sized single-board computers. CPU speed lies in the range of 700 MHz to 1.2 GHz for the Pi 3 whereas on board memory lies in the range of 256 MB to 1 GB RAM. Secure Digital SD cards are capable of storing the operating system as well as the program memory. Most of the boards contain USB slots, HDMI and composite video output, 3.5 mm phono jack for audio. The Raspberry Pi makes use of Linux-kernel-based operating systems.

Fig.2.7 Raspberry Pi

2.2.5 Current Sensor

The ACS712 current sensor measures up to 30A of DC /AC current. ACS712 Low Current Sensor Breakout gives analog voltage as an output signal. It shows linear variation with sensed current. In order to calibrate the current sensor, output offset is initially set to the desired level. A known value of current input is taken, output deflection is set in accordance with the gain pot. Sensitivity is determined through the following formula

$$\frac{¿\left(V_{ref}-V_{deflect}\right)}{input\,current}$$

The bandwidth is kept at 34Hz in order to cut down noise when used at high values of gain. Full bandwidth of 80 kHz can be recovered in case the C1 is completely removed.

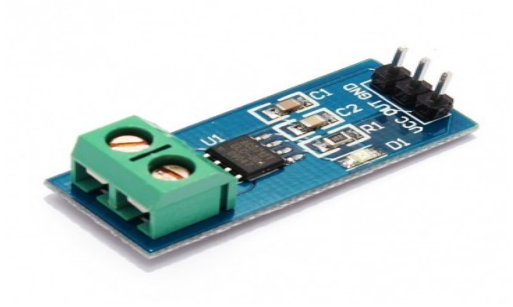


Fig.2.8 ACS-712

2.2.6 Accelerometer

It is an electromechanical device that is used to measure acceleration forces. The acceleration forces are of two types. They could be static or dynamic. After measuring the amount of static acceleration due to gravity, the measure of angle the device is tilted at with respect to the earth can be determined. On sensing the magnitude and direction of dynamic acceleration, we can clearly analyze the way the accelerometer is moving.

Following are the types of accelerometers:

- Mechanical Accelerometer
- Capacitive Accelerometer
- Piezoelectric Accelerometer

Fig.2.9 ADXL-335

2.2.6.1 Tilt Sensing

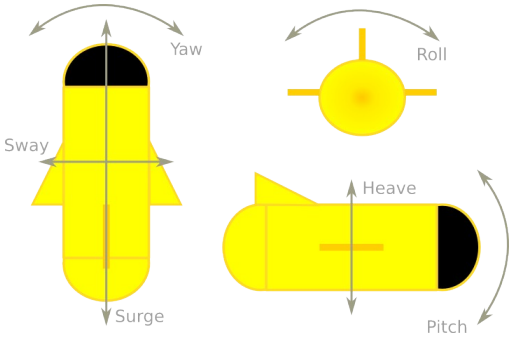


Fig.2.10 Roll,Pitch,Yaw

Accelerometers are used in tilt sensing applications. As it is clear in above given picture roll is clockwise or anti clock wise motion whereas pitch in up and down motion. The values of linear acceleration along all three axes help to determine tilt angles.

We have employed in our project ADXL335. ADXL335 is a capacitive accelerometer. It is a complete 3 axis accelerometer and it has conditioned output voltages which are easier to interface. It has a range of ±3g.

2.2.7 7806 Voltage Regulator

7806 (voltage regulator integrated circuit) belongs to 78xx series of fixed linear voltage regulator ICs. Voltage source in a circuit have fluctuations and fails to give the fixed voltage output. A voltage regulator IC basically keeps the value of V_{out} constant. The variable xx in 78xx denotes output voltage it is meant to provide the user with. 7806 is used to provide +6V of regulated power. Several capacitors of appropriate capacitance can be plugged in the input and output pins. Their capacitance depends upon the respective levels of voltage.

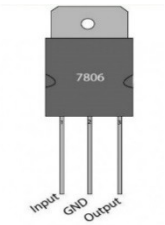


Fig.2.11 7806

Chapter 3: Analysis and Design

_

3.1 Basic working principle

The main problem in making the robot walk was the weight of the base. We had to shift the weight before each leg motion or else the robot would fall down. To overcome with this issue we applied weight shifting technique according to which, before pulling any leg off the ground the weight was distributed on the other three legs such that the robot is balance on just three legs. This solved our weight shifting problem and we could easily lift any leg of the robot without making it fall.

The next problem was to make a pattern of legs motion that can be run in a loop for continuous walk of the robot. Since we were going for sensor less control of the robot, this task was very difficult to achieve. After trying a number of gait patterns we came across one that could be used. The trick was to bring legs at same position as they were at the start of the motion. This resulted in reduced step size but we were able to run the code in loop.

3.2 Mechanical Design

The Mechanical Design was inspired from HyQ- a Hydraulically and Electrically Actuated Quadruped Robot [4]. Throughout the project we tried to keep the mechanical portion of the project simple and used easily available materials. There were two options in consideration for the material of the body - Aluminum or Acrylic. Both of them have their own merits and demerits. Both of the materials are light weight. Aluminum is much stronger but according to our requirements we didn't need strength as much, as we needed less weight because of the limitations of the actuators. So we decided to go with acrylic because of its light weight and easy machinability.

3.2.1 Leg Design:

Legs of the robot were designed by keeping in mind the core requirement that they must have three degrees of freedom and the limitation of time. Each Leg consists of three parts. Lower Leg, Upper Leg and link. Upper leg and lower leg are very similar in shape. They vary in length and thickness. Four bearings are installed in Upper Leg (two at each end) so that lower leg, upper leg and link can freely rotate with respect to each other.

3.2.2 Base Design:

Base of the robot is a rectangular sheet of 8mm thick acrylic sheet. Its dimensions are 300x200mm. Aluminum angle irons are used for inter-connections between legs and base. Legs are connected to base via another bearing to allow rotation in plane perpendicular to the plane of motion of other two joints.

3.2.3 Actuators:

Each leg is controlled by three motors. Servo motors are used as actuators because of their in-built feedback system and low cost. 13kg and 17kg torque servos are used. With 13kg torque Servos on lower leg joints and 17 kg torque motors on other joints.

3.3 Electronic Design

In view of our power requirements we had the option to connect our battery directly to the PWM multiplexing boards which would further drive the servo motors. However the voltage rating of the multiplexing board was around 5-6V whereas our battery supplied 12.6V when charged. Hence we had to step down our voltage to supply the board in the safe operating range. For that we used 7806 regulators which gave a normal output of 6V and around 0.8 to 1A.

According to our calculations even if all 13 motors were somehow in stall condition, they would draw maximum 39A current. This assumption was taken up for the sake of safety, overheat conditions and smooth operation of the motors. Given one regulator provides up to 1A, current rating was not enough to drive 13 servo motors, where one servo motor draws around 1.5A on normal operating conditions and 2.5-3A on stall condition. The best option to implement was to cascade the regulators in parallel to enhance the current output and keep the voltage output constant. For that we had to cascade forty 7806 regulators to meet the above mentioned conditions.

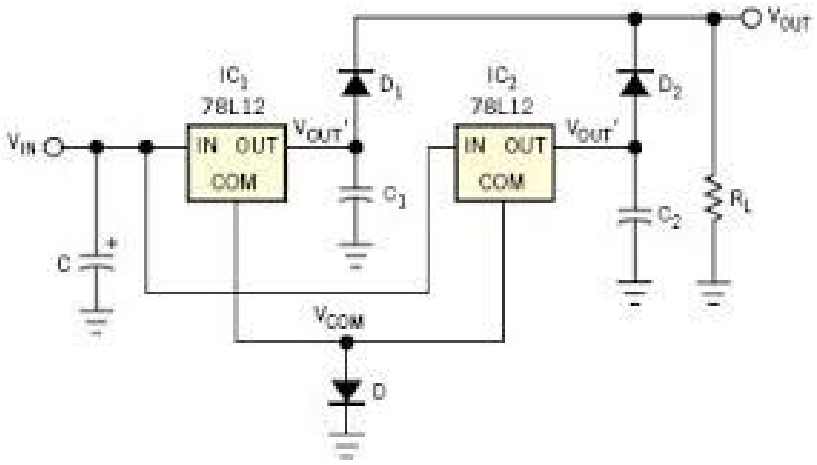


Fig.3.1 Parallel Voltage Regulators

The picture above shows two regulators cascaded in parallel.

The problem encountered when cascading the regulators was due to inherent variations in the manufacturing of the regulators. Not all regulators give the exact rating voltage but instead there is a tolerance level and each regulator gives output above or below the mentioned value which we call as mean value. This gives rise to a condition where one regulator with the output value above mean becomes a generator and the other one which is below the mean value becomes a load. To avoid this diode must be connected to the output of each regulator to avoid current flowing back if its voltage output is below mean value. The problem encountered due to this phenomena is that regulators start to overheat when current flows inside them and after a certain period of time becomes dysfunctional causing the circuit to break and hence all regulators fail.

Capacitors of 1000uF are connected at each output of the regulator to control the surges as well as smoothing of the output to avoid damage to the motors and multiplexing boards. A large rating capacitor of 2500uF was connected to the input for the same purposes and to avoid damage to the regulators in case of glitches in transient phase. Therefore, we made 2 separate

boards and in each PCB board we cascaded 20 regulators to supply one side of the robot i.e left side motors or the right sided motors. Each board had the supply capacity of 20A and 6V.

3.3.1 Sensors Interfacing

Different sensors used are:

- LM-35 Temperature Sensor
- ADXL-335 Accelerometer
- ACS-712 Current Sensor

3.3.1.1 Temperature Sensor (LM 35)

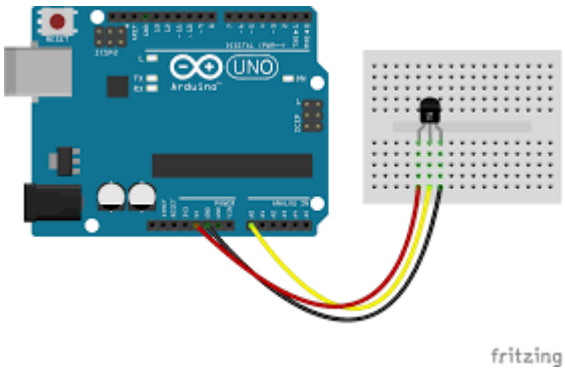


Fig.3.2 LM35 Interfacing

As it is shown in the above given picture, when the flat face of LM 35 is placed in front of you extreme left pin should be connected to 5V ,middle one in the analog pin and extreme right pin to the ground of Arduino.

The output gives the temperature in Celsius which is converted in Fahrenheit using the following formula:

$$temperature\ coversion ¿ celsius ¿ fahrenheit = \frac{9}{5}C + 32$$

The output is serially communicated to raspberry pi as it lacks an Analog to digital conversion (ADC). The temperature sensor outputs the temperature of the connected circuits and it is displayed on GUI.

3.3.1.2 Accelerometer

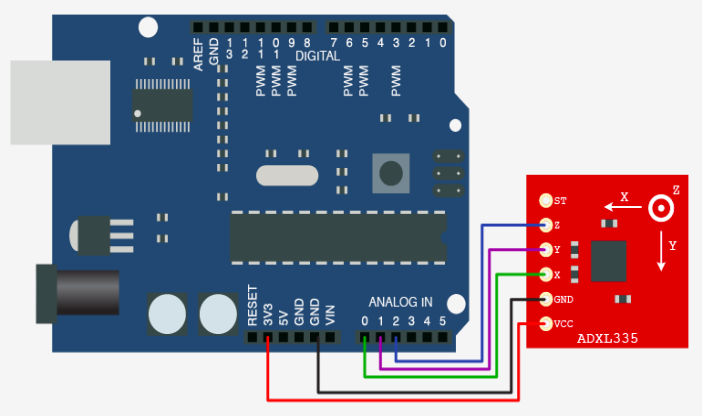


Fig.3.3 ADXL335 Interfacing

ADXL 335 is a low power complete three axis accelerometer with signal conditioned output voltages. It operates at 3.3 V.

Pin Configuration:

V cc : it is connected to 3.3 V of Arduino

GND: it is connected to ground pin

X: it is connected to analog pin

Y: it is connected to analog pin

Z: it is connected to analog pin

The output is raw value of linear accelerations along x , y & z axes . After conditioning, it is plugged in the following formulas to determine roll and pitch:

$$roll = \arctan\left(-G_x / G_z\right)$$

$$pitch = \frac{G_y}{\sqrt{G_x^2 + G_z^2}}$$

Range:

Pitch: [-90, 90]

Roll: [-180, 180]

The respective values of roll & pitch are displayed on GUI through serial communication of data from Arduino to raspberry pi.

3.3.2 Graphic User Interface (GUI)

GUI of the robot is made using Tkinter library of Python. Tkinter is a powerful tool for making GUIs in Python.

We have made the GUI on Raspberry Pi's operating system. We access the desktop of raspberry Pi via SSH client. Robot GUI consists of three buttons which are responsible for different functionalities of robot. It also consists of a Label which contains different sensor reading as received from Arduino. In addition to that controls for camera motion are also included in the GUI.

3.3.3 Serial Communication of Arduino & Raspberry Pi

Arduino and Pi communicate serially via a USB cable. Python library for Raspberry Pi, PySerial is used for this purpose. All the sensors are connected to Arduino which Processes their readings and transmits them serially to Raspberry pi which then shows the readings in GUI. The Serail communication between the two is taking place at 9600bps.

Block Diagram

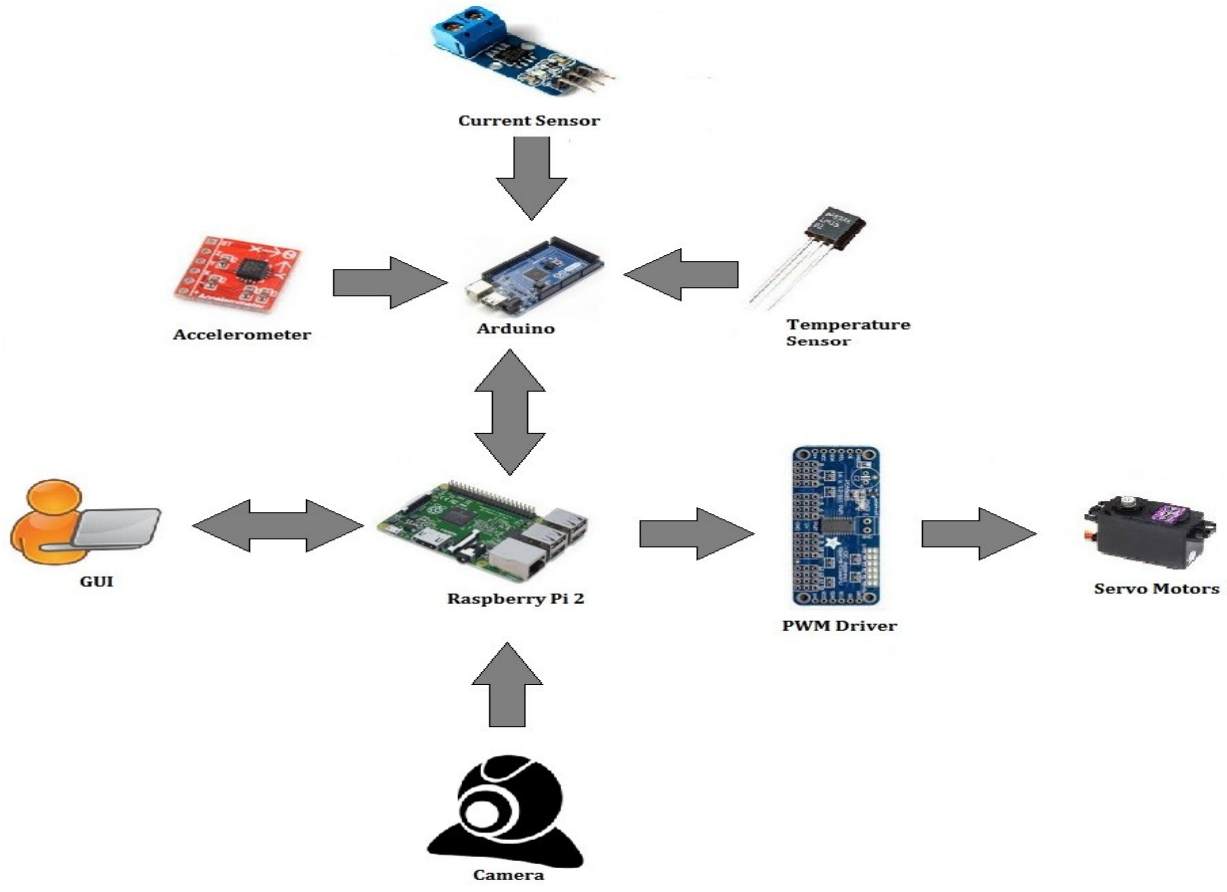


Fig.3.4 Block Diagram

Chapter 4: Conclusion

–

We have successfully designed the robot. We have developed a walking algorithm and have made the robot walk on different terrains like Grass, Road, Concrete and Rocky.

4.1 Main Features

- Robust electronic design
- Wirelessly controlled
- Parameters like temperature , roll ,pitch and current drawn displayed on GUI

4.2 Achievements

- First position in Final Year Project competition in Faculty of Electrical Engineering.
- Successful Implementation of Gait Pattern Algorithm.
- Institute wide Appreciation on Industrial Open House 2016.

4.3 Applications

There is an almost infinite list of possible applications where such high-mobile robots can be deployed:

- Remote Surveillance

- Construction and mining companies for construction site transporter

- Inspection of large scale industrial environments Police etc. for bomb disposal and for exploration Fire brigades

- Exploration in dangerous environment Search and rescue teams

- Operators of polluted facilities

- Security agencies as mobile security guard

- Luggage transporter

- As walking aid for persons with mobility limitations

- Researchers advertising agencies for interactive advertisement

- Entertainment industry e.g. for amusement park attraction

4.4 Future Enhancements

- The Robot we have designed can be improved in many ways for it to have a more versatile motion. Firstly, the robot motion can be made sensors based by designing specialized feet and by using force sensors at their bottom.
- Mechanical Design can be improved by using linear actuators which have a very quick response and have a greater load capacity.
- Image processing can be used for an efficient gait pattern and more stable foothold selection.

References

[1] Sven Böttcher, 'Principles of robot locomotion', Seminar on Human robot interaction, Department of Computer Science, TU Dortmund, 2006.

[2] Daniel J. Pack and A. C. Kak ,"A Simplified Forward Gait Control for a Quadruped Walking Robot", Robot Vision Laboratory, Conference on Intelligent Robots and Systems, 1994.

[3] R. McN. Alexander ,"The Gaits of Bipedal and Quadrupedal Animals", Department of Pure and Applied Zoology University of Leeds.

[4] Claudio Semini*, Nikos G. Tsagarakis *et al.," Design of HyQ – a Hydraulically and Electrically Actuated Quadruped Robot"*, Department of Advanced Robotics, Italian Institute of Technology, Genoa, Italy, April 2010.

Appendix A

Python Code:

```
from Tkinter import *
import serial
from PIL import ImageTk,Image
import os
import pygame
import random

from Adafruit_PWM_Servo_Driver import PWM
import time
import getch
from operator import pos

pwmR = PWM(0x40)   #I2C Address from Pie for right motors
pwmL=  PWM(0x41)   #I2C Address from Pie for left motors
ser = serial.Serial('/dev/ttyACM0', 9600)


class Motor:
   def __init__(self,num,mean,rl):

      self.mean=mean
      self.num=num
```

```
        self.rl=rl

    def gotoMean(self):
        if self.rl=='r':
            pwmR.setPWM(self.num,0,self.mean)
        elif self.rl=='l':
            pwmL.setPWM(self.num,0,self.mean)

    def move(self,pos,x,fb):

        if x=='r2' and fb=='f':
            self.pos=self.mean+pos
        if x=='r2' and fb=='b':
            self.pos=self.mean-pos

        if x=='l2' and fb=='f':
            self.pos=self.mean-pos

        if x=='l2' and fb=='b':
            self.pos=self.mean+pos

        if x=='r3' and fb=='f':
            self.pos=self.mean-pos

        if x=='r3' and fb=='b':
            self.pos=self.mean+pos

        if x=='l3' and fb=='f':
            self.pos=self.mean+pos

        if x=='l3' and fb=='b':
            self.pos=self.mean-pos




        if self.rl=='r':
            pwmR.setPWM(self.num,0,self.pos)
        elif self.rl=='l':
            pwmL.setPWM(self.num,0,self.pos)


def step(event):
    global start
    global stop

    start=12
```

```
stop=17


d2=60
d3=120



RF2.move(d2,'r2','f')
RF3.move(d3,'r3','b')

LF2.move(d2,'l2','f')
LF3.move(d3,'l3','b')

RB2.move(100,'r2','f')
RB3.move(150,'r3','b')

LB2.move(d2,'l2','f')
LB3.move(d3,'l3','b')



time.sleep(1)

LF2.move(200,'l2','f')          ###LEG 1 MOTION
time.sleep(0.3)
LF3.gotoMean()
time.sleep(0.3)
LF2.move(100,'l2','f')

time.sleep(1)
RB2.move(40,'r2','f')
RB3.move(80,'r3','b')
time.sleep(0.5)


# getch.getch()



time.sleep(0.5)

RF2.move(40,'r2','f')
RF3.move(100,'r3','b')
```

```
LB2.move(40,'l2','f')
LB3.move(100,'l3','b')


RB2.move(40,'r2','f')
RB3.move(80,'r3','b')


LF2.move(60,'l2','f')
LF3.move(120,'l3','b')









time.sleep(1)
# getch.getch()

RB3.move(200,'r3','b')
RB2.move(200,'r2','f')        ###LEG 3 MOTION

time.sleep(0.3)
# RB3.gotoMean()
# RB3.move(250,'r3','b')

time.sleep(0.3)

RB3.move(120,'r3','b')
RB2.move(60,'r2','f')


time.sleep(1)


LF2.move(60,'l2','f')
LF3.move(120,'l3','b')

RB2.move(60,'r2','f')
RB3.move(80,'r3','b')


LB2.move(100,'l2','f')
LB3.move(150,'l3','b')
```

```
time.sleep(1)

RF3.move(180,'r3','b')
RF2.move(200,'r2','f')        ###LEG 2 MOTION
time.sleep(0.3)
# RF3.gotoMean()
# time.sleep(0.3)


RF2.move(60,'r2','f')
RF3.move(120,'r3','b')



time.sleep(1)



                #wt shift



RF2.move(100,'r2','f')
RF3.move(180,'r3','b')

LB2.move(40,'l2','f')
LB3.move(80,'l3','b')


LF2.move(60,'l2','f')
LF3.move(120,'l3','b')


RB2.move(40,'r2','f')
RB3.move(120,'r3','b')




time.sleep(0.3)

time.sleep(0.5)
```

```
    # getch.getch()

    LB3.move(200,'l3','b')
    LB2.move(100,'l2','f')          ###LEG 4 MOTION
    time.sleep(0.3)
    # RF2.move(60,'r2','f')
    # RF3.move(120,'r3','b')


    # LB3.gotoMean()
    # time.sleep(0.3)



    LB2.move(60,'l2','f')
    LB3.move(120,'l3','b')

    start=8
    stop=9

def bend(event):
    global start
    global stop

    start=12
    stop=17
    for x in range(1,60):

        RF2.move(x,'r2','f')
        RF3.move(2*x,'r3','b')

        LF2.move(x,'l2','f')
        LF3.move(2*x,'l3','b')

        RB2.move(x,'r2','f')
        RB3.move(2*x,'r3','b')

        LB2.move(x,'l2','f')
        LB3.move(2*x,'l3','b')



    for x in reversed(range(1,60)):
```

```
        RF2.move(x,'r2','f')
        RF3.move(2*x,'r3','b')

        LF2.move(x,'l2','f')
        LF3.move(2*x,'l3','b')

        RB2.move(x,'r2','f')
        RB3.move(2*x,'r3','b')

        LB2.move(x,'l2','f')
        LB3.move(2*x,'l3','b')
        start=8
        stop=9


def alltomean(event):
    LF1.gotoMean()
    LF2.gotoMean()
    LF3.gotoMean()
    LB1.gotoMean()
    LB2.gotoMean()
    LB3.gotoMean()

    RF1.gotoMean()
    RF2.gotoMean()
    RF3.gotoMean()
    RB1.gotoMean()
    RB2.gotoMean()
    RB3.gotoMean()

camPos=375
pwmR.setPWM(11,0,camPos)

def cameraRight(event):

    global camPos
    camPos=camPos+10
    if camPos >=600:
        camPos=600

    pwmR.setPWM(11,0,camPos)

def cameraLeft(event):
    global camPos
    camPos=camPos-10
    if camPos <=150:
```

```
      camPos=150
   pwmR.setPWM(11,0,camPos)


F1=0
F2=3
F3=6
B1=9
B2=12
B3=15

LF1=Motor(F1,470,'l')
LF2=Motor(F2,440,'l')
LF3=Motor(F3,390,'l')
LB1=Motor(B1,375,'l')
LB2=Motor(B2,300,'l')
LB3=Motor(B3,370,'l')

RF1=Motor(F1,360,'r')
RF2=Motor(F2,380,'r')
RF3=Motor(F3,340,'r')
RB1=Motor(B1,375,'r')
RB2=Motor(B2,360,'r')
RB3=Motor(B3,375,'r')

C1=Motor(11,375,'r')

def Disp(event):
   print("U clicked a button")



root=Tk()

root.geometry("1111x675+300+300")
root.title("S.C.O.U.T")

im = Image.open('/home/pi/Adafruit/PWM/bg3.gif')
tkimage = ImageTk.PhotoImage(im)
bgnd=Label(root,image = tkimage)
bgnd.place(x=0, y=0, relwidth=1, relheight=1)
```

```
canvas_1=Canvas(bgnd,width=400,height=100)
canvas_1.create_image(0,0,image=tkimage,anchor=N)
txt=canvas_1.create_text(200,50,text="S.C.O.U.T \n Control
Pannel",fill="White",activefill="Red",font="Helvetica 30",justify="center")

canvas_1.grid(row=0,column=3,sticky=E)


buttonframe=Frame(bgnd)
buttonframe.grid(row=3,column=3,sticky=NE)

button_1=Button(buttonframe,width=100,height=100,text="Step",fg="white",image=tkim
age,compound=CENTER)

button_1.bind("<Button-1>",step)
button_1.pack(side=RIGHT)

button_2=Button(buttonframe,width=100,height=100,text="Mean",image=tkimage,fg="w
hite",compound=CENTER)

button_2.bind("<Button-1>",alltomean)
button_2.pack(side=LEFT)
#
button_3=Button(buttonframe,width=100,height=100,text="Bend",image=tkimage,fg="wh
ite",compound=CENTER)

button_3.bind("<Button-1>",bend)
button_3.pack()


cameraButton=Frame(bgnd)
cameraButton.grid(row=2,column=0)

cbutton_1=Button(cameraButton,width=100,height=100,text="Right",fg="white",image=t
kimage,compound=CENTER)
cbutton_1.bind("<Button-1>",Disp)
cbutton_1.pack(side=RIGHT)

cbutton_2=Button(cameraButton,width=100,height=100,text="Left",fg="white",image=tki
mage,compound=CENTER)
cbutton_2.bind("<Button-1>",Disp)
cbutton_2.pack(side=LEFT)

cbutton_1.bind("<Button-1>",cameraRight)
```

```
cbutton_2.bind("<Button-1>",cameraLeft)

paraframe=Frame(bgnd)
paraframe.grid(row=3,column=3,sticky=N)

paraframe2=Frame(bgnd)
paraframe2.grid(row=3,column=3,sticky=S)

temp=Label(paraframe,text="Temperature",font="Helvetica
15",justify="center",padx=10)
curr=Label(paraframe,text="Current Drawn",font="Helvetica
15",justify="center",padx=10)
ro=Label(paraframe,text="Roll",font="Helvetica 15",justify="center",padx=10)
pi=Label(paraframe,text="Pitch",font="Helvetica 15",justify="center",padx=10)

temp2=Label(paraframe2,font="Helvetica 15",justify="center",padx=10)
curr2=Label(paraframe2,font="Helvetica 15",justify="center",padx=10)
ro2=Label(paraframe2,font="Helvetica 15",justify="center",padx=10)
pi2=Label(paraframe2,font="Helvetica 15",justify="center",padx=10)

temperature=StringVar()
current=StringVar()
roll=StringVar()
pitch=StringVar()




temp.pack(side=RIGHT)
curr.pack(side=RIGHT)
ro.pack(side=RIGHT)
pi.pack(side=RIGHT)

temp2.pack(side=RIGHT)
curr2.pack(side=RIGHT)
ro2.pack(side=RIGHT)
pi2.pack(side=RIGHT)

temp2['textvariable'] = temperature
temperature.set('00')

curr2['textvariable'] = current
current.set('00')

ro2['textvariable'] = roll
```

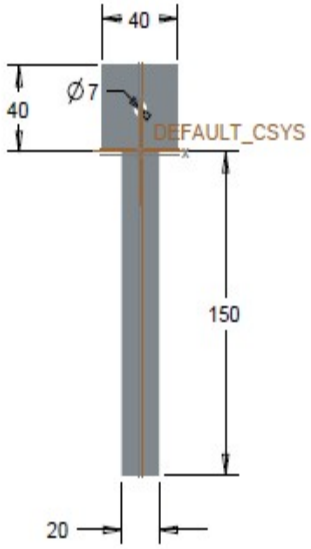
```
roll.set('00')

pi2['textvariable'] = pitch
pitch.set('00')


bgnd.columnconfigure(3, weight=1,pad=10)
bgnd.rowconfigure(2, weight=1,pad=10)

xroll=0
xpitch=0
start=8
stop=9

def updateparam():
   global temperature
   global current
   global roll
   global pitch
   global xroll
   global xpitch
   global start
   global stop

   time.sleep(1)

   ser.write('3')
   xtemp=ser.readline()
   time.sleep(0.5)

   xtemp=float(xtemp)


   ser.write('1')
   xroll=ser.readline()
   time.sleep(0.5)

   xroll=float(xroll)




   ser.write('2')
```

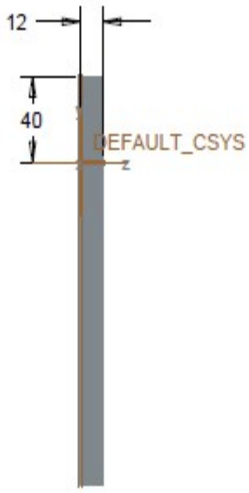
```
    xpitch=ser.readline()
    time.sleep(0.5)
    xpitch=float(xpitch)

    if xtemp !=xpitch and xtemp !=xroll and xtemp >=20:
        temperature.set(xtemp)
    else:
        temperature.set(25.26)

    if xroll != xtemp and xroll != xpitch:
        roll.set(xroll)
    else:
        roll.set(-11.7)
    if xpitch != xroll and xpitch != xtemp:
        pitch.set(xpitch)
    else:
        pitch.set(12.34)

    xcurr=random.uniform(start,stop)


    current.set("%.2f" % xcurr)


    paraframe2.after(500, updateparam)
# def updateroll():
    # global roll

    # time.sleep(1)

    # ser.write('1')
    # xroll=ser.readline()

    # roll.set(xroll)
    # paraframe2.after(1500, updateroll)



updateparam()
# updateroll()


root.mainloop()
```

```
##embed = Frame(root, width = 500, height = 500) #creates embed frame for pygame
window
##embed.grid(columnspan = (600), rowspan = 500) # Adds grid
##embed.pack(side = LEFT) #packs window to the left
##buttonwin = Frame(root, width = 75, height = 500)
##buttonwin.pack(side = LEFT)
##os.environ['SDL_WINDOWID'] = str(embed.winfo_id())
##os.environ['SDL_VIDEODRIVER'] = 'windib'
##screen = pygame.display.set_mode((500,500))
##screen.fill(pygame.Color(255,255,255))
##pygame.display.init()
##pygame.display.update()
##def draw():
##    pygame.draw.circle(screen, (0,0,0), (250,250), 125)
##    pygame.display.update()
##    button1 = Button(buttonwin,text = 'Draw',  command=draw)
##    button1.pack(side=LEFT)
##    root.update()
##
##while True:
##    pygame.display.update()
##    root.update()
##    root.mainloop()
```

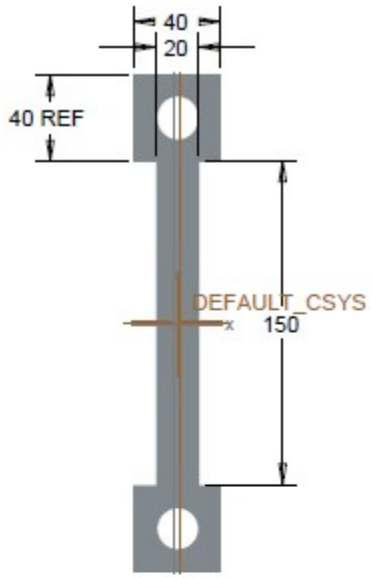
Appendix B

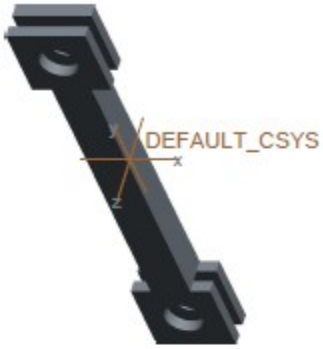
Lower Leg:

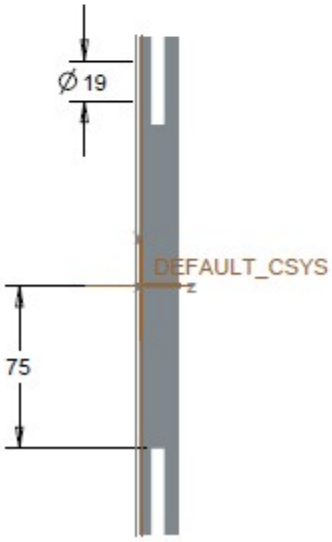




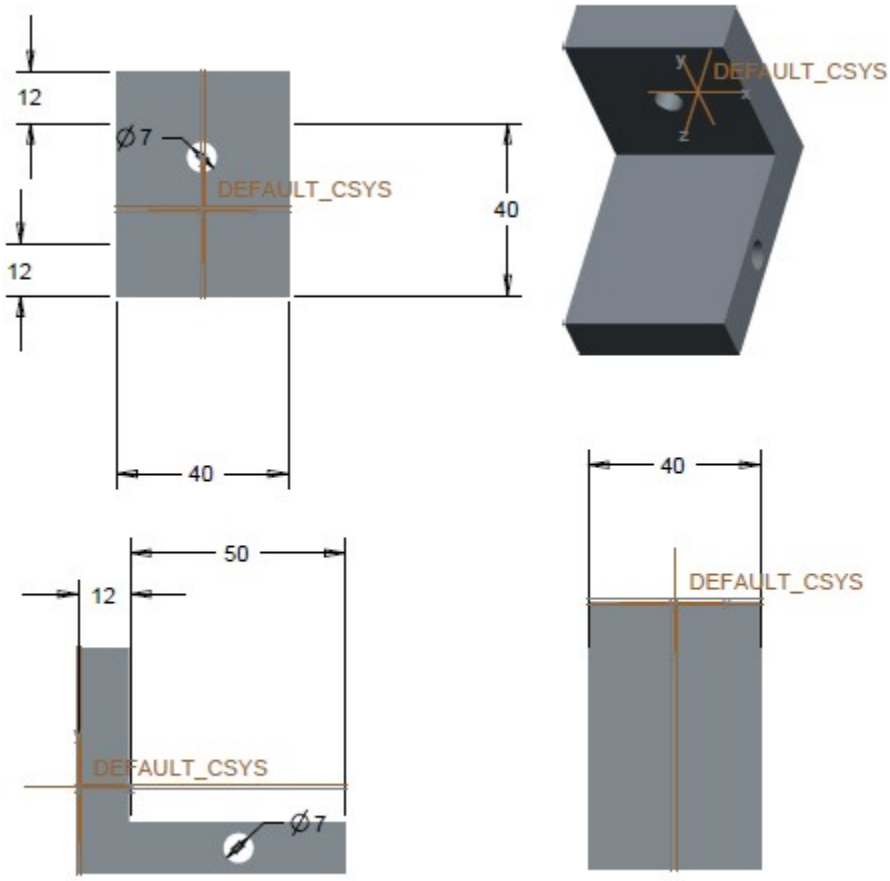
Upper Leg:


40
20
40 REF
DEFAULT_CSYS
150


DEFAULT_CSYS


Ø 19
DEFAULT_CSYS
75

Link:


12
Ø7
DEFAULT_CSYS
40
12
40
DEFAULT_CSYS
50
12
DEFAULT_CSYS
Ø7
40
DEFAULT_CSYS