Cairo University
Faculty of Engineering
Department of Computer Engineering

Forrest

A Graduation Project Report Submitted

to

Faculty of Engineering, Cairo University

in Partial Fulfillment of the requirements of the degree

of

Bachelor of Science in Computer Engineering.

Presented by

Ahmed Mohamed Soliman
Riad Adel Riad
Samir Hosny Mohamed
Omar Samir Touny

Supervised by

Prof. Amr Wassal

15/08/2020

Abstract

Robotics has become one of the most important fields in the recent years, a lot of companies are competing for building various humanoid robots as they can significantly ease the life of humans. Humanoid robots are learning to do the tasks that humans find it hard to in order to help humans in their daily life.
As the main task of humanoid robots is to do the tasks that humans do, they need to learn first how to walk like humans and this is what we do in our project, we teach humanoid bipedal robots how to walk like humans.

Forrest is a walking algorithm for humanoid robots, developed to make a humanoid robot able to walk on different planes without falling. Forrest can walk on flat and inclined planes with inclination angle up to 11.45 degrees. It also allows the user to draw a path for the robot to follow.

The Objective of our project is to make a walking algorithm that can be implemented on humanoid robots to allow these robots to be used in educational fields or hospitals.

Forrest provides the needed algorithms for humanoid bipedal robots for walking on flat planes and inclined planes. It keeps track of the tilting angles of the robot in order to keep itself balanced. It can sense any change in the inclination angle of the plane it is walking on and sends new values to its motors to adapt to this change. It also allows the user to draw a path by using a very simple GUI and allow the user to watch the robot while following that path.

الملخص

ان علم الروبوتات اصبح من اهم المجالات فى السنوات الاخيرة، العديد من الشركات تتنافس لبناء المختلف من الروبوتات على شكل انسان لكى تيسر حياة الناس. الروبوتات على شكل انسان تتعلم كيف تنفذ المهام التى تصعب على الانسان لكى تساعد الانسان فى حياته اليومية.

بما ان المهمة الرئيسية للروبوت على شكل انسان ان تنفذ المهام التى يقوم بها الانسان، اول شئ يجب على الروبوت تعلمه ان يمشى كالانسان وذلك ما ننفذه فى مشروعنا، نحن نعلم الروبوت كيفية المشى كالانسان.

Forrest هو خوارزمية سير للروبوتات على شكل انسان، صمم لكى يجعل الروبوت يمشى على مختلف الاسطح دون السقوط. Forrest يستطيع السير على الاسطح المستوية والمائلة بدرجة انحناء تصل الى 11.45 درجة. كما انه يسمح للمستخدم ان يرسم مسار للروبوت لكى يتبعه.

ان مهمتنا فى المشروع ان نبنى الخوارزمية التى يمكن استخدامها على الروبوتات على شكل انسان لكى يتمكن الانسان من استخدام هذه الروبوتات فى المجالات التعليمية والمستشفيات.

ان Forrest يقدم الخوارزميات اللازمة للروبوتات على شكل انسان التى تجعلها تمشى على الاسطح المستوية والمائلة. الخوارزمية تقوم بتتبع زاوية الميل للروبوت لكى تجعله يحافظ على توازنه. الخوارزمية تستطيع ان تشعر بأى تغيير فى زاوية ميل الاسطح التى يمشى عليها الروبوت وترسل قيم جديدة لمواتير الروبوت لكى يتأقلم على هذا التغيير. انها ايضا تسمح للمستخدم ان يرسم مسار عن طريق واجهة مستخدم بسيطة كما انها تسمح للمستخدم ان يشاهد الروبوت اثناء تتبعه لهذا المسار.

ACKNOWLEDGMENT

First and foremost, we would like to thank our supervisor Prof. Amr Wassal for his encouragement, support, and guidance throughout this project. We also want to thank the teaching staff of Computer Engineering Department at Faculty of Engineering, Cairo University for giving us the needed knowledge and information both within and outside the courses. We learned so much from them on both academic and personal levels.

In addition to that we must thank our families, friends, and loved ones, who have tolerated us in difficult times and shared the joy in the good ones. Your support and encouragement made this journey fun and bearable to all of us.

Table of Contents

List of Figures

List of Tables

List of Abbreviation

CAGR	Compound Annual Growth Rate
DK	Direct Kinematics
DOF	Degrees of Freedom
GUI	Graphical User Interface
IK	Inverse Kinematics
PID	Proportional Integral Derivative
SVG	Scalable Vector Graphics
USD	United States Dollar
WPG	Walking Pattern Generator

List of Symbols

g	Gravity
q	Joint angles

Contacts

Team Members

Name	Email	Phone Number
Ahmed Mohamed Soliman	ahmadalkady1997@yahoo.com	+2 01112258645
Omar Samir Touny	omartouny2@gmail.com	+2 01017993046
Riad Adel Riad	riad.adel22@gmail.com	+2 01066747537
Samir Hosny Mohamed	samir7osny@yahoo.com	+2 01146633721

Supervisor

Name	Email	Number
Prof. Amr Wassal	a_wassal@yahoo.com	+2 01008715559

This page is left intentionally empty

Chapter 1: Introduction

Forrest is a walking algorithm for humanoid robot, developed to make a humanoid robot able to walk on different planes without falling. Forrest can walk on flat and inclined planes with inclination angle up to 11.45 degrees. It also allows the user to draw a path for the robot to follow. This document is organized into many chapters that cover: A brief introduction, Market research and Feasibility study, literature overview, System Design and Architecture, System Testing and Verification, Conclusion and Future work.

1.1. Motivation and Justification

Robotics has become one of the most important fields in the recent years, a lot of companies are competing for building various humanoid robots as they can significantly ease the life of humans. Humanoid robots are learning to do the tasks that humans find it hard to in order to help humans in their daily life. As the main task of humanoid robots is to do the tasks that humans do, they need to learn first how to walk like humans and this is what we do in our project, we teach humanoid bipedal robots how to walk like humans.

1.2. Project Objectives and Problem Definition

The Objective of our project is to make a walking algorithm that can be implemented on humanoid robots to allow these robots to be used in educational fields or hospitals. So we have some challenges facing us to achieve this goal which leads to some objectives that needs to be achieved like: choose the right sensors to fit with the robot model, maintain the balancing of the robot while walking on flat surface, keep the robot balanced on inclined surface or curved one, and walking on a user's specified path.

1.3. Project Outcomes

The outcomes of this project are the walking algorithm that allows humanoid (bipedal) robots to walk on flat and inclined planes as well as a simple GUI to allow the user to draw a path for the robot to follow.

1.4. Document Organization

This document covers all the important details in our project in order to allow interested readers to understand the whole project and build a similar project if they want. It is divided into a group of chapters where each chapter covers some certain topic about the project. Chapter 2 contains market research and feasibility study which shows our targeted customers, competitors and how to effectively penetrate the market and the project's financial analysis. Chapter 3 is literature survey; it provides the needed concepts that helped us to build this project and it focuses on previous work related to our project and how that helped us to choose what are the optimum methods to implement. In chapter 4 we will discuss the system design and architecture of our project with full details. And in chapter 5 we will cover all our testing trials, setups and environment to ensure that project outcomes are realized correctly. At last there is chapter 6 which concludes the whole report and indicates what we intend to do in the future to optimize our project.

Chapter 2: Market Feasibility Study

Humanoid Robot Market size valued at USD 450 million in 2016 and will grow at a CAGR of over 35% from 2017 to 2024 [1]. The global shipments are expected to reach 1 million units by 2024.
The increasing popularity of companion robotic technology in research, healthcare, and hospitality sectors to gain enhanced assistance & operational efficiency is driving the humanoid robot market.

2.1. Targeted Customers

Our target customers are in the healthcare sector as medical assistants & training aids, further adding up to the demand [1]. The rapid development & transformation of the medical infrastructure by implementing robotic & digitally advanced products for enhanced patient care & engagement supports robot deployment. Additionally, changing dynamics in industrial & manufacturing sectors, which include robotic additions to aid employees & workers is adding up to the industry expansion. These products are utilized to perform engineering tasks in various industries to eliminate human involvement in dangerous tasks and allow them to focus on high-value operations [1]. Also, one of our main target customers is in the education sector as the small robots could be used widely for education purposes.

2.2. Market Survey

Market survey is the research and analysis of the market for a particular product/service which includes the investigation into customer inclinations. A study of various customer capabilities such as investment attributes and buying potential. Market surveys are tools to directly collect feedback from the target audience to understand their characteristics, expectations, and requirements [2]. So, in this section we introduce similar products to our project.

2.2.1. UBTECH

UBTECH is a global leading AI and humanoid robotics company founded in 2012. Born out of industry-leading breakthroughs first made in 2008 to digital servos – the core part that enables humanoid robots to move – UBTECH has grown its worldwide research, design, development, manufacturing, sales, and marketing capabilities to successfully launch an entire portfolio of world-class robots, including

consumer humanoid robots, enterprise service robots, and STEM skill-building robots for kids at home with JIMU Robot and in the classroom through UBTECH Education [3]. UBTECH Products are very accurate and has a lot of features but with very high cost $1300 [4].

2.2.2. ROBOTIS

ROBOTIS is a global robot solutions provider and one of the leading manufacturers of robotic hardware. ROBOTIS is the exclusive producer of the DYNAMIXEL brand of all-in-one smart servos. ROBOTIS specializes in the manufacture of robotic hardware and full robot platforms for use in all fields of study and industry, as well as educational robotics kits for all ages and skill levels [5]. ROBOTIS products are very good for basic educational purposes but could only move on a flat surface.

2.3. Business Case and Financial Analysis

2.3.1. Business Case

Now we are going to describe our Business case, based on the previous market survey and after analyzing our competitors we decided to set our initial price to be $500. For the next 5 years we expect to provide 2480 Unit with %17 expected loss so the total sold units expected to be 2059 Unit.

2.3.2. Financial Analysis

The financial projections for our project cash flow are highlighted in the table below. And here are some expected Capital Expenditure (Capex) which are one-time spending like costs of Computers Equipment and long-term software licenses. Also, there are other expenses called Operational Expenses (Opex), these types of expenses are recurring payments for Salaries paid for employees, Rent, Insurance, Repairs and maintenance, Advertising, Sales, and Direct material cost.

Table 2-1 - Financial Analysis

Measure	Year 1	Year 2	Year 3	Year 4	Year 5
Computer Components	$2200				
Long Term licenses	$5000				
Salaries	$22500	$22500	$22500	$37500	$45000
Rent	$3800	$3800	$3800	$3800	$3800
Repairs and	$5000	$5000	$5000	$5000	$5000

maintenance					
Advertising	$15000	$30000	$45000	$60000	$100000
Direct Material Cost	$19200	$38400	$76800	$153600	$307200
Total Cost	$72700	$99700	$153100	$259900	$420500
Total Sales	$30000	$60000	$136000	$288000	$576000
Cash Flow	-$42700	-$39700	-$17100	$28100	$155500

Based on the table above, our breakeven point would be in the 4th year which is expected in the hardware products due to the high cost for the raw material and loss which happens in the production.

Chapter 3: Literature Survey

In this chapter, we will discuss the main topics on which our project is built, these topics are: Inverse Kinematics, PID controllers and Robot Localization. We will then talk about related publications and our approach we implemented in order to build our project.

3.1. Background on Inverse Kinematics

Inverse Kinematics is the process of obtaining joint angles from known coordinates of end effector. The IK function takes a target position as the input, and calculates the pose required for the end effector to reach the target position [6].

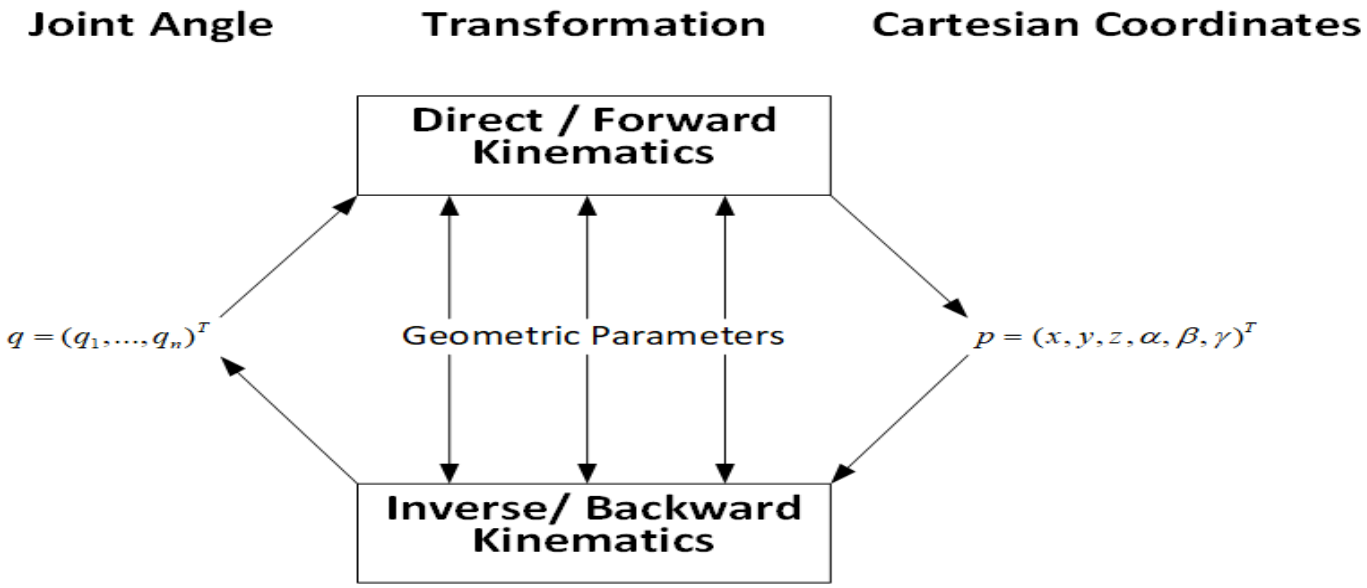


Figure 3-1 - Direct Kinematics VS Inverse Kinematics

IK can be used for a robot arm to reach a certain target [7]; it can also be used for bipedal robots to make its feet reach a desired position.

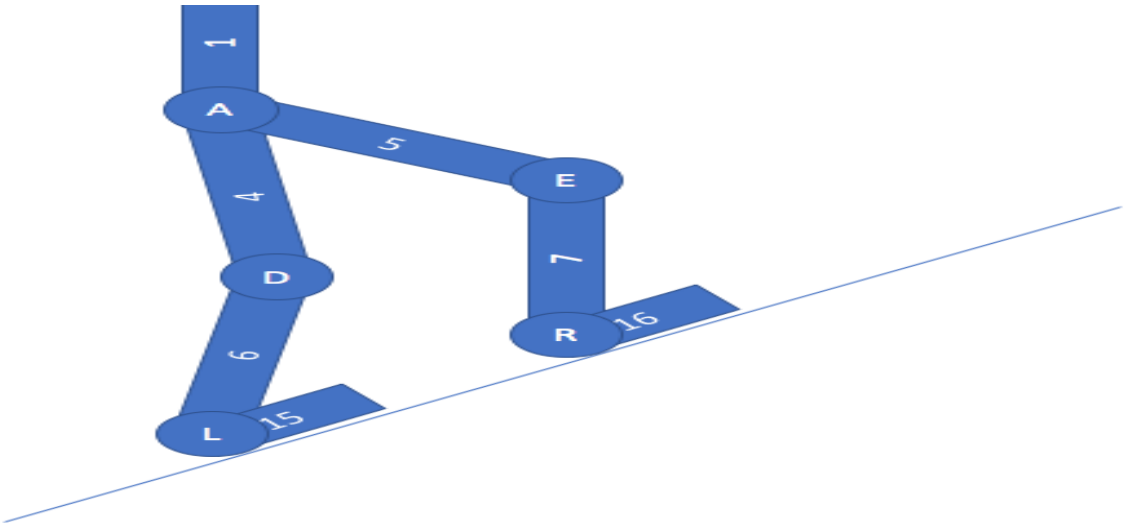


Figure 3-2 - Inverse Kinematics in Bipedal Robot Motion

3.2. Background on PID Controllers

A **PID controller** is an instrument used in industrial control applications to regulate temperature, flow, pressure, speed and other process variables. PID (proportional integral derivative) controllers use a control loop feedback mechanism to control process variables and are the most accurate and stable controller [8].

PID control is a way of driving a system towards a target position or level. PID control uses closed-loop control feedback to keep the actual output from a process as close to the target or setpoint output as possible.

3.2.1 PID Controller Basics

The purpose of a PID controller is to force feedback to match a setpoint. PID controllers are best used in systems which have a relatively small mass and those which react quickly to changes in the energy added to the process.

3.2.2 PID Controller Working Principle

The working principle behind a PID controller is that the proportional, integral and derivative terms must be individually adjusted or "tuned." Based on the difference between these values a correction factor is calculated and applied to the input. Here are the three steps:

1. **Proportional tuning** involves correcting a target proportional to the difference. Thus, the target value is never achieved because as the difference approaches zero, so too does the applied correction.
2. **Integral tuning** attempts to remedy this by effectively cumulating the error result from the "P" action to increase the correction factor. For example, if the oven remained below temperature, “I” would act to increase the head delivered. However, rather than stop heating when the target is reached, "I" attempts to drive the cumulative error to zero, resulting in an overshoot.
3. **Derivative tuning** attempts to minimize this overshoot by slowing the correction factor applied as the target is approached.

3.3. Background on Robot Localization

Robot localization is the process of determining where a mobile robot is located with respect to its environment. Localization is one of the most fundamental competencies required by an autonomous robot as the knowledge of the robot's own location is an essential precursor to making decisions about future actions [9].

A mobile robot equipped with sensors to monitor its own motion can compute an estimate of its location relative to where it started if a mathematical model of the motion is available. Robot localization techniques need to be able to deal with noisy observations and generate not only an estimate of the robot location but also a measure of the uncertainty of the location estimate.

3.4. Comparative Study of Previous Work

There are many recent publications in the field of bipedal robots. Every paper addresses some challenge that faces bipedal robots. Some of them just talk about motion pattern generation so a robot can only walk on a flat plane, other papers talk about walking on inclined surfaces and some other papers talk about robot localization by giving it a map of the environment. In our project we combine many techniques from many papers to address many problems at once. A recent study in 2019 made a balancing strategy for a bipedal robot so it can walk on inclined planes up to 9 degrees [10], our robot can walk on inclined planes up to 11.45 degrees.

3.5. Implemented Approach

After reviewing many papers addressing the problem of bipedal robots’ motion, we decided to implement Inverse Kinematics algorithm in order to generate our motion pattern by giving the algorithm the expected position in each step and it computes the joint angles. We also use PID controllers for balancing the robot, we read the tilt angles values from the gyroscope sensor and sends these values to the PID controllers along with the desired angles which is zero so the robot can keep itself balanced, the PID

controllers computes the changes in joint angles and the desired position that is sent to IK algorithm so after the PID controllers, we get new joint angles sent to the robot motors. At last, for the localization problem, we read the data from the gyroscope and accelerometer sensors and integrate their values in order to get the overall displacement of the robot with respect to its starting point in the environment. We also read the (x, y) cartesian position of the robot from the localization provided in the simulation tool to make sure our estimation is correct. We use this method to address the problem of giving the robot a path to follow, we keep track of the robot with respect to the given path so we make sure the robot follows its path and doesn't deviate from it.

Chapter 4: System Design and Architecture

In this chapter, we will discuss the design of our project in details and how everything was done. We will give a detailed description about every module and how these modules interact with each other in order to reach our target.

4.1. Overview and Assumptions

Forrest is a humanoid bipedal robot that can walk on flat and inclined planes, it also comes with a GUI that allows the user to draw a path for the robot to follow. After taking the path, it generates a motion pattern in order to know the cartesian position of its feet in each step and send these positions to the Inverse Kinematics module to calculate the required joint angles. There is also the controllers module which is based on PID controller in order to make sure the robot is stable and doesn't fall while walking.

In the following sections, we will show the block diagram of the project and discuss each module in details.

4.2. System Architecture

In this section we will show the block diagram of the project and how the modules interact with each other, we show the inputs and outputs of every module.

4.2.1. Block Diagram

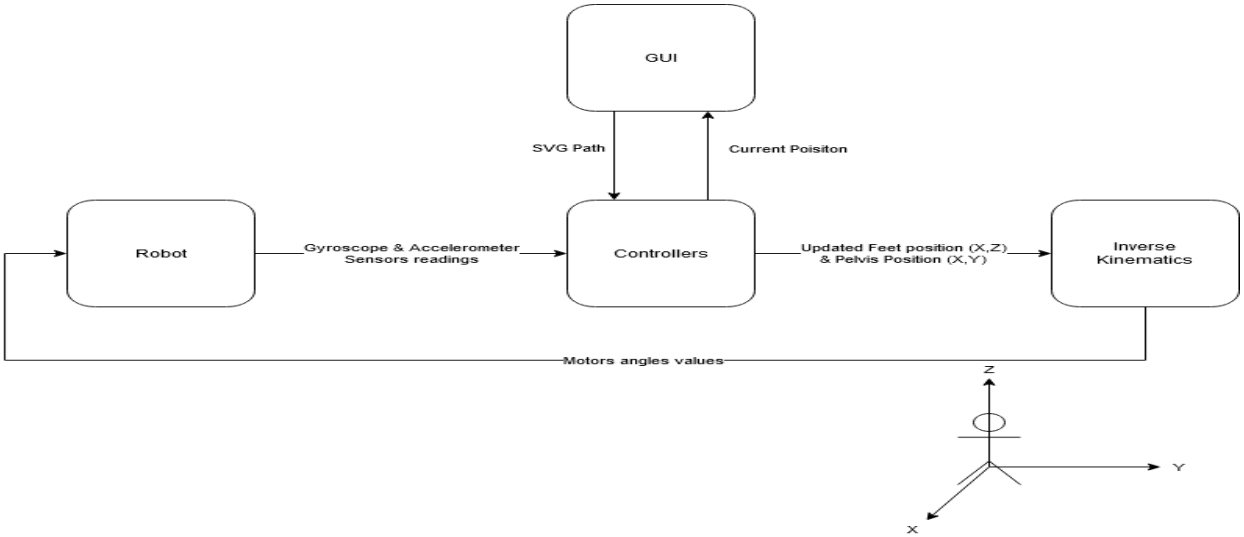


Figure 4-1 - Block Diagram

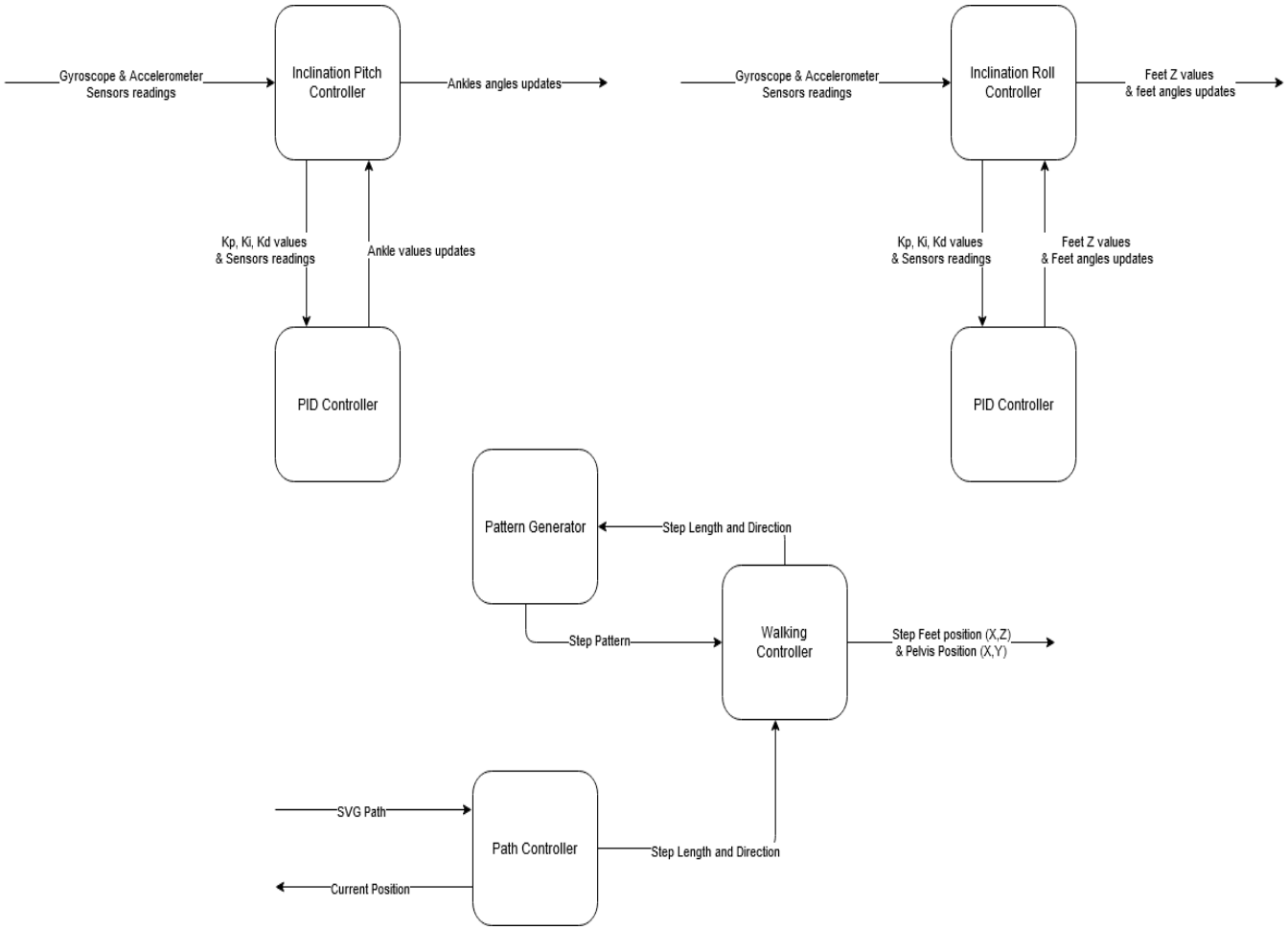


Figure 4-2 - Controllers Diagram

4.3. Robot

This module is the core of our project, it acts as the interface between our code and the robot hardware (simulation). The robot we use in simulation is called Robotis OP3, it is manufactured by a Korean robot manufacturer called ROBOTIS. It contains 20 motors (20 DOF), a 3-axes accelerometer and a 3-axes gyroscope [11].

Figure 4-3 - ROBOTIS OP3

4.3.1. Functional Description

The function of this module is to be the interface between the robot hardware and the code. It gets the sensors readings from the robot and it outputs the values of the joint angles on the robot's motors.

4.3.2. Modular Decomposition

This module consists of many main functions the directly deal with the robot body, it contains the components of the robot such as motors and sensors.

4.3.2.1 Robot Body

This function defines the distances between different motors in the motor in order to use these distances in generating the motion pattern. It also assigns unique ID for every motor in order to deal with these motors easily and defines the sensors used which are gyroscope and accelerometer.

4.3.2.2 Get Sensors

This function gets the sensors readings from the sensors in the body of the robot in the simulation and saves these readings in a file in order to process these readings in other modules.

4.3.2.3 Apply Angles

This function is considered as the output of this module, it takes the joint angles values after being calculated from the IK module and after being modified from the controllers module and applies these values to the motors in the robot body in order to make the robot take a step.

4.3.3. Design Constraints

The design of this module depends entirely on the model of the robot used. If another robot is used then this module will be changed in order to fit the new robot.

4.3.4. Other Description of Robot

Robot module also contains some functions used for plotting some graphs in order to make the function of the robot and its sensors more clear for the user.

4.4. Inverse Kinematics

IK module contains all the necessary equations and variables used in order to calculate the joint angles values.

4.4.1. Functional Description

The function of this modules is to calculate the joint angles values. It takes the cartesian position (x, y, z) as an input and outputs the joint angles values that should be send to the robot's motors.

4.4.2. Modular Decomposition

IK module consists of three functions, we will discuss each of them in details in the next subsections.

4.4.2.1 Inverse Kinematics XZ

This module takes the cartesian (x, z) position of the feet and the cartesian (x) position of the pelvis in the next step and calculates the joint angles values of the three motors in each leg which are LegUpper, LegLower and Ankle.

4.4.2.2 Inverse Kinematics Y

This module takes the cartesian (y) position of the feet in the next step and calculates the joint angles values of the motors in the pelvis of the robot body.

4.4.2.3 Get Inverse Kinematics Angles

This module takes the cartesian (x, z) position of the feet, the cartesian (x, y) position of the pelvis in the next step and the distances between the robot's motors. It calls the other two functions described in the previous two subsections and calculates the final values of the joint angles to be sent to the robot's motors.

4.5. Walking Pattern Generator

WPG is one of the most important modules in our project. It is responsible for generating the pattern which the robot will follow in order to take a step.

4.5.1. Functional Description

This module takes some constant parameters which are delay time between steps, maximum foot height, maximum step forward displacement, maximum pelvis side displacement, maximum pelvis forward displacement and the length between the pelvis and the foot. It outputs the step pattern which the robot will follow during its movement.

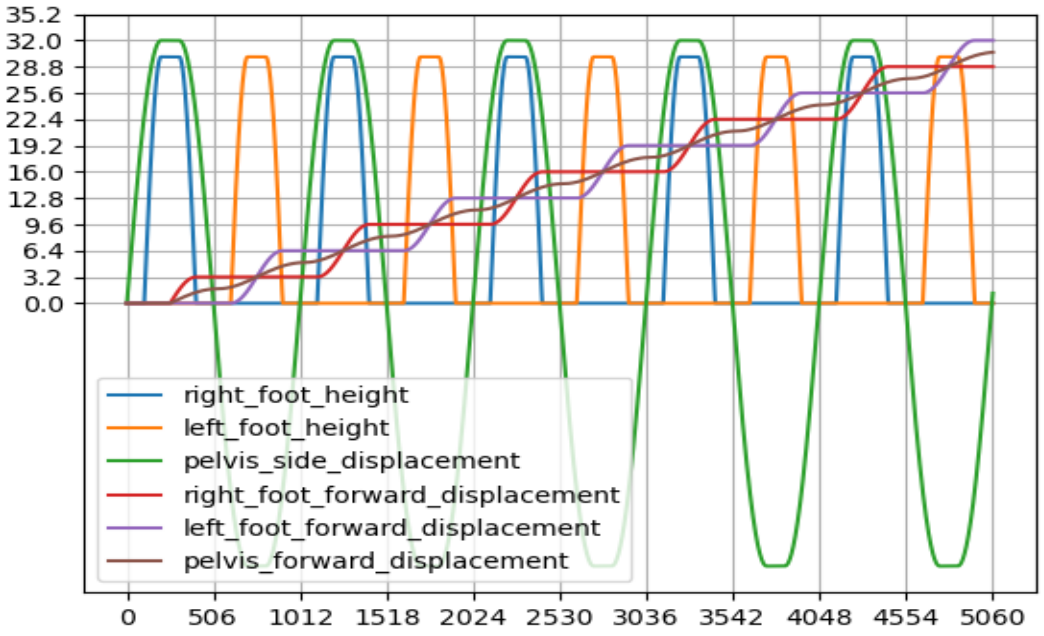


Figure 4-4 - Walking Pattern

4.5.2. Modular Decomposition

WPG module contains many functions which are used together in order to calculate the cartesian (x, y, z) position of the robot during its motion in order to send this position to the IK module to calculate the joint angles values. In the following subsections we will discuss these functions in details.

4.5.2.1 Calculate Stride Time

This function takes the constant g and the leg height in order to calculate the time needed for each stride (2 successive steps).

4.5.2.2 Foot Height

This function takes the foot which will move next (right or left) and the time for each level during step (on ground, moving upwards, hanged in its maximum elevation, moving downwards and reaching the ground again) and calculates the values of z cartesian position of the foot during this step.

4.5.2.3 Foot Forward Displacement

This function takes the foot which will move next (right or left) and the time for each level during step (standing on ground, moving forward and standing on ground again) and calculates the x cartesian position of the foot during this step.

4.5.2.4 Pelvis Side Displacement

This function takes the time for each level during step (moving sideward, staying in its maximum position and returning back to its position) and calculates the y cartesian position of the pelvis during this step.

4.5.2.5 Pelvis Forward Displacement

This function takes the time for each level during step (moving forward with right leg and moving forward with left leg) and calculates the x cartesian position of the pelvis during this step.

4.5.2.6 Get Smooth Value

This function takes the period of step and the cartesian (x, y, z) position of this step and generates a sin function for the leg to follow in order to smooth the movement of the leg.

4.5.2.7 Foot rotation

This function takes the time of rotation of a leg to complete the rotation and which leg will move (right of left) and return the cartesian position of the leg during rotation.

4.5.2.8 Get Step

This function takes the leg which will move next (right or left), the distance of the last step and if this step is the final step. It generates the motion pattern for the leg in order to make a step forward. It calculates all the needed cartesian values (feet height, feet displacement, pelvis side and forward displacement and legs rotation) and the time per each movement in order to complete a step.

4.5.3. Design Constraints

This module contains many constants as mentioned before in section 4.5.1 which depends on the robot itself, these constants will change if the robot used is changed.

4.6. Path Controller

This module is responsible for making the robot follows the path given by the user.

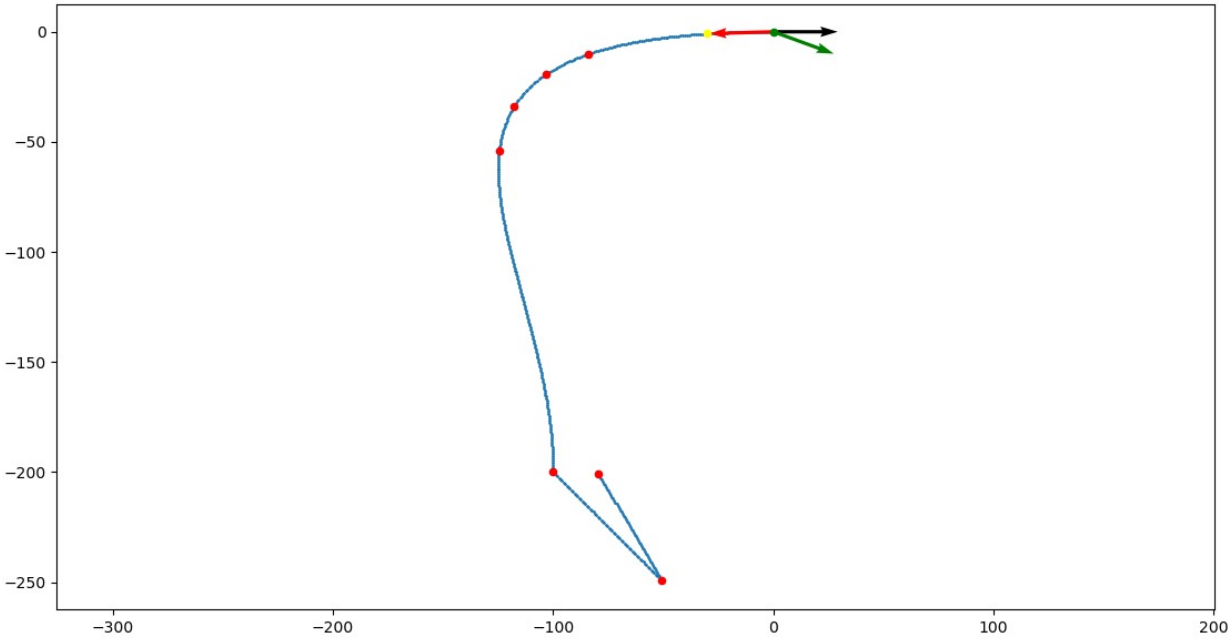


Figure 4-5 - User Defined Path

4.6.1. Functional Description

This module takes the input path from the GUI as SVG and returns to the GUI the current position of the robot so the user can keep track of the movement of the robot. It also sends the step direction and length to the walking controller in order to generate a motion pattern for this step.

4.6.2. Modular Decomposition

This module contains the functions used in order to keep track of the robot position and divides the path into number of steps in order to generate the motion pattern for each step.

4.6.2.1 Get Vectors

This function is used to divide the given SVG path to vectors, it also returns the orientation of vectors in order to get the angle between two successive vectors to make the robot rotate around its axis with this angle.

4.6.2.2 Find Checkpoints

This function checks the angles between two successive vectors and compare it with a certain threshold. If the angle exceeds this threshold it considers the intersection point between these vectors as a checkpoint. Checkpoints are used to make sure the robot is following the given path.

4.6.2.3 Reduce Checkpoints

After defining the checkpoints on the path, this function checks if the distance between two successive checkpoints is below a certain threshold. If so, it removes one of the checkpoints. This makes the robot does less calculations in order to walk faster and finish the given path faster.

4.6.2.4 Update Plot

This function keeps track of the position of the robot and sends the plot to the GUI module so the user can see the movement of the robot on its path.

4.6.2.5 Get Step

This function is the most important function in the path module. It keeps track of the current position of the robot and check where the robot will go in the next step. It gets the distance between the current position and the next position and checks if there are any checkpoints in the path between these points. It also gets the angle between the vector of the current position and the next position to check if the robot needs to rotate around itself.

4.7. Walking Controller

This module is the interface between the path controller and walking pattern generator. It communicates with both of them to get the cartesian points that the robot will reach in order to make a step.

4.7.1. Functional Description

Walking Controller module gets the position and the orientation of the next point that the robot will go to. It sends this position to the walking pattern generator in order to get the pattern of the step. After getting the pattern of the step, it sends this pattern to the IK module to compute the joint angles to be send to the robot’s motors.

4.7.2. Modular Decomposition

This module contains two main functions to keep track of the current position and to get the pattern of the next step.

4.7.2.1 Accumulate

This function keeps track of the current position of the robot by adding the distance between the current position and the next position to the current position value.

4.7.2.2 Get Step

This function is the main function of this module. It combines together the Walking Pattern Generator module and the Path controller module. First, it gets the direction of the next step to see if the robot will need to rotate around itself and it gets the step length. It sends the step length to the WPG module to get the motion pattern of this step and then sends the rotation angle as well as the motion pattern to the IK module to calculate the needed value for the joint angles.

4.8. PID Controller

PID Controller is the base module that is used in Balancing Controllers to make the robot walks on inclined planes.

4.8.1. Functional Description

This module takes the values of the PID Controller which are Kp, Ki and Kd and the maximum step of the motors and calculates the needed values of the motors in order to balance the robot on inclined planes.

4.8.2. Modular Decomposition

This module contains two main functions for calculating the values needs in order to make the error (the tilting angles in our case) equal to zero and make sure that the robot can reach the values that the PID controller will output.

4.8.2.1 Apply Guard

This function acts as a threshold. It takes the values that the PID controllers calculates and compare them to the maximum reach of the robot's motors in order to send to the motors reasonable values they can reach.

4.8.2.2 Update

This function contains the algorithm of the PID controller. It takes some error (which is the tilting angles in our case) and the values of the angles we want the robot to reach (which is zero). It also takes the PID controller constants which are Kp, Ki and Kd. It calculates the values that will be sent to the IK module to convert these values to the joint angles which will be sent to the motors and call the Apply Guard function in order to make sure that the robot can reach these values.

4.9. Inclination Pitch Controller

This Controller is responsible for making the robot balanced on inclined planes where the value of the pitch angle (rotation around y-axis) exceeds zero.

4.9.1. Functional Description

This module takes the values of tilting in the pitch angle and calculates the required values of joint angles that need to be sent to the motors.

4.9.2. Modular Decomposition

This module contains two functions, one to calculate the needed change in motors values and the other to check the stability of the robot.

4.9.1.1 Check Stability

This function gets the sensors reading and compare the tilting of the pitch angle with a certain threshold to make sure the robot is stable.

4.9.1.2 Get Step

This function is the core of the Inclined Pitch Controller module. It creates an instance of the PID controller and sends to it the PID constants and the reading of the sensors. It gets the values of the ankle angles that needs to be sent to the robot to adapt to the inclination in plane.

4.10. Inclination Roll Controller

This Controller is responsible for making the robot balanced on inclined planes where the value of the roll angle (rotation around x-axis) exceeds zero.

4.10.1. Functional Description

This module takes the values of tilting in the roll angle and calculates the required values of joint angles that need to be sent to the motors.

4.10.2. Modular Decomposition

This module contains two functions, one to calculate the needed change in motors values and the other to check the stability of the robot.

4.10.1.1 Check Stability

This function gets the sensors reading and compare the tilting of the roll angle with a certain threshold to make sure the robot is stable.

4.10.1.2 Get Step

This function is the core of the Inclined Roll Controller module. It creates an instance of the PID controller and sends to it the PID constants and the reading of the sensors. It gets the values of the feet angles that needs to be sent to the robot as well as the height (cartesian z value) of the feet to adapt to the inclination in plane.

4.11. Graphical User Interface

This module eases the usage of our project to the user. It allows the user to draw a path to the robot to follow. It also allows the user to watch the robot while walking and keeps track of the robot position in the path.

4.11.1. Functional Description

This module takes the path from the user and shows him the robot while walking and the current position of the robot on the path. It also allows the user to start, stop and reset simulation.

4.11.2. Modular Decomposition

This module consists of 4 main parts shown in the figure 4.6.

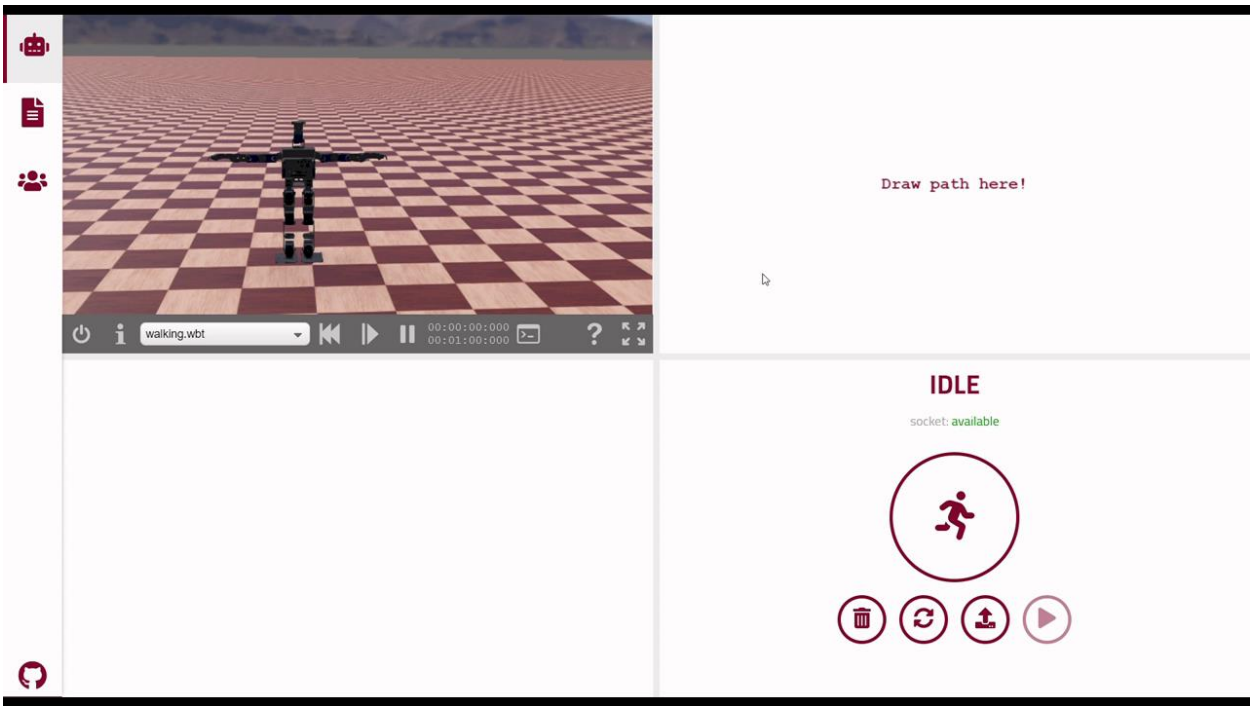


Figure 4-6 - GUI Home Screen

The upper left part shows the screen from the simulation tool Webots, it allows the user to watch the robot while walking in the loaded environment.
In the upper right part, the user can draw a path to be sent to the robot to follow as we see in figure 4.7.

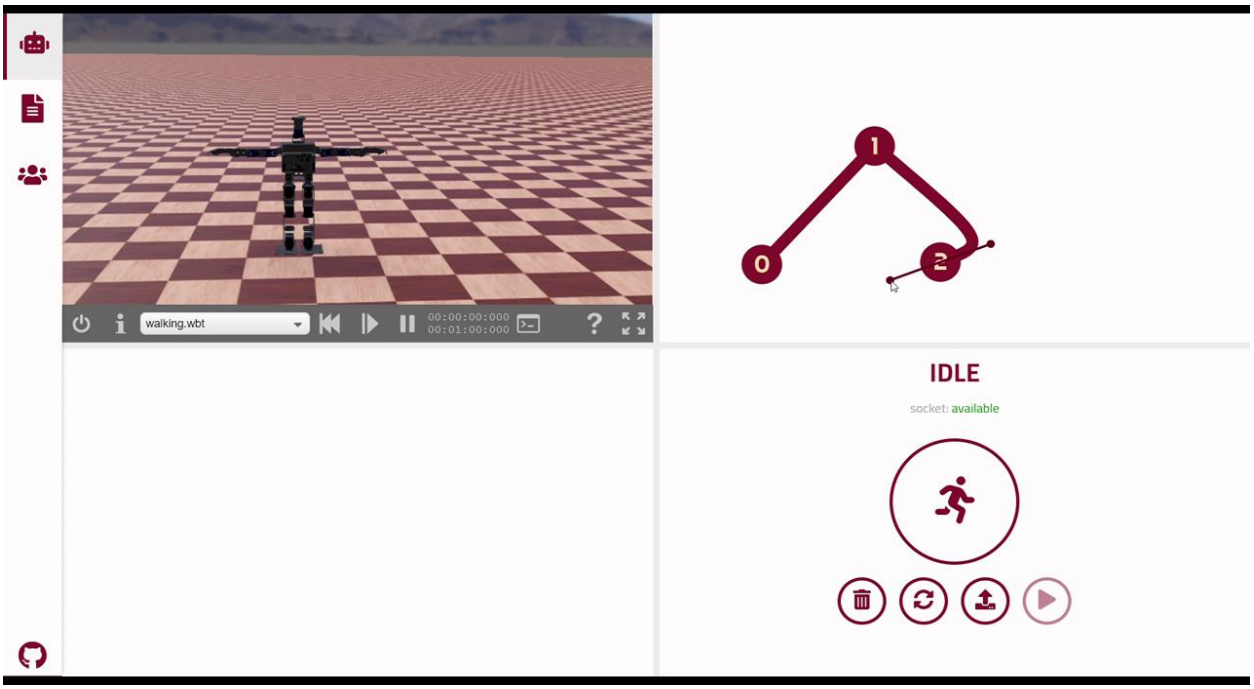


Figure 4-7 - User Drawing Path

In the lower left part, the application shows the user the current position of the robot on the path as we see in figure 4.8.

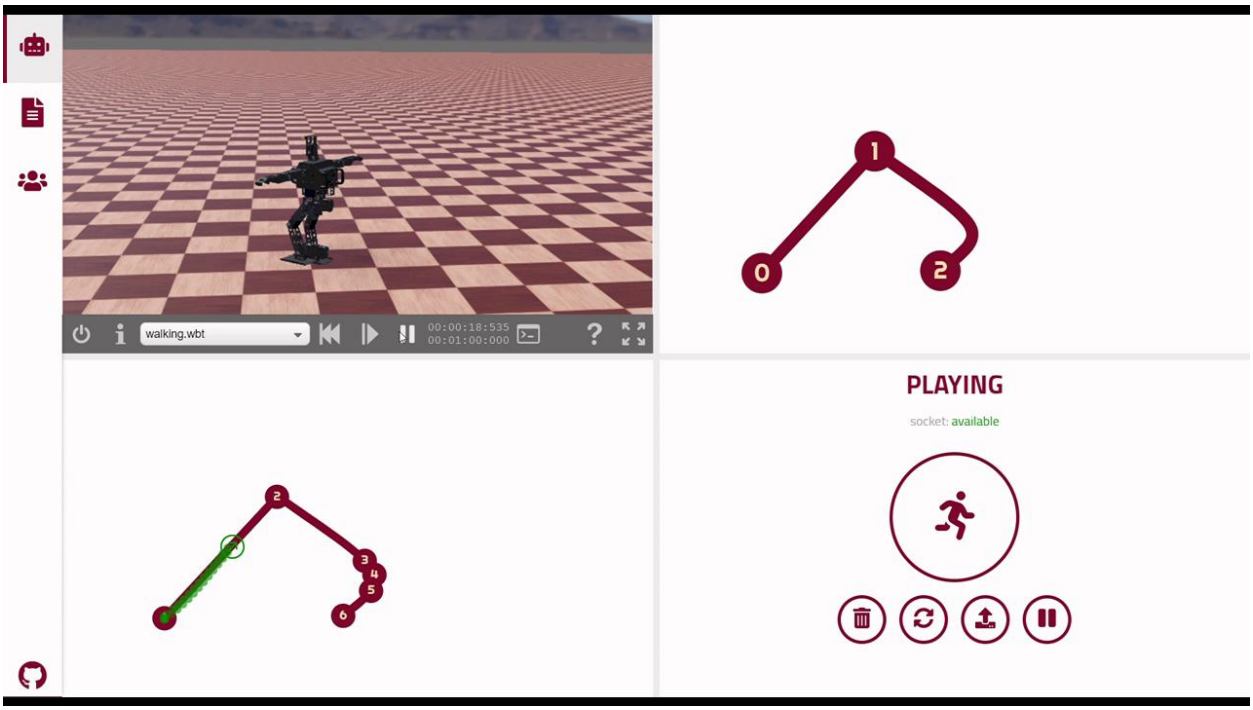


Figure 4-8 - Robot Walking on Path

The green arrow shows the current position of the robot and the green line indicates the distance cut by the robot on the path.
The lower right part is where the user orders the program to start, stop, reset, delete drawn path and uploads a saved SVG path for the robot to follow.

4.12. OP3 Motion Player

OP3 Motion Player is the main module of the project. It combines all other modules together in order to make the project work.

4.12.1. Functional Description

This module is responsible for integrating the whole project together. It calls the other modules in order and connects them all together to make the project work.

4.12.2. Modular Decomposition

This module launches two threads, one for the walking of the robot and the other for talking input from the user.

4.12.2.1 WebSocket Handler

This function is responsible for talking inputs from user, it checks if the user sent an action by sending ping messages to the server and waits until the user sends an action.

When the user sends an action, it checks the type of action sent (Path – Play - Pause) in order to change the current state of the application so the control function acts according to this state.

4.12.2.2 Control

This function combines all the modules needed for walking together. It gets the current state of the application from the WebSocket Handler and acts accordingly. If the current state is Play. It starts the motion of the robot by calling the controllers module. It gets the joint angles values needed to be sent to the robot, check the stability of the robot and then send these values to the robot in order to move its motors to walk.

Chapter 5: System Testing and Verification

Testing is the activity to check whether the actual results match the expected results and to ensure that the software system is Defect free. It involves the execution of system components to evaluate one or more properties of interest. Testing also helps to identify errors, gaps, or missing requirements in contrary to the actual requirements. Testing our project was very important in order to make sure our robot can walk on any plane without falling and to make sure it follows the path that is given to it. In the following sections we will discuss in details how we tested our project in order to make sure it meets our target.

5.1. Testing Setup

In order to test our project, we created many scenarios for the robot. We tested our robot's motion on flat and inclined planes as well as giving the robot a specific path to follow and make sure it follows it. We used Webots simulation tool in order to create many scenarios to test our robot's motion.

5.2. Testing Plan and Strategy

Our strategy in testing was creating different environments with different planes in Webots in order to test all the functions of the robot. We faced some challenges in this part as Webots documentation wasn't good enough and it didn't explain how to create different environment nor to add many planes with many inclination angles. We had to go through Webots and discover how to do so. We tested motion on flat planes, inclined planes and both flat and inclined planes. At last after adding user specific path module we tested it by giving it many paths with different rotation angles. In the following subsections we will discuss how we created these environments and how did our robot respond to these environments.

5.2.1. Module Testing

5.2.1.1 Walking Pattern and Inverse Kinematics Testing

These two modules are responsible for the robot motion on flat planes. We created a simulation environment with a flat plane and we ran the simulation in order to see how the robot walks on this plane.

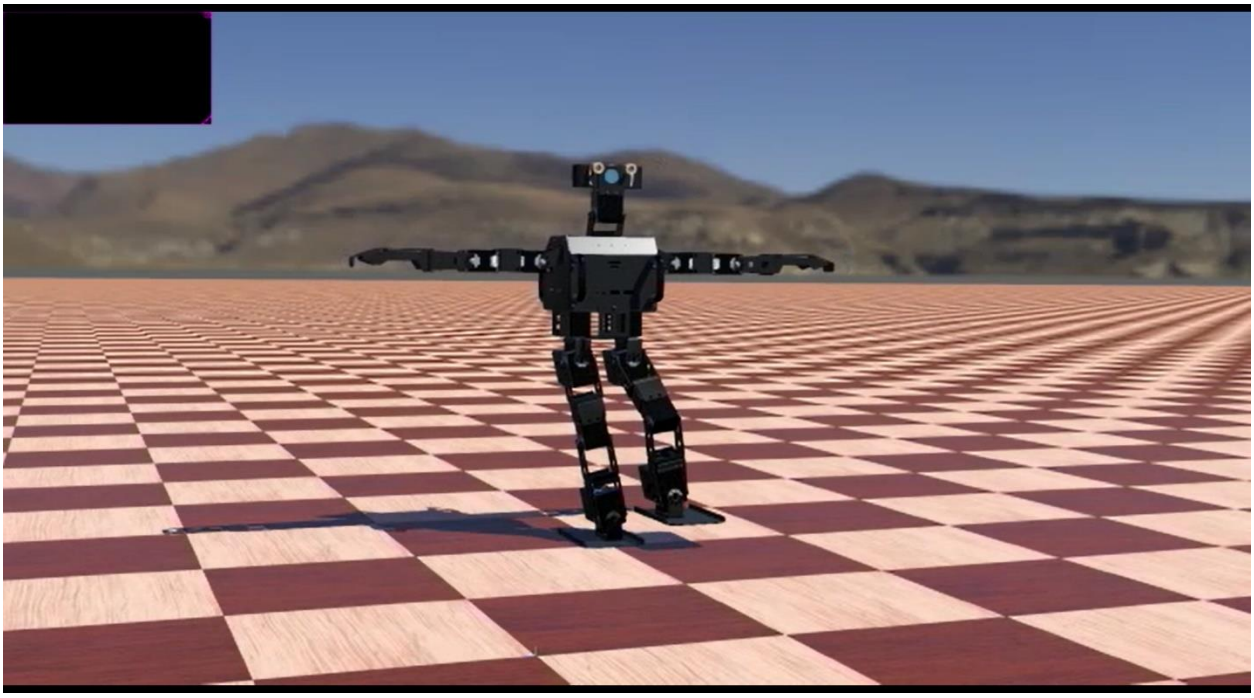


Figure 5-1 - Walking step 1

As we see here, we loaded a basic environment with only a flat plane. The robot starts it motion by tilting towards the fixed leg on the floor and moves the second one forward.

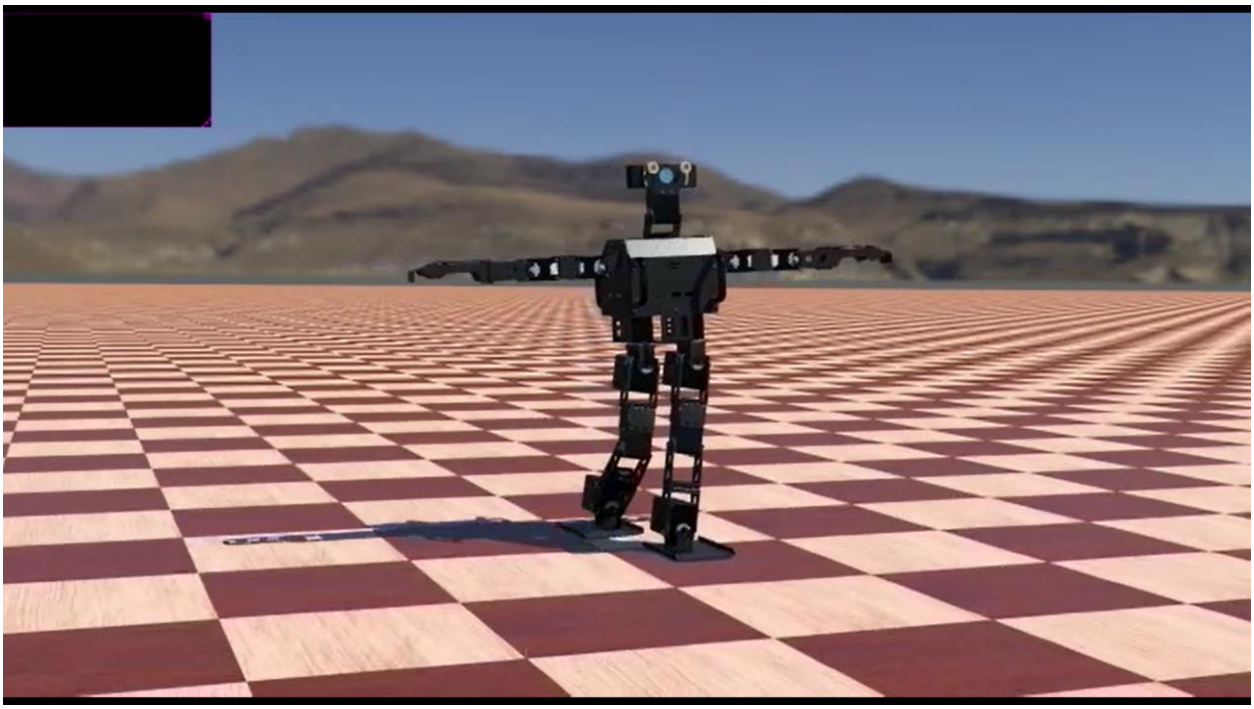


Figure 5-2 - Walking step 2

After that it tilts towards the leg which it moved forward and starts moving the other leg.

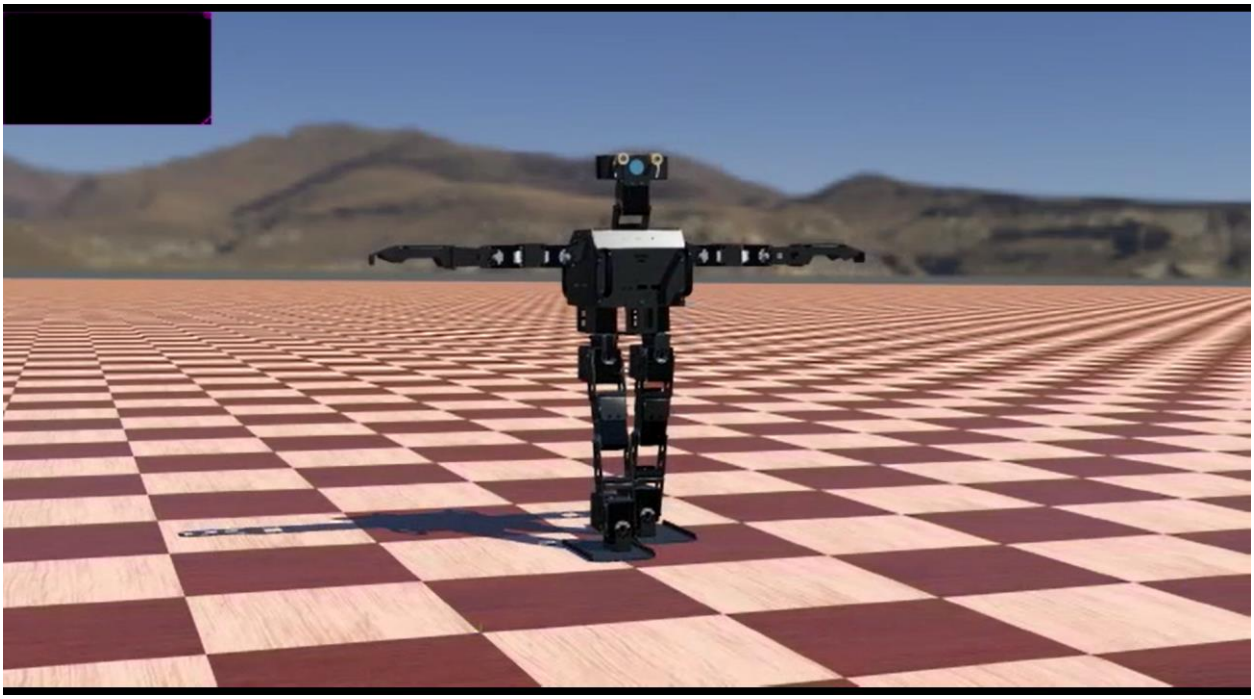


Figure 5-3 - Walking step 3

At last it lands this leg, tilts towards it and move the other leg. It keeps doing so until the number of steps specified ends or the user stop the simulation.

5.2.1.2 Balancing Controllers Testing

The controllers module is responsible for keeping the robot balanced on inclined planes so it doesn't fall. It keeps track of the tilting angles and tries to make them zero so the robot keeps itself balanced. For testing this module, we changed the plane in the environment to be inclined instead of flat and let the robot walk.

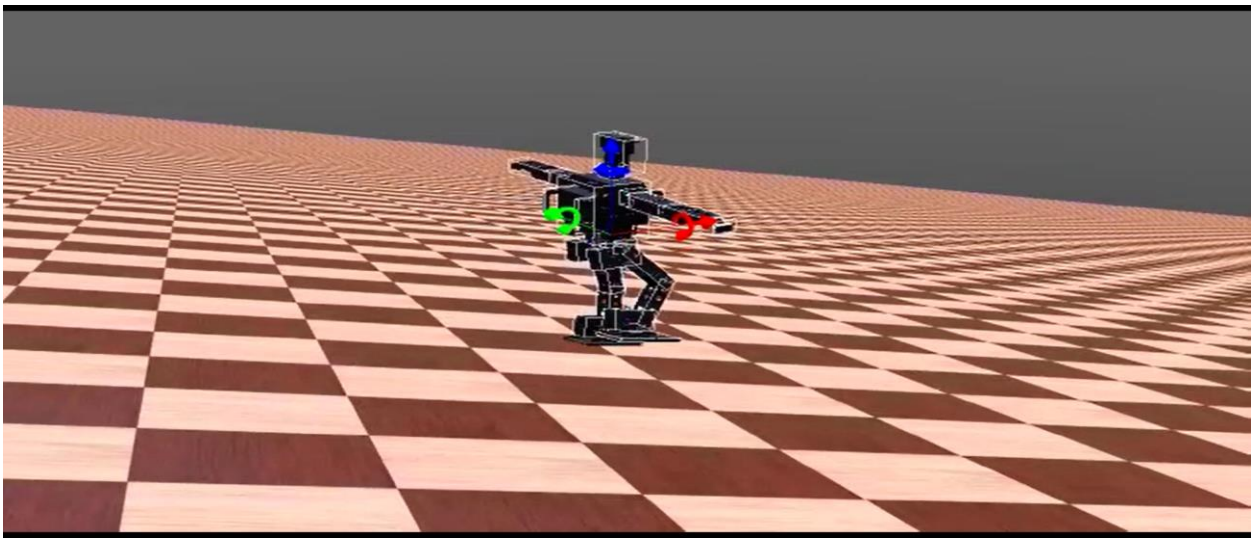


Figure 5-4 - Walking on inclined plane

Here we set the plane inclination angle to 0.2 rad (approximately 11.45 degrees), the motors that control the tilting angle of the feet (ankle and foot motors) make the feet tilting angles equal to the inclination angle of the plane in order to make the body of the robot stable and balanced.

5.2.1.3 Walking with Balancing Testing

Here we integrate the previous test cases into one full scenario testing where the robot walks on all kinds of planes in one test case. It walks on a flat plane then on and inclined plane upwards then flat plane again then inclined plane downwards then flat plane again. The robot always keeps track of its tilting angles in order to change the tilting angles of the feet to keep itself balanced.

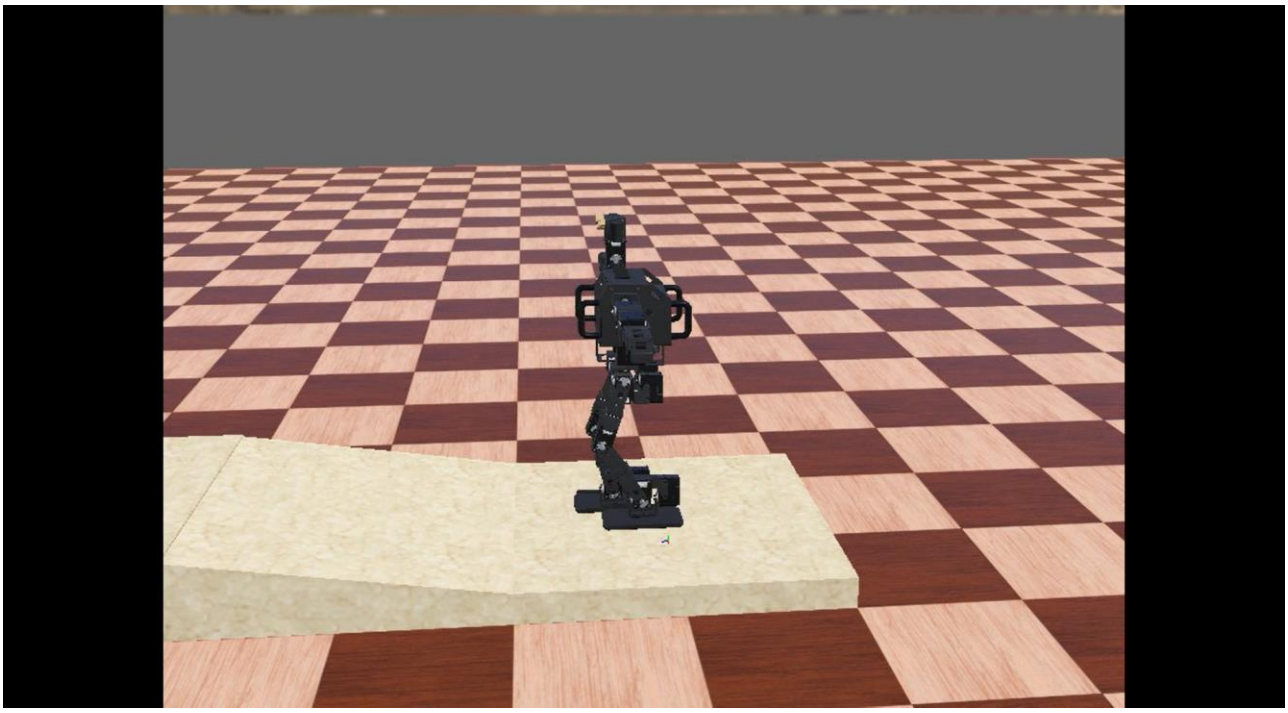


Figure 5-5 - Full Scenario (flat plane 1)

Here we added a marble block object in the environment, we added five blocks (3 flat, 1 inclined upwards with angle 0.15 rad and 1 inclined downwards with angle 0.15 rad) and then we placed the robot on the first flat plane and let it start walking.
When the simulation starts, it finds itself on a flat plane so it keeps the feet inclination angles equals to zero. It continues walking until it finds itself tilting backwards due to the inclined plane as we see in figure 5.6 so it changes the feet tilting angle to be equal to the inclination angle of the plane so the full body tilting angle be equal to zero again as we see in figure 5.7.

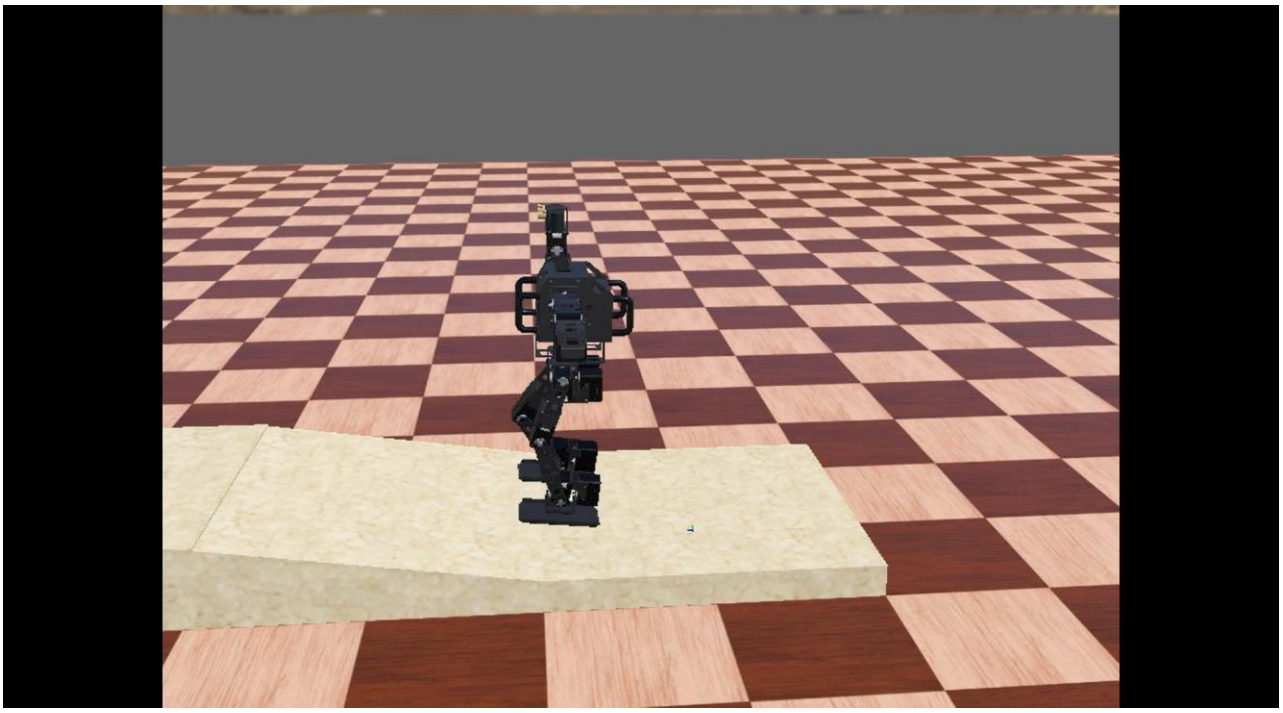


Figure 5-6 - Full Scenario (edge of inclined plane upwards)

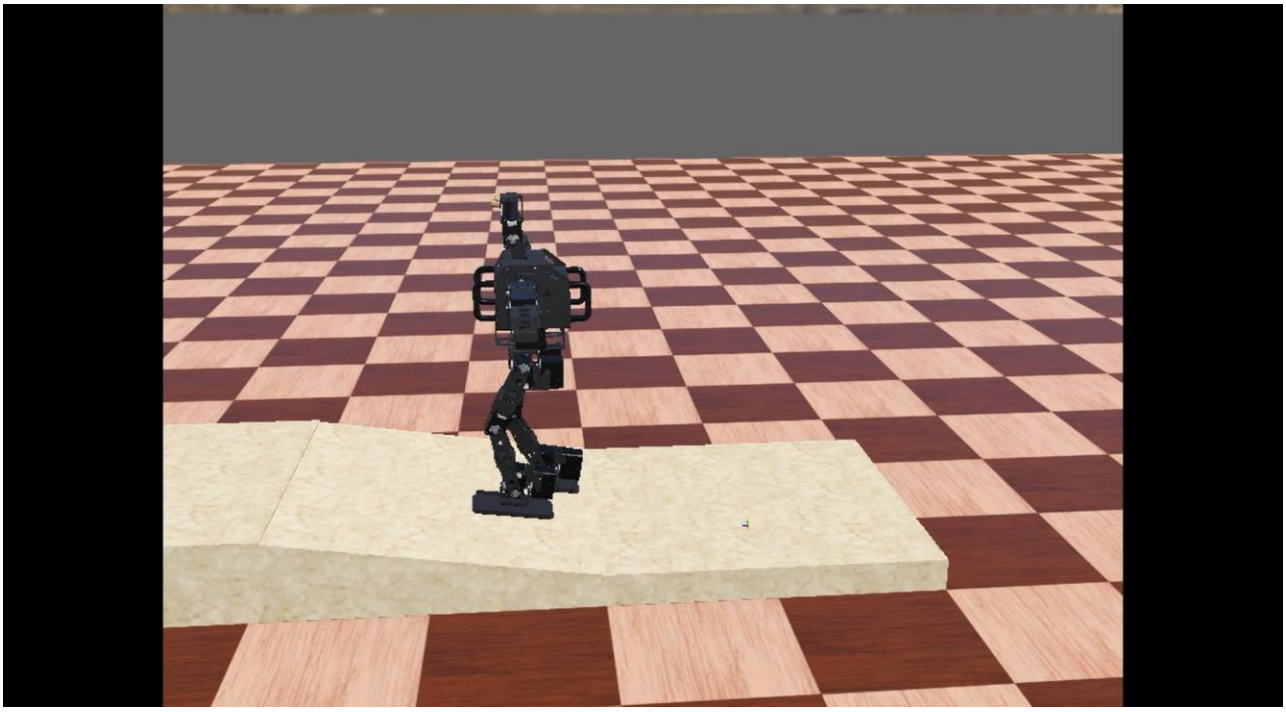


Figure 5-7 - Full Scenario (middle of inclined plane upwards)

It keeps walking upwards until it finds itself on a flat plane again as we see in figure 5.8. As soon as it finds itself on a flat plane, it changes the tilting angles of the feet to be equal to zero again.

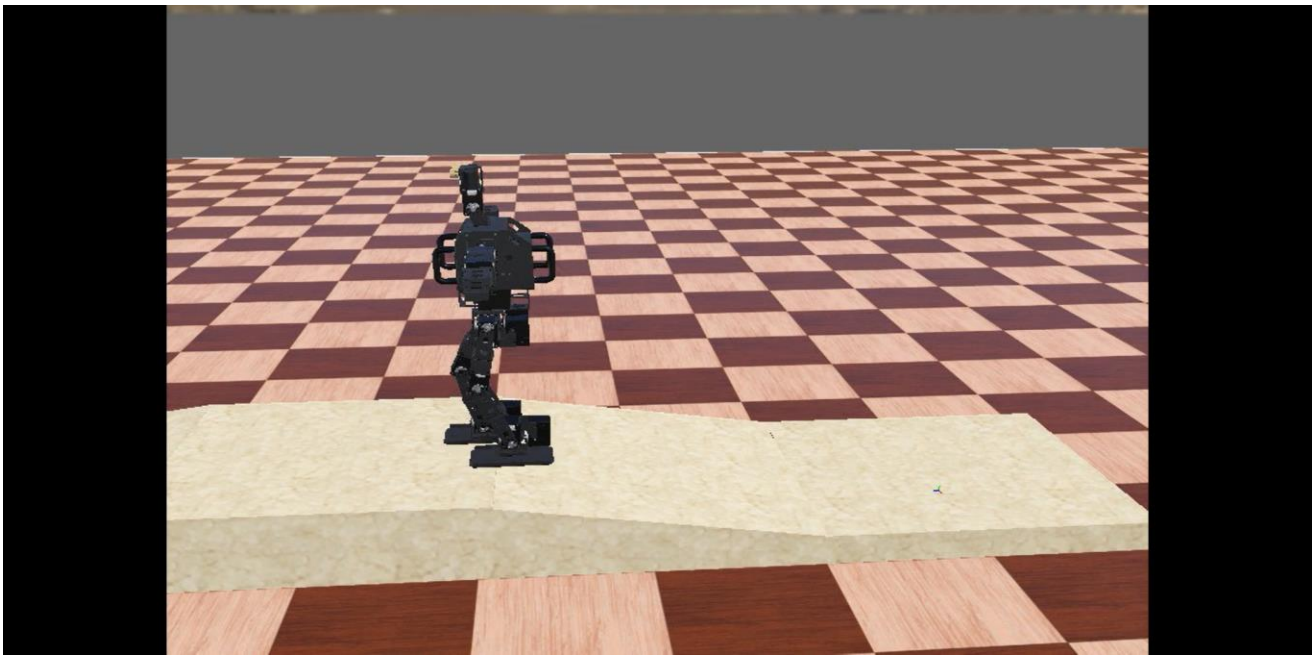


Figure 5-8 - Full Scenario (flat plane 2)

So, it keeps moving until the edge of the flat plane and as soon as it finds itself tilting forwards at the edge of the downwards inclined plane as we notice in figure 5.9, it sets the tilting angles of the feet to be equal to the inclination angle again so the body tilting angle return back to zero as we see in figure 5.10.

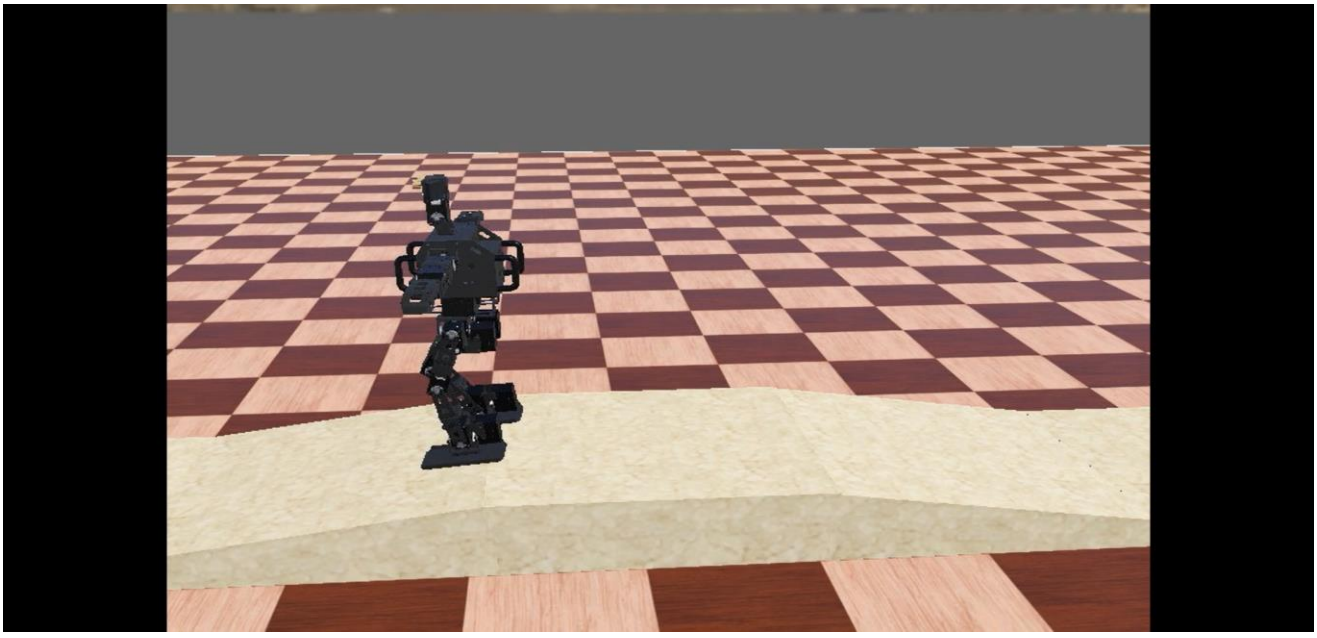


Figure 5-9 - Full Scenario (edge of inclined plane downwards)

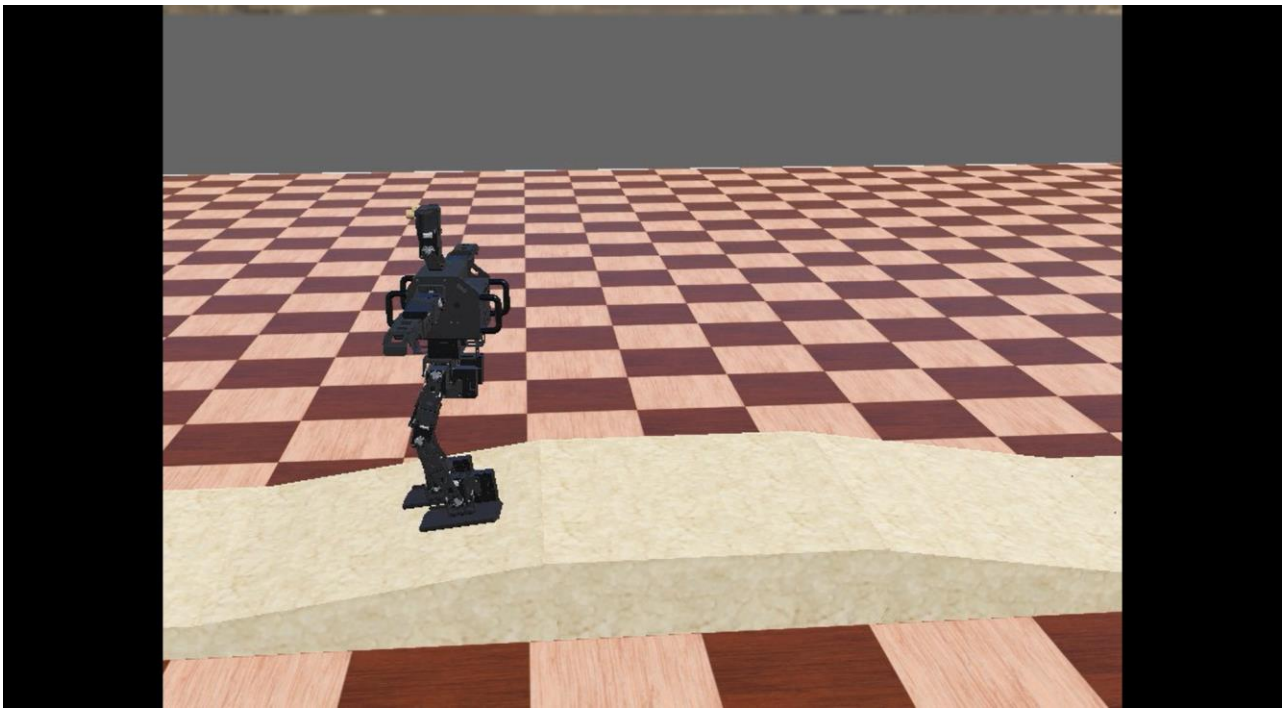


Figure 5-10 - Full Scenario (middle of inclined plane downwards)

It then keeps moving with this tilting in the feet until it reaches the last flat plane (figure 5.11). It makes the tilting angles of the feet to be equal to zero again and keeps walking until the simulation ends.

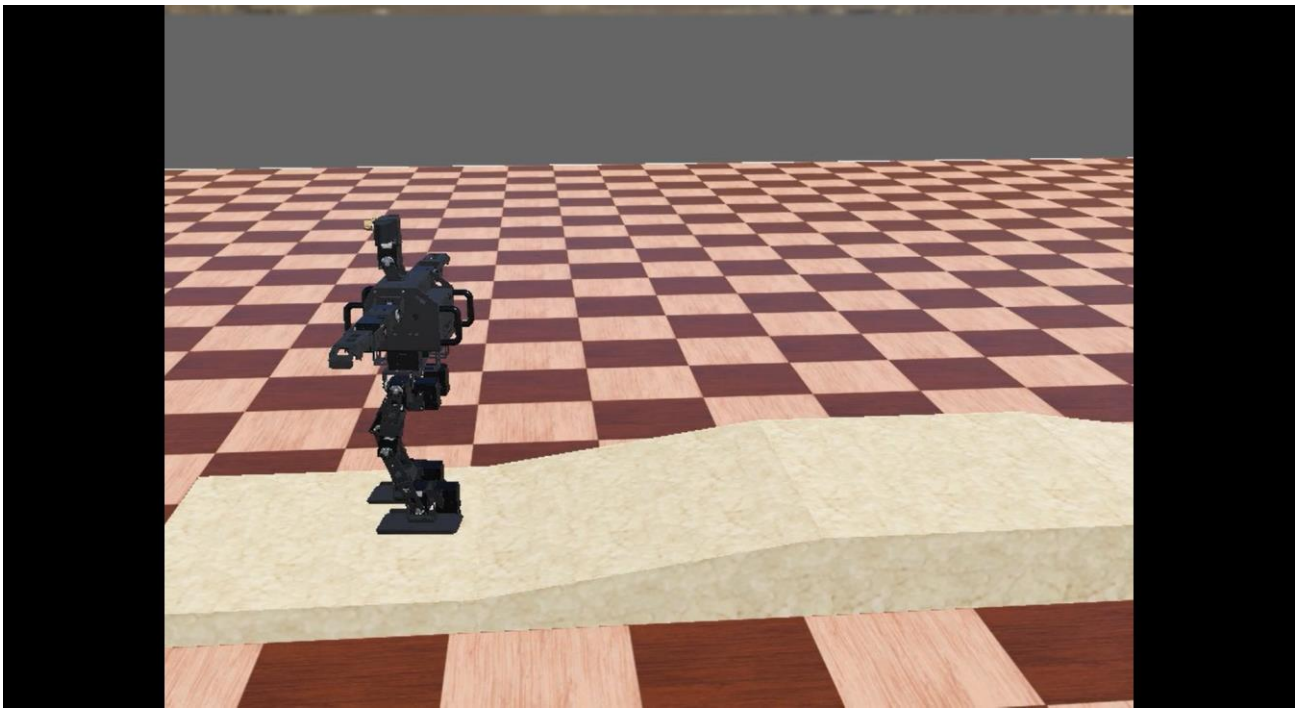


Figure 5-11 - Full Scenario (flat plane 3)

5.2.1.4 User Specified Path Testing

For testing this module, we created an SVG path to send it to the robot. The path that was sent in in figure 5.12.

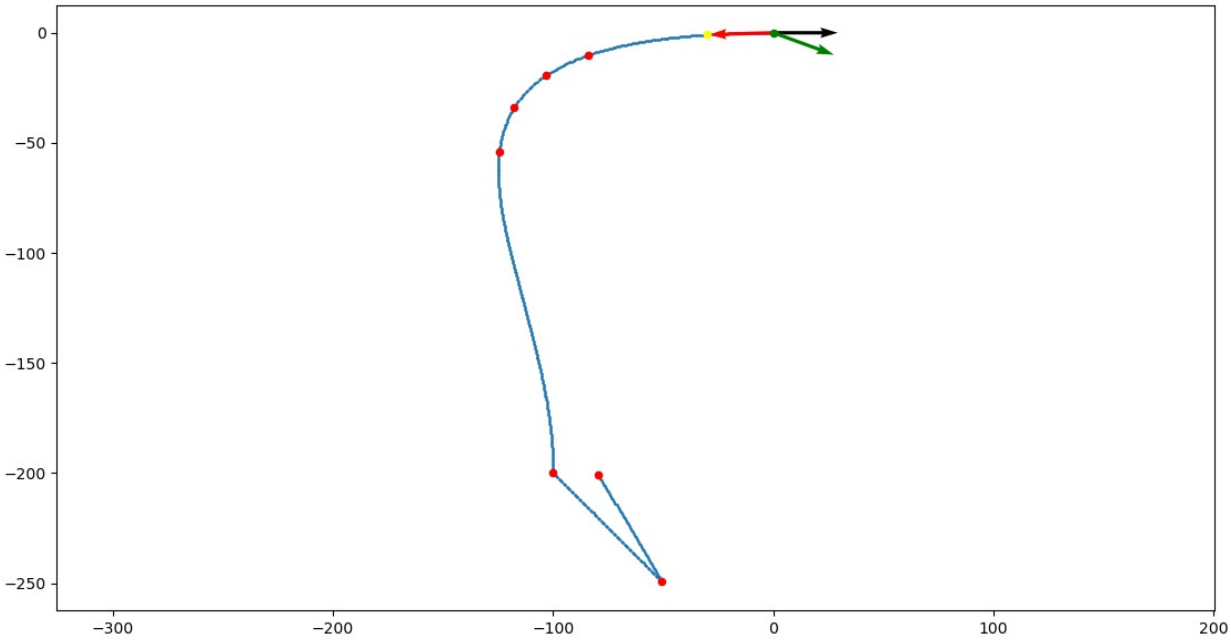


Figure 5-12 – User Defined Path

The Black arrow is the orientation of the robot, the red arrow is the starting angle in the path and the green arrow is the next step that the robot will do in order to get to the starting position of the path.

After giving this path to the robot, we started the simulation.

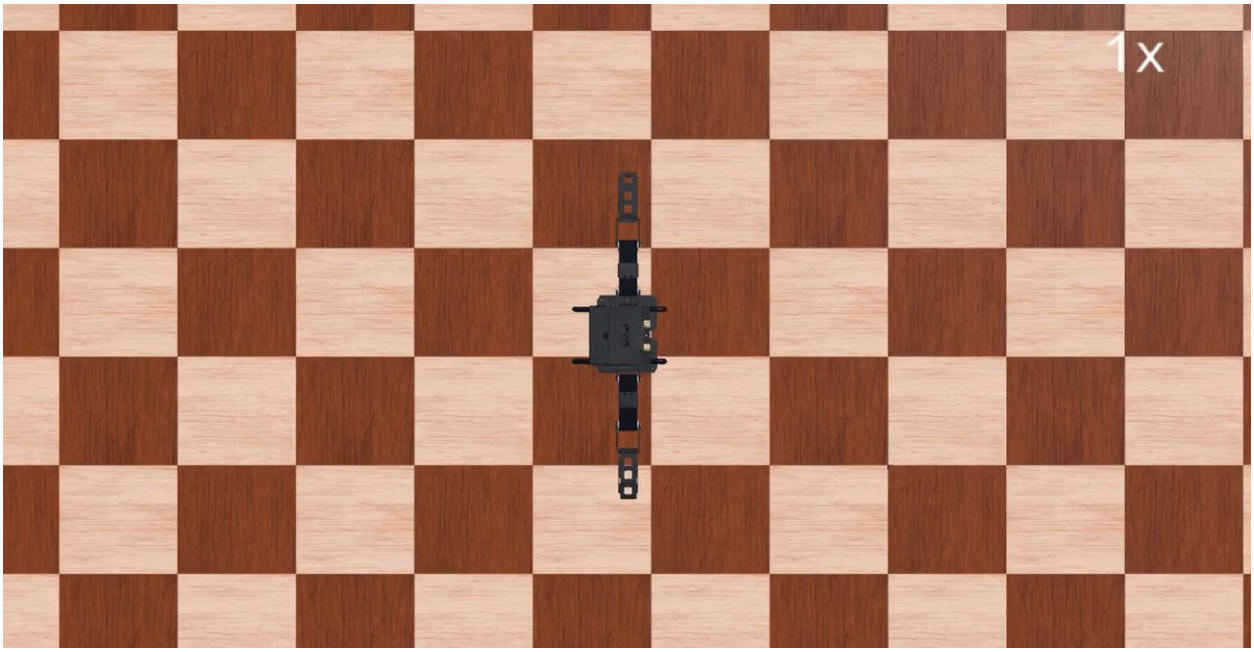


Figure 5-13 - Robot Walking on Path (Start)

As we see in figure 5.13, when the robot started motion, it was looking to the right as the black arrow indicates in figure 5.12.

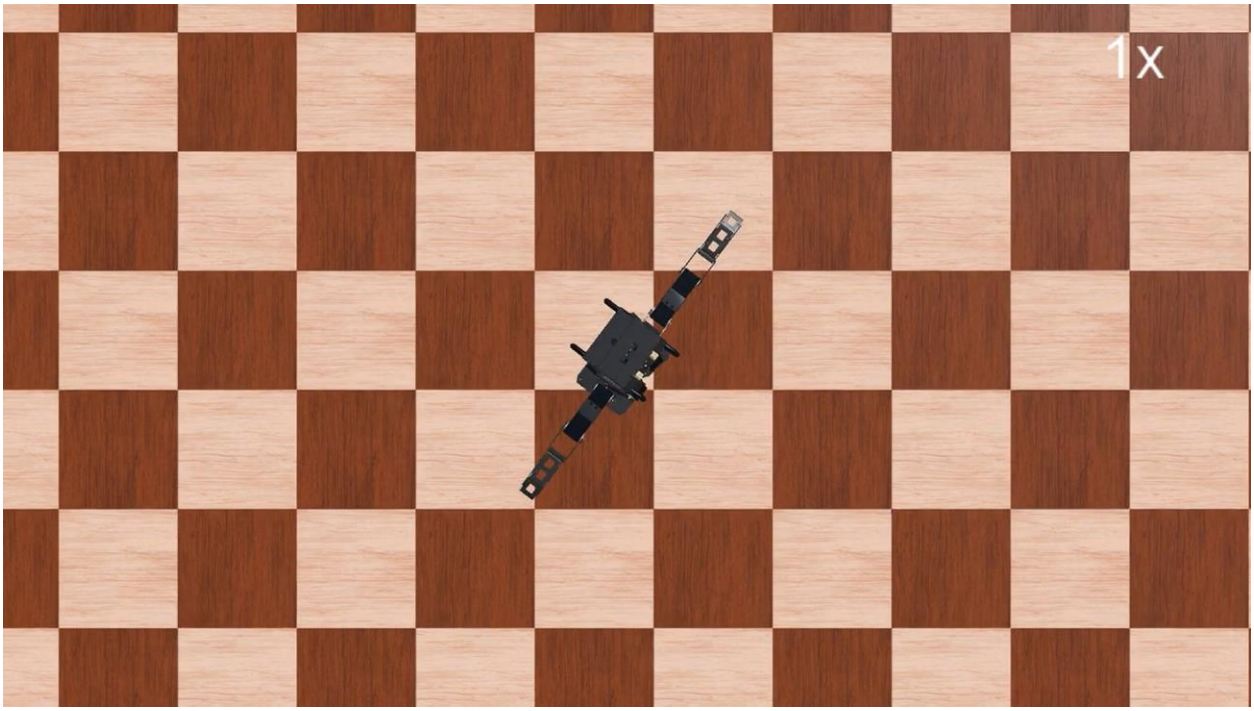


Figure 5-14 - Robot Walking on Path (Rotating)

As we see here, the robot started rotating around itself, the green arrow in figure 5.12 shows the position of next step, so here it shows the position of the robot after taking the first step in rotating.

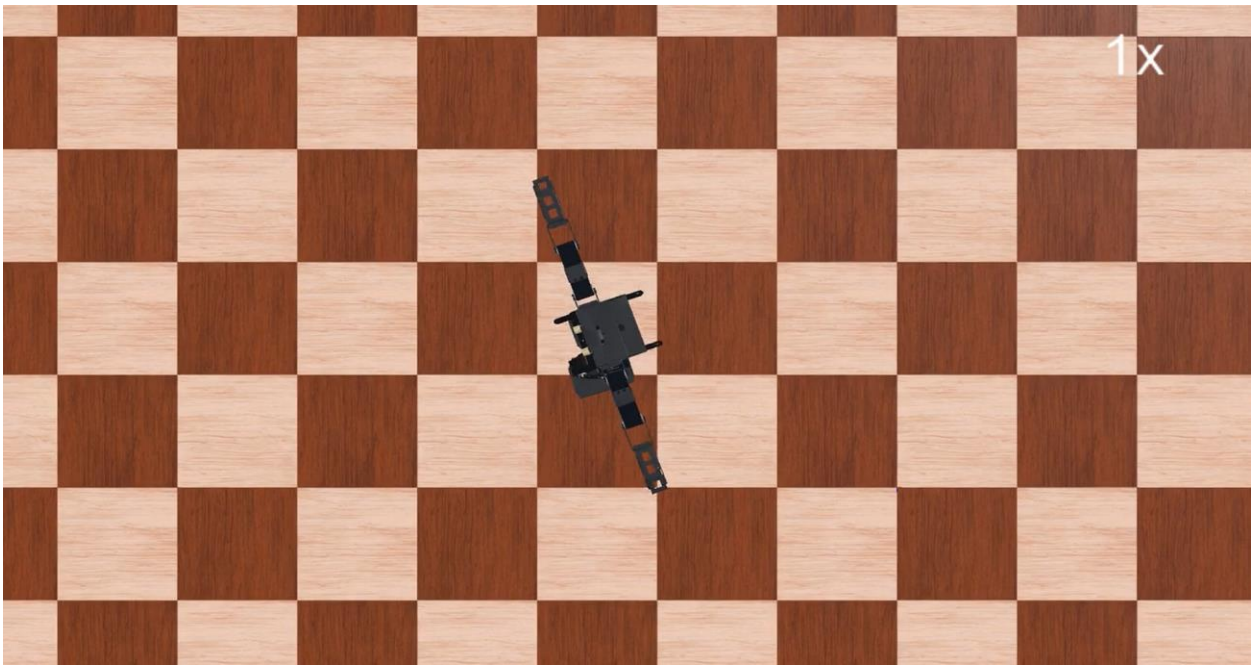


Figure 5-15 - Robot Walking on Path (Started Walking)

Here the robot reached the orientation of the starting point of the path so it will start moving forward.

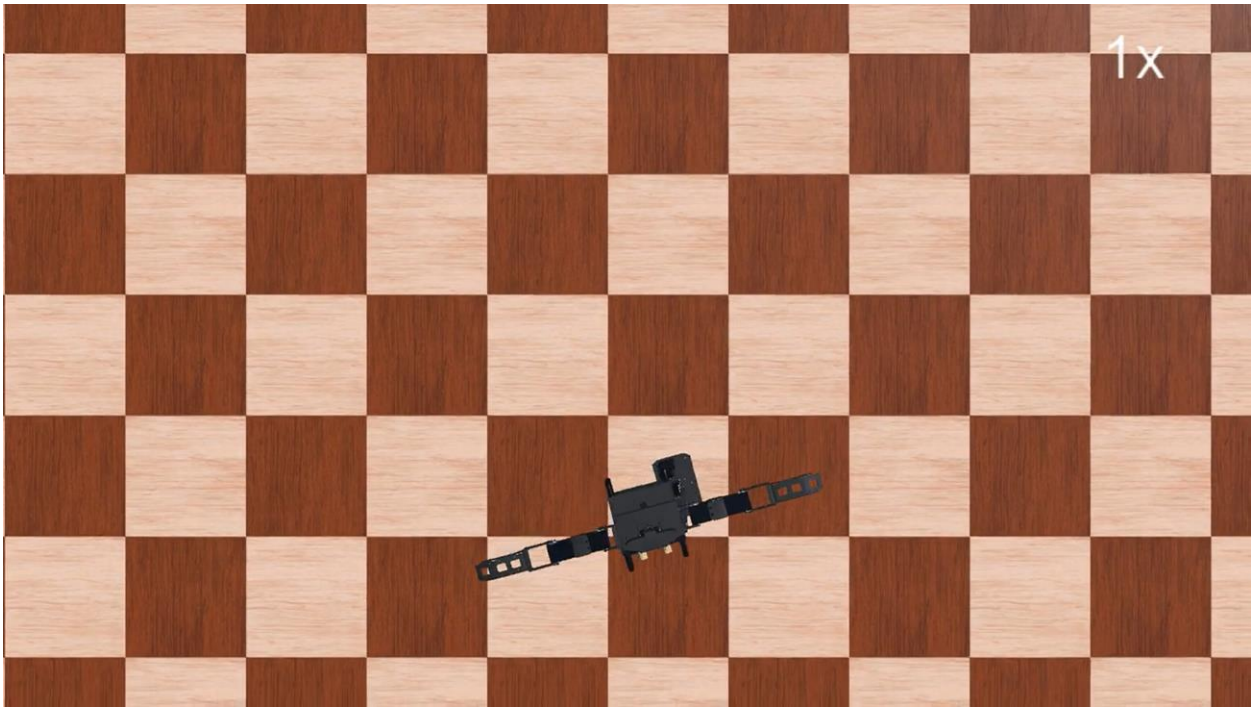


Figure 5-16 - Robot Walking on Path (Walking downwards)

As we notice in figure 5.12, the path contains a line downwards from -50 on y-axis to -200 on y-axis, figure 5.16 shows the robot while walking on this line.

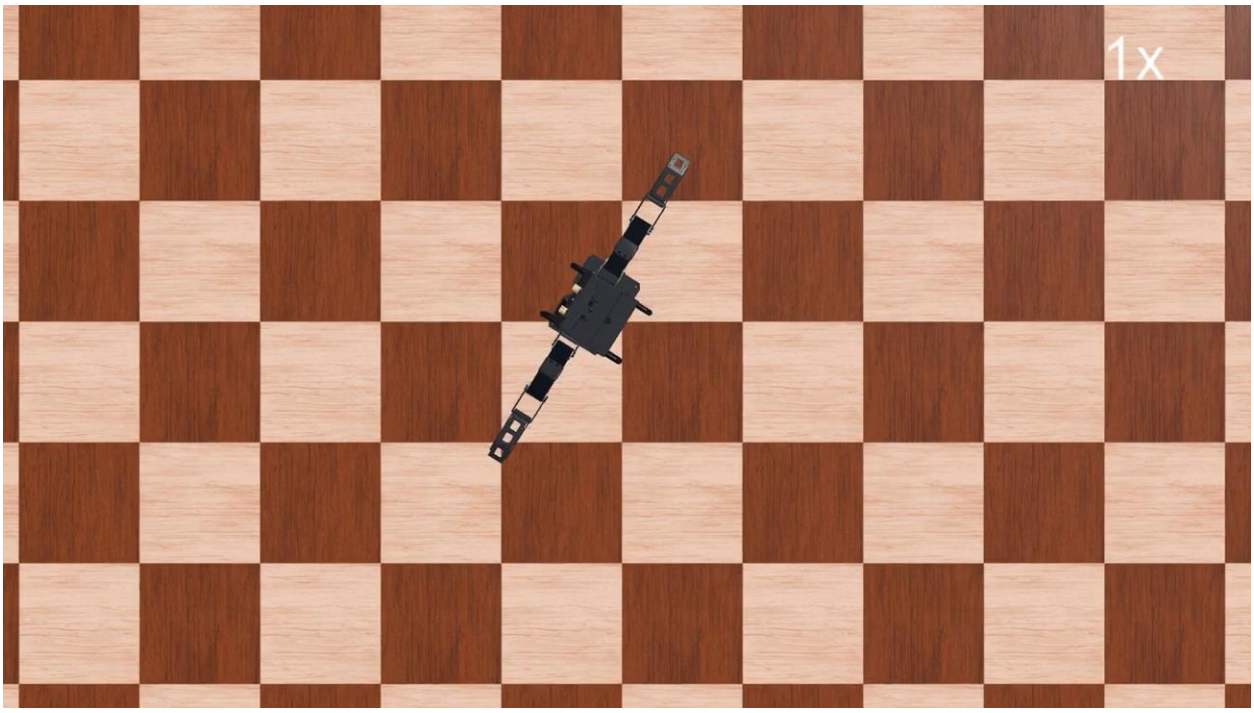


Figure 5-17 - Robot Walking on Path (End)

In figure 5.17, the robot reached the final point on the path. When it noticed one of the feet reached the final point. It gets the other feet beside of it instead of talking a normal step. This only happens when the robot reaches the end of the path.

5.2.2. Integration Testing

After implementing the latest module in the project which is the GUI module, it's normal to test the project as a whole to make sure that everything works fine. We started this by opening the application and sees the connect screen shown in figure 5.18.

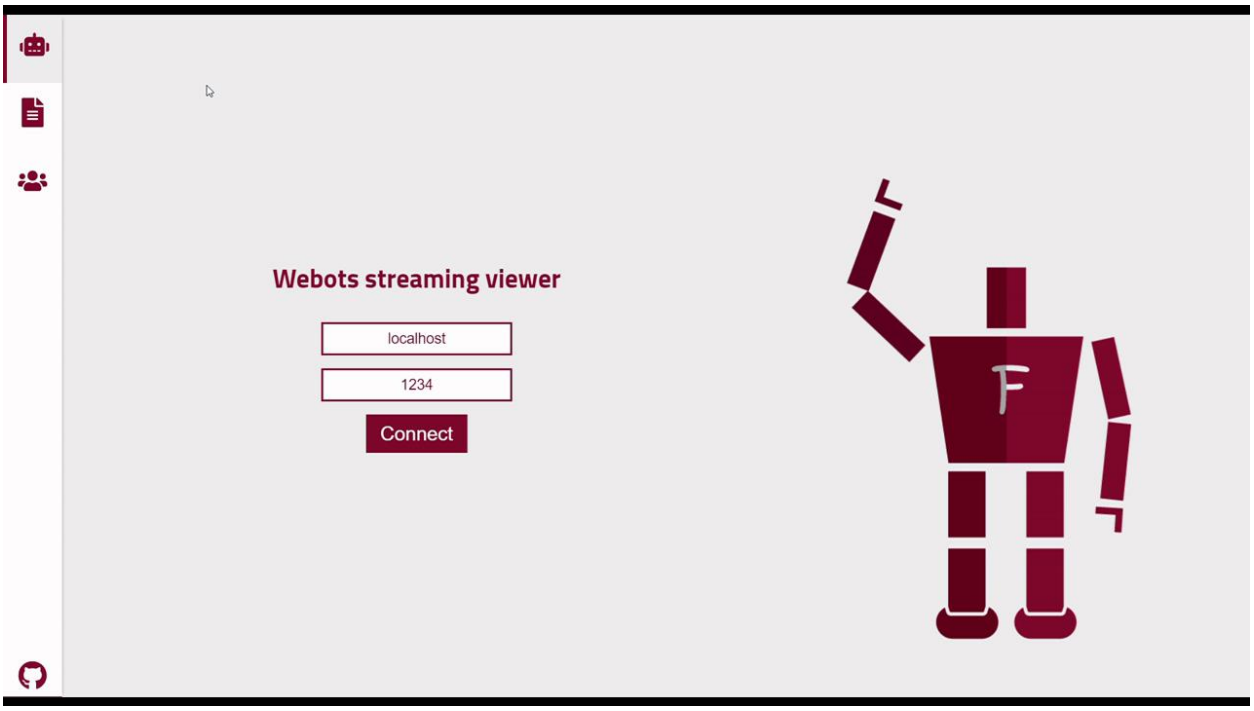


Figure 5-18 - GUI Connect page

After clicking on connect button, we expect to see the home page which is shown in figure 5.19.

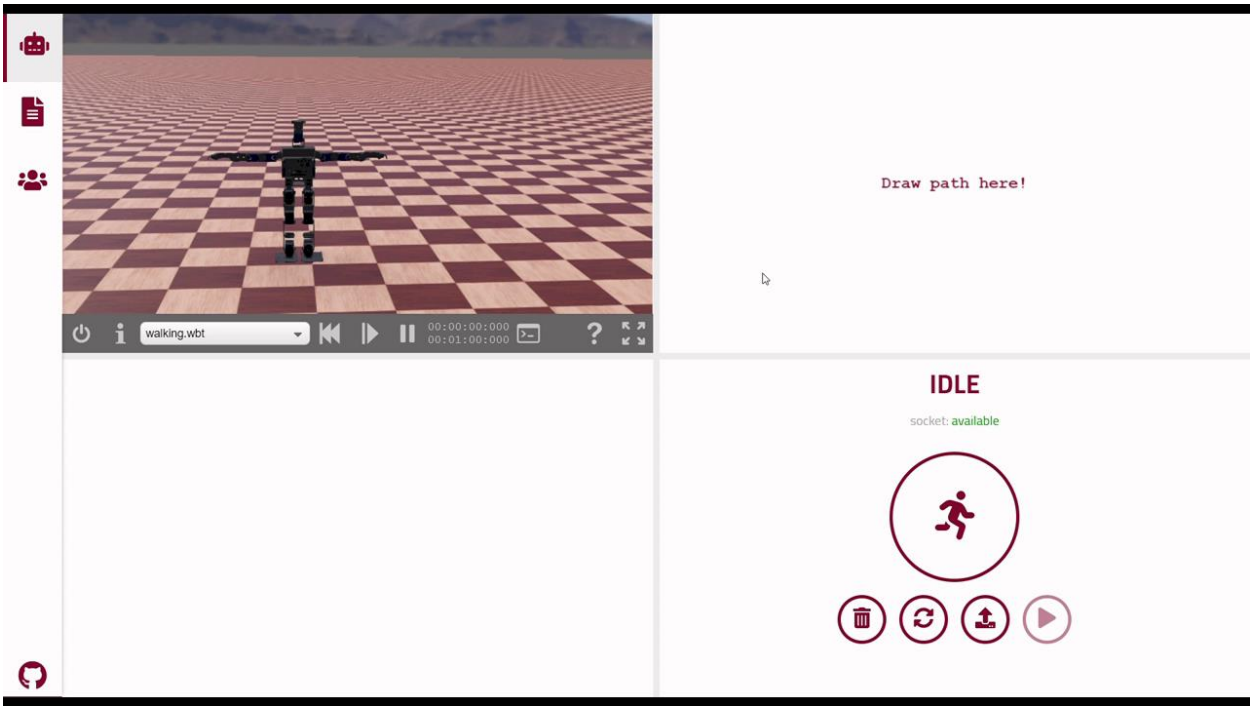


Figure 5-19 - GUI Home page (Draw)

We start drawing a path in the upper right part in the page, after drawing it we clicked on the run button. The lower left part now shows the path that the robot will follow in addition to the checkpoints which the robot makes sure it reaches them as shown in figure 5.20.

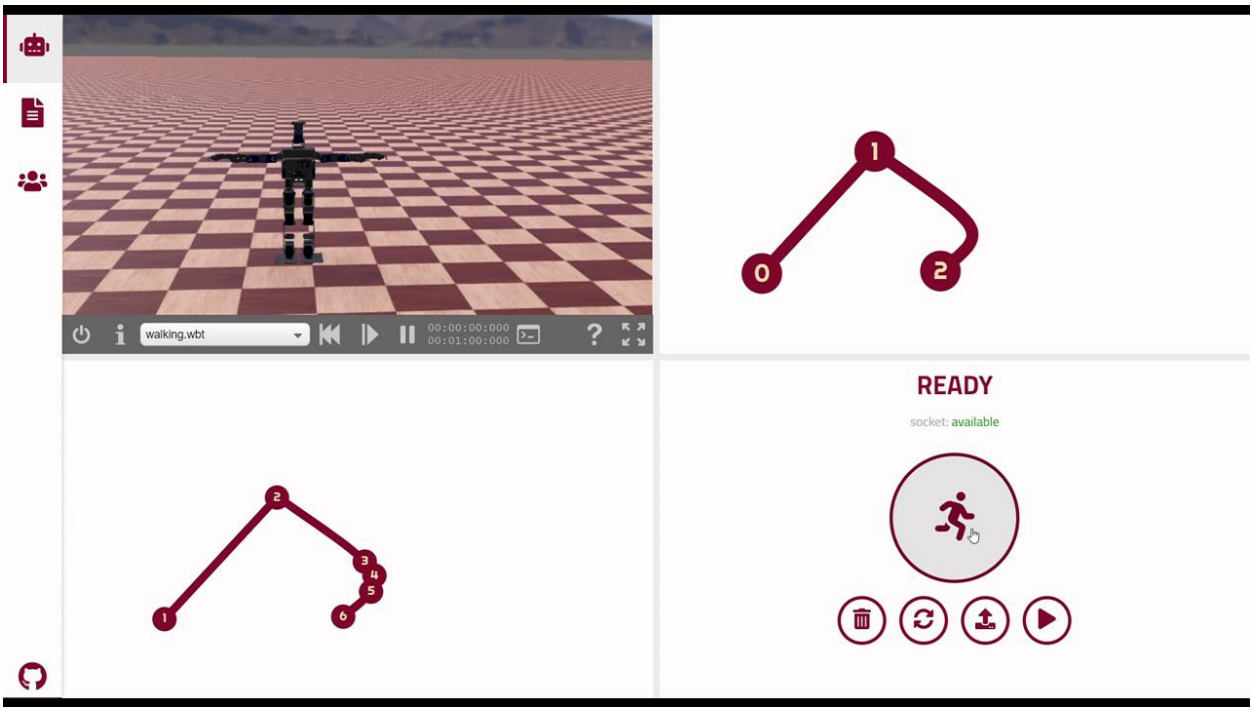


Figure 5-20 - GUI Home page (Start)

After that the robot starts walking as shown in figure 5.21.

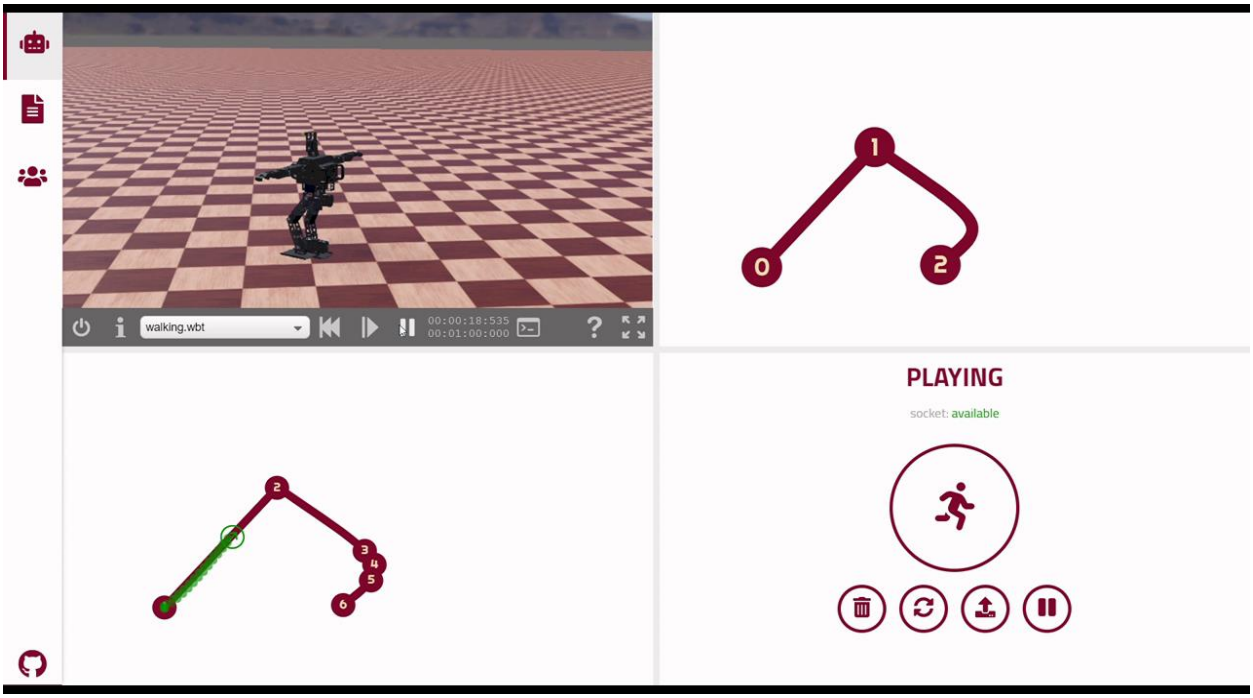


Figure 5-21 - GUI Home page (Walking)

We notice in figure 5.21 that the lower left part shows a green arrow and a green line. The green arrow is the current position of the robot and the green line is the distance that the robot cut in the path.
At the end, we expect the robot to reach the final point on the path (checkpoint 6) as shown in figure 5.21 and the path in the lower left part be completely colored in green.
Figure 5.22 shows that the robot reached the end of the path.

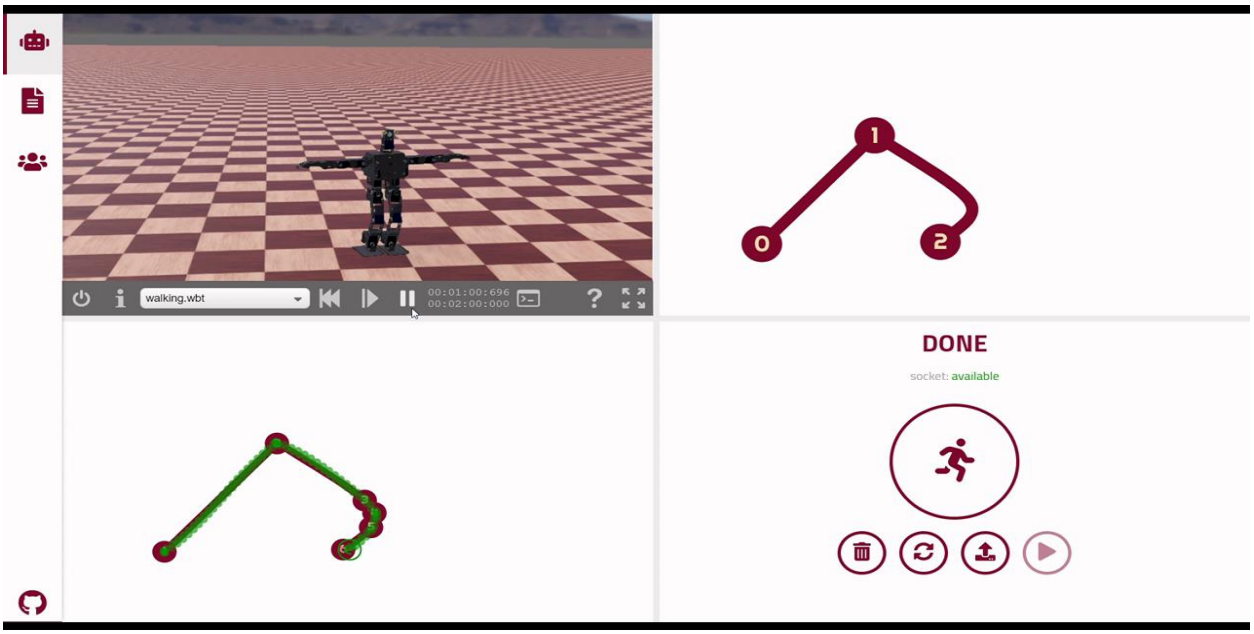


Figure 5-22 - GUI Home page (End)

When the robot reached the final point, it stopped moving as we notice in the upper left part of the home page as shown in figure 5.22. At this moment we made sure that our project works well after integrating all the modules together.

5.3. Testing Schedule

As testing our project can't depend on only one module, we started testing our projects after many modules have been done in order to make the robot walk. We started testing the project after finishing the basic modules for walking which are Robot, Inverse Kinematics and Pattern Generation module. After that we implemented the controllers module for balancing the robot on inclined planes. After that we tested these modules together in the full case scenario explained in subsection 5.2.1.3. Then we finished the User Specified Path module and tested it. At last we implemented a simple GUI to make it easier for the user to give the robot a path, we integrated the whole project together then started testing it. Below in table 5.1 the dates of testing the previously mentioned modules.

Table 5-1 - Testing Schedule

Tested Module	Testing Start Date	Testing End Date
Walking Pattern and Inverse Kinematics	21st May 2020	25th May 2020
Balancing Controllers	30th May 2020	3rd June 2020
Full motion without path	21st June 2020	24th June 2020
User Specified Path	29th July 2020	3rd August 2020
Full System		

5.4. Comparative Results to Previous Work

Bipedal robot motion differs from one robot to another, we can't really compare our work to others' work as we use different robots. The only point of comparison is the maximum angle of inclination that the robot can keep itself balanced on it. In the following table we provide some examples of previous projects of bipedal balancing on inclined planes and the maximum angle of inclination it can reach. We also provide the number of DOF (number of motors) as numbers of motors plays an important rule in balancing the robot.

Table 0-2 - Comparative Results

Project Name	Maximum Inclination Angle (in degrees)	Degrees of Freedom
Forrest (our project)	11.45	20
SURALP [12]	3.2	29
Ardubio-Walker [10]	9	12

Chapter 6: Conclusions and Future Work

In this chapter we will summarize our project and state all the challenges that we faced throughout the project and all the skills and experiences we gained. Moreover, we will suggest enhancements and future work to improve our project.

6.1. Faced Challenges

During building of this project, we faced a lot of challenges. The lack of resources as walking is a world challenging problem and needs a lot of research to reach reasonable results. One of the main challenges we faced is keeping the robot balanced in different environments mentioned before in chapter 5. Also, the sensors' errors while the robot is walking was a big one. And how we could deal with the paths defined by the user because of the localization problems.

6.2. Gained Experience

This project was one of the biggest – If not the biggest – challenges that faced us since we started our journey towards becoming Computer Engineers. It took a tremendous amount of effort and many sleepless nights, but despite all that, it was one of the most influential experiences from which we emerged enlightened.

6.2.1. Scientific Research

The process of research and planning helped us tremendously in deciding our best courses of action, the areas where we needed to strengthen ourselves and ultimately, we learnt how to actually search papers, summarize them and actually extract relevant data to our project. After reading a lot of papers, we started to develop an intuition towards analyzing the challenges mentioned in the previous section and find the solution for each challenge.

6.2.2. Software development

Working on this project was a great opportunity for us to apply what we learned throughout the years in an organized manner that would help us maintain the code over long periods of time with proper testing, documentation and organized structure.
We followed -to the best of our abilities- known Python clean code structures to avoid redundancy of code and to write readable functions that are easy to understand to process of development. In addition, we make use of GitHub tool, learned how to make a git flow to track the development process carefully.

6.2.3. Robotics Field

During the project we learned a lot about the robotics field through knowing the simulation tools and going deep into the hardware components used to build a robot, also the mechanics of the humanoid robots and how to apply the scientific concepts such as the inverse kinematics, and robot localization in a real project.

6.3. Conclusions

Forrest provides the needed algorithms for humanoid bipedal robots for walking on flat planes and inclined planes. It keeps track of the tilting angles of the robot in order to keep itself balanced. It can sense any change in the inclination angle of the plane it is walking on and sends new values to its motors to adapt to this change. It also allows the user to draw a path by using a very simple GUI and allow the user to watch the robot while following that path.

6.4. Future Work

We are planning to move our walking algorithm from the simulation environment to the hardware robot. Adding some features such as moving upstairs and downstairs, also object detection to avoid collisions. And human detection feature to make the robot more interactive and could be used widely for the education purposes or in hospitals.

References

[1] Preeti Wadhwani, Prasenjit Saha, “Humanoid Robot Market Size By Product (Wheel Drive, Biped), By Application (Retail, Hospitality, Education & Scientific Research, Healthcare, Residential, Military & Defense, Construction, Underwater Systems), Industry Analysis Report, Regional Outlook, Growth Potential, Price Trends, Competitive Market Share & Forecast, 2017 – 2024”, 2017, https://www.gminsights.com/industry-analysis/humanoid-robot-market, last accessed: 2020.

[2] QuestionPro, “Market Survey: Definition, Purpose, Importance, Types and Examples”, 2018, https://www.questionpro.com/blog/market-survey/, last accessed: 2020.

[3] UBTECH, “About UBTECH”, 2019, https://www.ubtrobot.com/pages/about-ubtech?ls=en, last accessed: 2020.

[4] Jon Phillips, “Hands on with Alpha 2, an 'interactive family robot' that tells bedtime stories and teaches yoga”, 2016, https://www.techhive.com/article/3019306/hands-on-with-alpha-2-an-interactive-family-robot-that-tells-bedtime-stories-and-teaches-yoga.html, last accessed: 2020.

[5] ROBOTIS, “About Us”, 2020, http://www.robotis.us/about-us/, last accessed: 2020.

[6] Marko B. Popovic, “Biomechatronics”, Academic Press, 2019.

[7] Luis Bermudez, "Overview of Inverse Kinematics", 2017, https://medium.com/unity3danimation/overview-of-inverse-kinematics-9769a43ba956, last accessed:2020.

[8] OMEGA, "What is a PID Controller", 2018, https://www.omega.co.uk/prodinfo/pid-controllers.html, last accessed: 2020.

[9] Shoudong Huang, Gamini Dissanayake, "Robot Localization: An Introduction", Published online: https://onlinelibrary.wiley.com/doi/abs/10.1002/047134608X.W8318, 2016.

[10] Riyanto, Carmadi Machbub, Hilwadi Hindersah, Widyawardana Adiprawita, "Slope Balancing Strategyfor Bipedal Robot Walking Based on Inclination EstimationUsing Sensors Fusion", International Journal on Electrical Engineering and Informatic, vol. 11, no. 3, 2019.

[11] Cyberbotics, "ROBOTIS' Robotis OP3",2018, https://www.cyberbotics.com/doc/guide/robotis-op3, last accessed: 2020.

[12] Evrim Taşkıran,Utku Seven, özer Koca, Metin Yılmaz, Kemalettin Erbatur, “Walking Control of a Biped Robot on an Inclined Plane”, IFAC Proceedings Volumes, vol. 42, no. 19, pp. 254-259, 2009.

Appendix A: Development Platforms and Tools

This appendix will include all the tools and platforms we used in order to build our project. We will list all hardware and software tools we used during building this project.

A.1. Hardware Platforms

A.1.1 SainSmart 17-DOF Biped Humanoid Kit

This humanoid bipedal kit is manufactured by SainSmart, it contains 17 DOF (17 motors). Its material is ultra-light hard aluminum alloy. It contains 1 DOF in head, 3 per arm and 5 per leg. Its weight is 1.65 KG and its dimensions are 505mm length, 120mm width and 320mm height.

A.1.2 32 channels steering servo drive control panel

This 32 channel Servo motor controller board for robot can control up to 32 servo motors at the same time, either by using a software on a PC, or UART communication (TTL serial port) of MCU (51, AVR, ARM, FPGA, PIC, etc.) to send commands to control the steering.

A.1.3 SainSmart SR319 digital servo

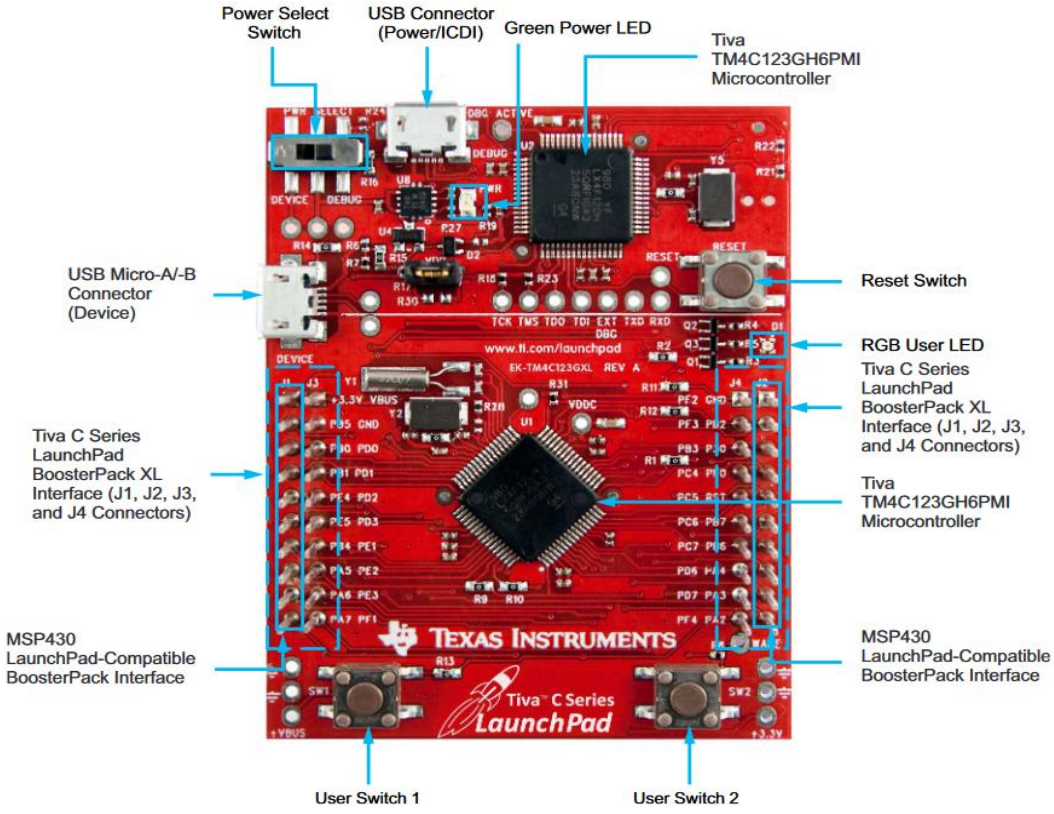
Digital servo with 1723 core motors and copper gear. Its specifications are:

- Voltage: 5-7.4V
- Dimensions: 40x20x40.5mm / 1.57x0.79x1.59 inch
- Speed:(sec/60°):
- 13Kg.cm/0.16sec/60°@5.0V
- 15Kg.cm/0.15sec/60°@6.0V
- 16Kg.cm/0.14sec/60°@7.4V

A.1.4 Tiva™C SeriesTM4C123G

Tiva™C SeriesTM4C123G is a low-cost evaluation platform for ARM® Cortex™ M4F-based microcontrollers. The Tiva C Series LaunchPad design highlights the

TM4C123GH6PMI microcontroller USB 2.0 device interface, hibernation module, and motion control pulse-width modulator (MCPWM) module. The Tiva C Series LaunchPad also features programmable user buttons and an RGB LED for custom applications. The stackable headers of the Tiva C Series TM4C123G LaunchPad BoosterPack XL interface demonstrate how easy it is to expand the functionality of the Tiva C Series LaunchPad when interfacing to other peripherals on many existing BoosterPack add-on boards as well as future products.



A.2. Software Tools

A.2.1 Webots

Webots is an open source and multi-platform desktop application used to simulate robots. It provides a complete development environment to model, program and simulate robots.

It has been designed for a professional use, and it is widely used in industry, education and research. Cyberbotics Ltd. maintains Webots as its main product continuously since 1998.

A.2.1.1 Fast Prototyping

Design easily complete robotics simulations using the large Webots asset library which includes robots, sensors, actuators, objects and materials. Import your existing CAD models (from Blender or from URDF). Import OpenStreeMap maps. Use a modern GUI to edit your simulation and your robot controllers. Save time in the development of your robotics project.

A.2.1.2 Applications

Create a wide variety of simulations including two-wheeled table robots, industrial arms, bipeds, multi-legs robots, modular robots, automobiles, flying drones, autonomous underwater vehicles, tracked robots, aerospace vehicles, etc. Set-up indoor or outdoor interactive environments. Use Webots to create robot prototypes, develop, test and validate your AI and control algorithms, teach robotics to your students, etc.

A.2.1.3 Features

Webots core is based on the combination of a modern GUI (Qt), a physics engine (ODE fork) and an OpenGL 3.3 rendering engine (wren). It runs on Windows, Linux and macOS. Webots simulations can be exported as movies, interactive HTML scenes or animations or even be streamed to any web browser using webgl and Websockets. Robot may be programmed in C, C++, Python, Java, MATLAB or ROS with a simple API covering all the basic robotics needs.

A.2.1.4 Documentation

Learn quickly the fundamentals going through the tutorial. Explore simple examples which are working out of the box. Refer to the Webots User Guide and Webots Reference Manual to get an exhaustive documentation, including the Webots nodes and the APIs to control them. Discover Webots for automobiles guide and learn how to set-up efficient vehicle simulations using integrated tools and interfaces to third party software.

A.2.1.5 Quality

Webots is robust, deterministic and well documented. To ensure code quality, every code modification is peer-reviewed and submitted to an automatic test suite

testing all the API. Backward compatibility is guaranteed and well documented between major versions. Every release is assessed by quality assurance tests conducted by humans.

A.2.2 NumPy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

At the core of the NumPy package, there is the *ndarray* object. This encapsulates n-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance. There are several important differences between NumPy arrays and the standard Python sequences:

- NumPy arrays have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of *ndarray* will create a new array and delete the original.
- The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory. The exception: one can have arrays of (Python, including NumPy) objects, thereby allowing for arrays of different sized elements.
- NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are executed more efficiently and with less code than is possible using Python's built-in sequences.
- A growing plethora of scientific and mathematical Python-based packages are using NumPy arrays; though these typically support Python-sequence input, they convert such input to NumPy arrays prior to processing, and they often output NumPy arrays. In other words, in order to efficiently use much (perhaps even most) of today's scientific/mathematical Python-based software, just knowing how to use Python's built-in sequence types is insufficient - one also needs to know how to use NumPy arrays.

The points about sequence size and speed are particularly important in scientific computing. As a simple example, consider the case of multiplying each element in a 1-D sequence with the corresponding element in another sequence of the same length. If the data are stored in two Python lists, a and b, we could iterate over each element:

```
c = []

for i in range(len(a)):
```

```
    c.append(a[i]*b[i])
```

This produces the correct answer, but if `a` and `b` each contain millions of numbers, we will pay the price for the inefficiencies of looping in Python. We could accomplish the same task much more quickly in C by writing (for clarity we neglect variable declarations and initializations, memory allocation, etc.)

```
for (i = 0; i < rows; i++): {

  c[i] = a[i]*b[i];

}
```

This saves all the overhead involved in interpreting the Python code and manipulating Python objects, but at the expense of the benefits gained from coding in Python. Furthermore, the coding work required increases with the dimensionality of our data. In the case of a 2-D array, for example, the C code (abridged as before) expands to

```
for (i = 0; i < rows; i++): {

  for (j = 0; j < columns; j++): {

    c[i][j] = a[i][j]*b[i][j];

  }

}
```

NumPy gives us the best of both worlds: element-by-element operations are the “default mode” when an ndarray is involved, but the element-by-element operation is speedily executed by pre-compiled C code. In NumPy

```
c = a * b
```

does what the earlier examples do, at near-C speeds, but with the code simplicity we expect from something based on Python. Indeed, the NumPy idiom is even simpler! This last example illustrates two of NumPy’s features which are the basis of much of its power: vectorization and broadcasting.

A.2.3 MatPlot

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Check out our home page for more information. Matplotlib produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shell, web application servers, and various graphical user interface toolkits.

A.2.4 Pickle

The pickle module implements binary protocols for serializing and de-serializing a Python object structure. “Pickling” is the process whereby a Python object hierarchy is converted into a byte stream, and “unpickling” is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as “serialization”, “marshalling,” or “flattening”; however, to avoid confusion, the terms used here are “pickling” and “unpickling”. The data format used by pickle is Python-specific. This has the advantage that there are no restrictions imposed by external standards such as JSON or XDR (which can’t represent pointer sharing); however, it means that non-Python programs may not be able to reconstruct pickled Python objects. By default, the pickle data format uses a relatively compact binary representation. If you need optimal size characteristics, you can efficiently compress pickled data. The module pickletools contains tools for analyzing data streams generated by pickle. pickletools source code has extensive comments about opcodes used by pickle protocols. To serialize an object hierarchy, you simply call the dumps() function. Similarly, to de-serialize a data stream, you call the loads() function. However, if you want more control over serialization and de-serialization, you can create a Pickler or an Unpickler object, respectively.

A.2.5 Math

This module provides access to the mathematical functions defined by the C standard.

These functions cannot be used with complex numbers; use the functions of the same name from the cmath module if you require support for complex numbers. The distinction between functions which support complex numbers and those which don’t is made since most users do not want to learn quite as much mathematics as required to understand complex numbers. Receiving an exception instead of a complex result allows earlier detection of the unexpected complex number used as a parameter, so that the programmer can determine how and why it was generated in the first place.

The following functions are provided by this module. Except when explicitly noted otherwise, all return values are floats.

math.ceil(x):

Return the ceiling of x, the smallest integer greater than or equal to x. If x is not a float, delegates to x.__ceil__(), which should return an Integral value.

math.fabs(x):

Return the absolute value of x.

math.factorial(x):

Return x factorial as an integer. Raises ValueError if x is not integral or is negative.

math.floor(x):

Return the floor of x, the largest integer less than or equal to x. If x is not a float, delegates to x.__floor__(), which should return an Integral value.

math.fmod(x, y):

Return fmod(x, y), as defined by the platform C library. Note that the Python expression x % y may not return the same result. The intent of the C standard is that fmod(x, y) be exactly (mathematically; to infinite precision) equal to x - n*y for some integer n such that the result has the same sign as x and magnitude less than abs(y). Python's x % y returns a result with the sign of y instead, and may not be exactly computable for float arguments. For example, fmod(-1e-100, 1e100) is -1e-100, but the result of Python's -1e-100 % 1e100 is 1e100-1e-100, which cannot be represented exactly as a float, and rounds to the surprising 1e100. For this reason, function fmod() is generally preferred when working with floats, while Python's x % y is preferred when working with integers.

math.gcd(a, b):

Return the greatest common divisor of the integers a and b. If either a or b is nonzero, then the value of gcd(a, b) is the largest positive integer that divides both a and b. gcd(0, 0) returns 0.

math.exp(x):

Return e raised to the power x, where e = 2.718281… is the base of natural logarithms. This is usually more accurate than math.e ** x or pow(math.e, x).

math.log(x[, base]):

With one argument, return the natural logarithm of x (to base e).

With two arguments, return the logarithm of x to the given base, calculated as log(x)/log(base).

math.pow(x, y):

Return x raised to the power y. Exceptional cases follow Annex 'F' of the C99 standard as far as possible. In particular, pow(1.0, x) and pow(x, 0.0) always return 1.0, even when x is a zero or a NaN. If both x and y are finite, x is negative, and y is not an integer then pow(x, y) is undefined, and raises ValueError.

Unlike the built-in ** operator, math.pow() converts both its arguments to type float. Use ** or the built-in pow() function for computing exact integer powers.

math.sqrt(x):

Return the square root of x.

math.acos(x):

Return the arc cosine of x, in radians.

math.asin(x):

Return the arc sine of x, in radians.

math.atan(x):

Return the arc tangent of x, in radians.

math.cos(x):

Return the cosine of x radians.

math.sin(x):

Return the sine of x radians.

math.tan(x):

Return the tangent of x radians.

A.2.6 Time

This module provides various time-related functions. For related functionality, see also the datetime and calendar modules.

Although this module is always available, not all functions are available on all platforms. Most of the functions defined in this module call platform C library functions with the same name. It may sometimes be helpful to consult the platform documentation, because the semantics of these functions varies among platforms.

An explanation of some terminology and conventions is in order:

- The epoch is the point where the time starts, and is platform dependent. For Unix, the epoch is January 1, 1970, 00:00:00 (UTC). To find out what the epoch is on a given platform, look at time.gmtime(0).
- The term seconds since the epoch refers to the total number of elapsed seconds since the epoch, typically excluding leap seconds. Leap seconds are excluded from this total on all POSIX-compliant platforms.
- The functions in this module may not handle dates and times before the epoch or far in the future. The cut-off point in the future is determined by the C library; for 32-bit systems, it is typically in 2038.
- Function strptime() can parse 2-digit years when given %y format code. When 2-digit years are parsed, they are converted according to the POSIX and ISO C standards: values 69–99 are mapped to 1969–1999, and values 0–68 are mapped to 2000–2068.
- UTC is Coordinated Universal Time (formerly known as Greenwich Mean Time, or GMT). The acronym UTC is not a mistake but a compromise between English and French.
- DST is Daylight Saving Time, an adjustment of the time zone by (usually) one hour during part of the year. DST rules are magic (determined by local law) and can change from year to year. The C library has a table containing the local rules (often it is read from a system file for flexibility) and is the only source of True Wisdom in this respect.
- The precision of the various real-time functions may be less than suggested by the units in which their value or argument is expressed. E.g. on most Unix systems, the clock “ticks” only 50 or 100 times a second.

- On the other hand, the precision of time() and sleep() is better than their Unix equivalents: times are expressed as floating point numbers, time() returns the most accurate time available (using Unix gettimeofday() where available), and sleep() will accept a time with a nonzero fraction (Unix select() is used to implement this, where available).
- The time value as returned by gmtime(), localtime(), and strptime(), and accepted by asctime(), mktime() and strftime(), is a sequence of 9 integers. The return values of gmtime(), localtime(), and strptime() also offer attribute names for individual fields.

See struct_time for a description of these objects.
Changed in version 3.3: The struct_time type was extended to provide the tm_gmtoff and tm_zone attributes when platform supports corresponding struct tm members.
Changed in version 3.6: The struct_time attributes tm_gmtoff and tm_zone are now available on all platforms.

A.2.7 Websockets

Websockets is a library for building WebSocket servers and clients in Python with a focus on correctness and simplicity.
Built on top of asyncio, Python's standard asynchronous I/O framework, it provides an elegant coroutine-based API.
The development of Websockets is shaped by four principles:

1. Simplicity: all you need to understand is msg = await ws.recv() and await ws.send(msg); Websockets takes care of managing connections so you can focus on your application.
2. Robustness: Websockets is built for production; for example, it was the only library to handle backpressure correctly before the issue became widely known in the Python community.
3. Quality: Websockets is heavily tested. Continuous integration fails under 100% branch coverage. Also it passes the industry-standard Autobahn Test suite.
4. Performance: memory use is configurable. An extension written in C accelerates expensive operations. It's pre-compiled for Linux, macOS and Windows and packaged in the wheel format for each system and Python version.

A.2.8 Threading

This module constructs higher-level threading interfaces on top of the lower level _thread module.

This module defines the following functions:

threading.active_count():

Return the number of Thread objects currently alive. The returned count is equal to the length of the list returned by enumerate().

threading.current_thread():

Return the current Thread object, corresponding to the caller's thread of control. If the caller's thread of control was not created through the threading module, a dummy thread object with limited functionality is returned.

threading.excepthook(args, /):

Handle uncaught exception raised by Thread.run().

The args argument has the following attributes:

- exc_type: Exception type.
- exc_value: Exception value, can be None.
- exc_traceback: Exception traceback, can be None.
- thread: Thread which raised the exception, can be None.

If exc_type is SystemExit, the exception is silently ignored. Otherwise, the exception is printed out on sys.stderr.

If this function raises an exception, sys.excepthook() is called to handle it.

threading.excepthook() can be overridden to control how uncaught exceptions raised by Thread.run() are handled.

Storing exc_value using a custom hook can create a reference cycle. It should be cleared explicitly to break the reference cycle when the exception is no longer needed.

Storing thread using a custom hook can resurrect it if it is set to an object which is being finalized. Avoid storing thread after the custom hook completes to avoid resurrecting objects.

threading.get_ident():

Return the 'thread identifier' of the current thread. This is a nonzero integer. Its value has no direct meaning; it is intended as a magic cookie to be used e.g. to index a dictionary of thread-specific data. Thread identifiers may be recycled when a thread exits and another thread is created.

threading.get_native_id():

Return the native integral Thread ID of the current thread assigned by the kernel. This is a non-negative integer. Its value may be used to uniquely identify this particular thread system-wide (until the thread terminates, after which the value may be recycled by the OS).

threading.enumerate():

Return a list of all Thread objects currently alive. The list includes daemonic threads, dummy thread objects created by current_thread(), and the main thread. It excludes terminated threads and threads that have not yet been started.

threading.main_thread():

Return the main Thread object. In normal conditions, the main thread is the thread from which the Python interpreter was started.

threading.settrace(func):

Set a trace function for all threads started from the threading module. The func will be passed to sys.settrace() for each thread, before its run() method is called.

threading.setprofile(func):

Set a profile function for all threads started from the threading module. The func will be passed to sys.setprofile() for each thread, before its run() method is called.

threading.stack_size([size]):

Return the thread stack size used when creating new threads. The optional size argument specifies the stack size to be used for subsequently created threads, and must be 0 (use platform or configured default) or a positive integer value of at least 32,768 (32 KiB). If size is not specified, 0 is used. If changing the thread stack size is unsupported, a RuntimeError is raised. If the specified stack size is invalid, a ValueError is raised and the stack size is unmodified. 32 KiB is currently the minimum supported stack size value to guarantee sufficient stack space for the interpreter itself. Note that some platforms may have particular restrictions on values for the stack size, such as requiring a minimum stack size > 32 KiB or requiring allocation in multiples of the system memory page size - platform documentation should be referred to for more information (4 KiB pages are common; using multiples of 4096 for the stack size is the suggested approach in the absence of more specific information).

A.2.9 React Native

React Native is an open-source mobile application framework created by Facebook, Inc. It is used to develop applications for Android, iOS, Web and UWP by enabling developers to use React along with native platform capabilities.

The working principles of React Native are virtually identical to React except that React Native does not manipulate the DOM via the Virtual DOM. It runs in a background process (which interprets the JavaScript written by the developers) directly on the end-device and communicates with the native platform via a serialization, asynchronous and batched Bridge.

React components wrap existing native code and interact with native APIs via React's declarative UI paradigm and JavaScript. This enables native app development for whole new teams of developers, and can let existing native teams work much faster.

React Native does not use HTML or CSS. Instead, messages from the JavaScript thread are used to manipulate native views. React Native also allows developers to write native code in languages such as Java for Android and Objective-C or Swift for iOS which make it even more flexible.

A.2.10 Electron

Electron is an open-source software framework developed and maintained by GitHub. It allows for the development of desktop GUI applications using web technologies: it combines the Chromium rendering engine and the Node.js runtime. Electron is the main GUI framework behind several notable open-source projects including Atom, GitHub Desktop, Light Table, Visual Studio Code, and WordPress Desktop.

Electron applications are composed of multiple processes. There is the "browser" process and several "renderer" processes. The browser process runs the application logic, and can then launch multiple renderer processes, rendering the windows that appear on a user's screen rendering HTML and CSS.

Both the browser and renderer processes can run with Node.js integration if enabled.

Most of Electron's APIs are written in C++ or Objective-C and then exposed directly to the application code through JavaScript bindings.

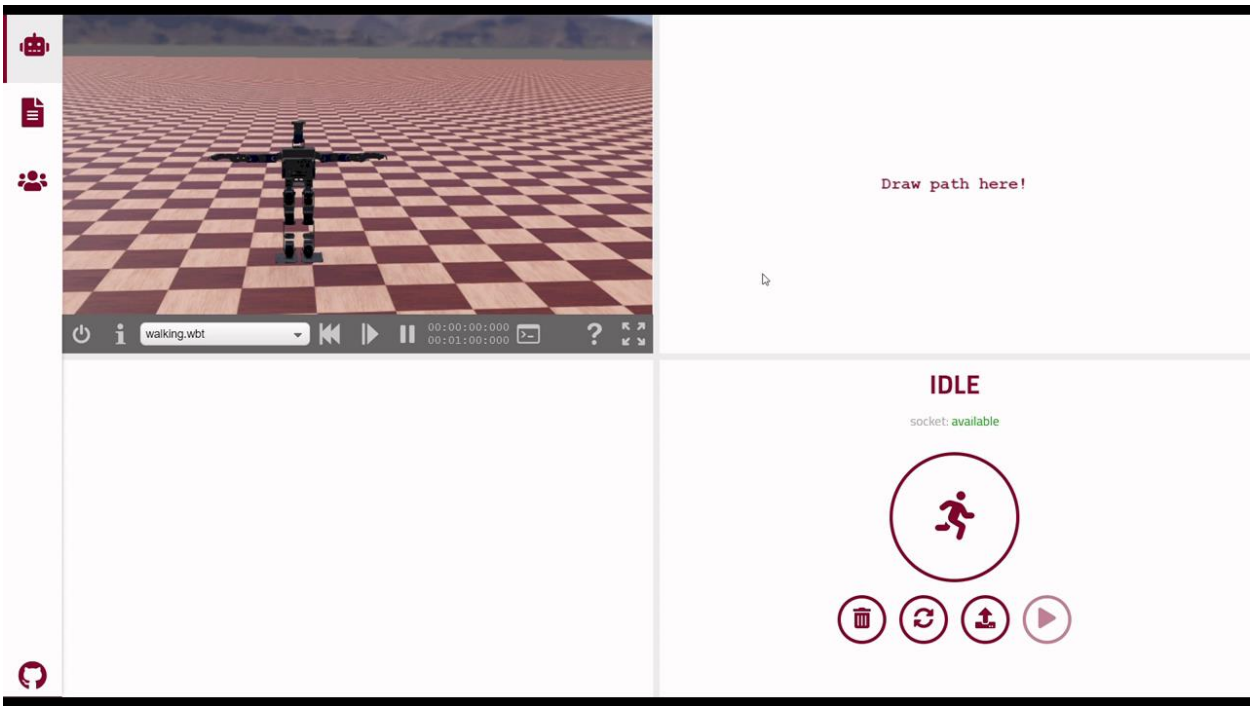
Appendix B: User Guide

In this appendix, we are going to talk about the simulation GUI and how to use it step by step. Once you open the program, you will see the main view as shown in figure A.1, the first field is for the server host address, and the next one is for the port, they are containing the default configuration values to connect to Webots server.

First, you have to run the simulation server at Webots directory using the command line:

webots --stream --minimize –batch

then you should press connect button, after pressing the button, the view shown in the next figure will appear, which is the main view in the simulation program.



B.1. Simulation tab

The simulation window is divided into four sections, now we are going to explain each section in details:

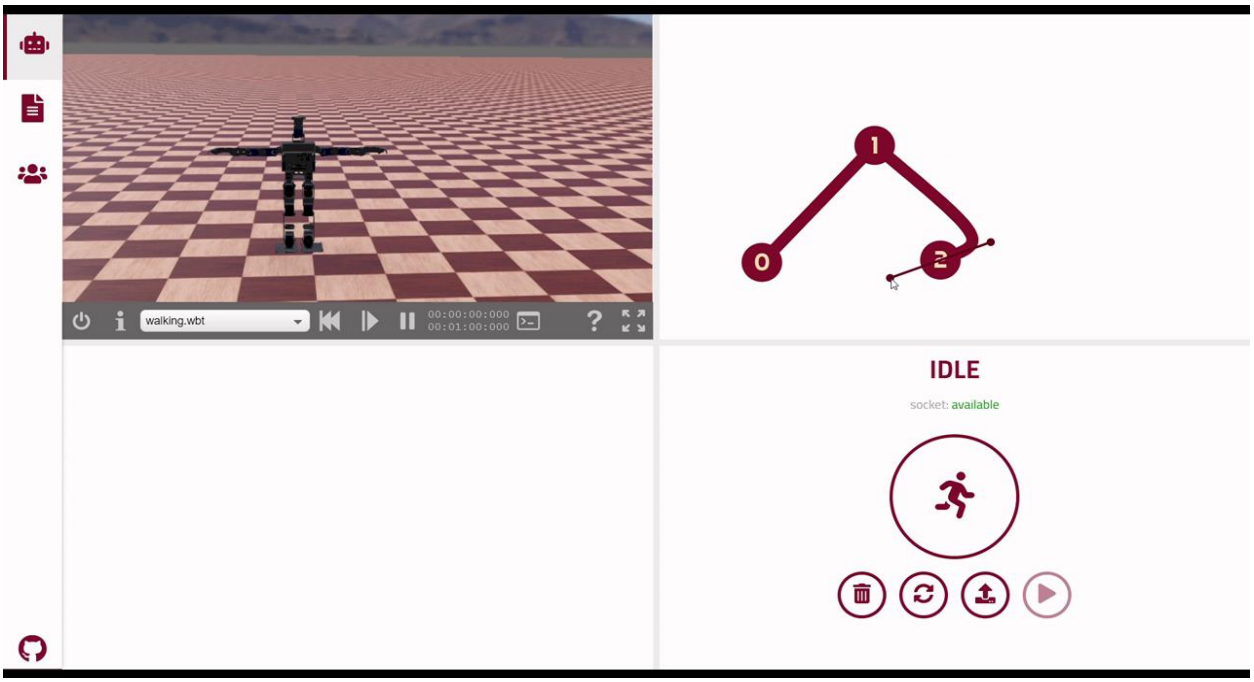
B.1.1 Simulation stream window:

This is the top left quarter, which contains the streaming window that streams the simulation from Webots. It contains many buttons you can use to control the simulation behavior. Now we will describe the functionality of the most important ones:

- The first button from the left is used to close the simulation.
- The menu that contains “walking.wbt” is used to choose any environment you want to simulate on (flat or incline floor).
- The button at the right of the menu explained above is used to restart the simulation from beginning.
- The most left button is for full-screen mode.

B.1.2 Path drawing window

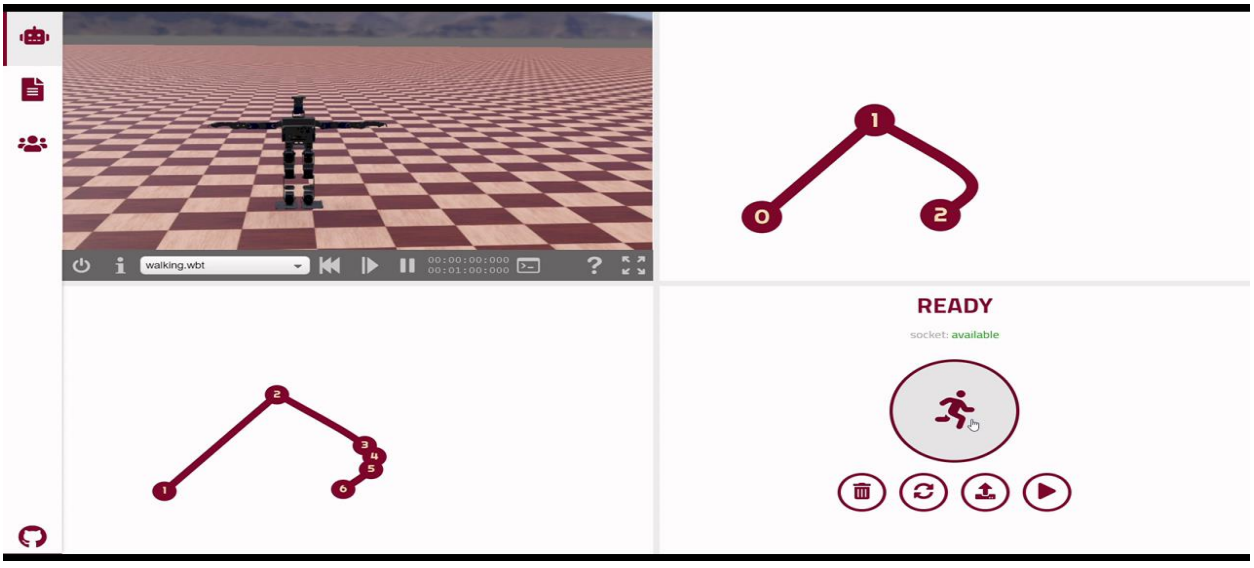
This is the top right quarter, which is used to insert the path you want the robot to walk on. You can draw the path by pressing the mouse’s left button, the first point will determine from where the robot should start, any next point you draw will make a line, if you want to make an arc, you have to drag while pressing the mouse’s left button.
After drawing your path, the window will look like the window in the next figure.



B.1.3 Robot check points window

This is the bottom left quarter, once you draw the path and send it to the server (we will show later how to send your path to Webots server) this section will contain the current location of the robot and the progress of walking on the drawn path. As shown in the next figure:

- The arrow indicates the current location of the robot
- The green dots are the previous locations of the robot
- The numbered points are the check points that the robot determine to follow the given path



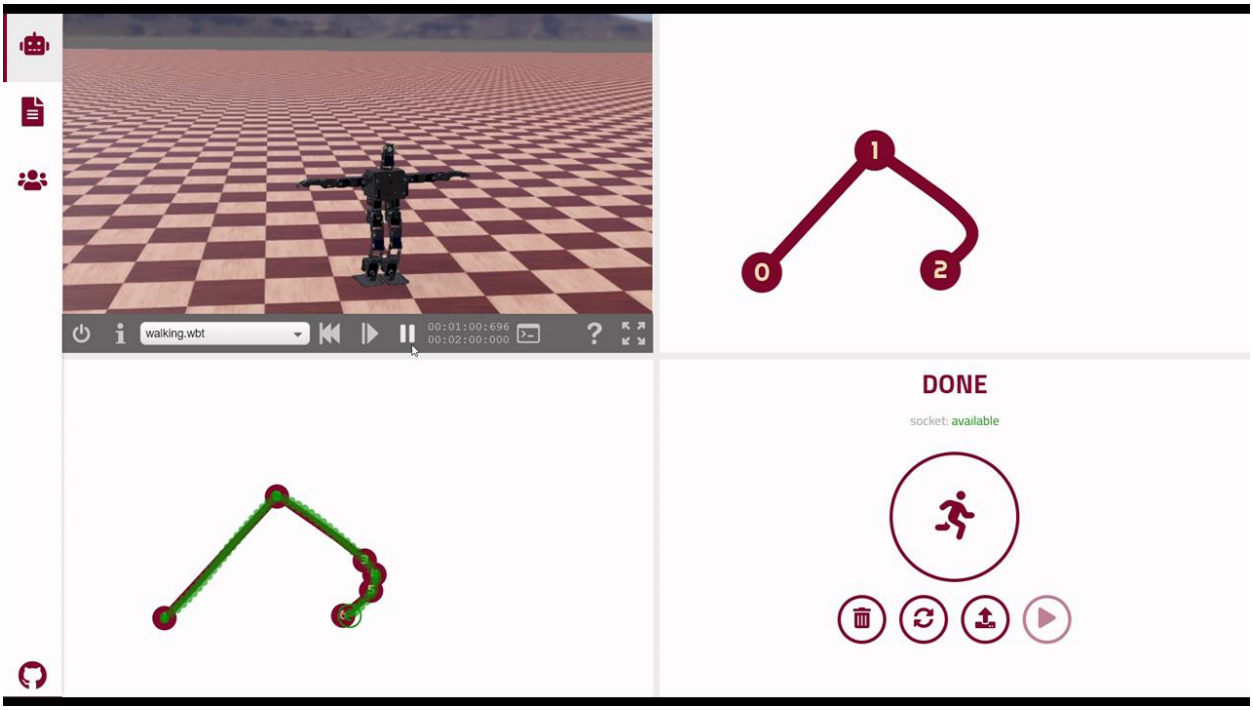
B.1.4 Control panel window

This is the bottom right section, which contains two main things:

B.1.4.1 The state of the simulation:

The word “READY” in the previous figure indicates the state of the simulation, the state of the simulation can be one of these values:

- **IDLE**: which indicates that the GUI is making the connection with the server.
- **READY**: the state becomes READY when you draw the path and send it to the robot, which means that the robot is ready to start moving.
- **PLAYING**: which indicates that the robot is walking on the path right now.
- **DONE**: which appears when the robot reaches the end of the path, as shown in the next figure.



The word “Socket: available” shown in figure indicates the state of the socket established between the GUI and Webots server, it takes two values:

- **Available**: when the server is running and the connection is established
- **Unavailable**: when the connection isn’t established, or the GUI sends or receives a message with the server.

B.1.4.2. The control buttons:

Those buttons are used to communicate with Webots server and controls the path drawing area:

- **Run button:** after creating the path you want the robot to walk on, you should use this button to send the path to Webots, after that, the robot will response with the check points graph as shown in figure A.4.
- **Trash button:** used to clear the drawing path area, in case you want to re-draw a new path.
- **Reset button:** used to restart the simulation from the beginning.
- **Upload button:** used to upload any saved svg path to the path drawing area.
- **Play button:** and the last button is used to start or pause the simulation.

Appendix C: Feasibility Study

The financial projections for our project cash flow are highlighted in the table below. And here are some expected Capital Expenditure (Capex) which are one-time spending like costs of Computers Equipment and long-term software licenses. Also, there are other expenses called Operational Expenses (Opex), these types of expenses are recurring payments for Salaries paid for employees, Rent, Insurance, Repairs and maintenance, Advertising, Sales, and Direct material cost.
During our first year we will hire 3 robotics engineers with $7500 annual salary and we expect the production would be only 80 Unit with 25% loss in the production as In the first year we will test the quality of our product and do the market-fit, also we have high cost due to the Capex. During the Second year we will increase our production to 160 units with the same amount of loss 25% as we don't have the ability to hire new engineers so with high production it 'snot expected for better performance, also we will increase our marketing campaigns to $30k to reach more customers as we now have a place in the market. During the third year, the production will be 320 Unit with 15% loss, our engineers now have experience so the mistake percentage would be less than the first two years, our marketing budget also would go hire to $45k. In the fourth year, we will hire 2 more engineers to increase the production as we now more stable in the market, our production will be 640 Unit with 10% loss due to the hardware issues. Our marketing budget would go to $60k and as shown in the table we will reach the breakeven point in this year. In the fifth year we will hire 1 more robotics engineer so now we have team of 6 engineers, our production will be 1280 Unit with 10% loss and our marketing budget would go up to $100k so we could reach more customer segment after gaining the trust over the first four years.

Measure	Year 1	Year 2	Year 3	Year 4	Year 5
Computer Components	$2200				
Long Term licenses	$5000				
Salaries	$22500	$22500	$22500	$37500	$45000
Rent	$3800	$3800	$3800	$3800	$3800
Repairs and Maintenance	$5000	$5000	$5000	$5000	$5000
Advertising	$15000	$30000	$45000	$60000	$100000
Direct Material Cost	$19200	$38400	$76800	$153600	$307200
Total Cost	$72700	$99700	$153100	$259900	$420500

Total Sales	$30000	$60000	$136000	$288000	$576000
Cash Flow	-$42700	-$39700	-$17100	$28100	$155500