Group 6 - Interim Report

Modelling of a MATLAB Quadruped

By Lucas Argyrou(200121613),George Craft(200326269),Ollie Burge (200326595), Jenna Hazard (200326872), Jamie Whitaker(200332345), Ollie Rawlings (200327097) and Hari Nanda(200121196)

EXECUTIVE SUMMARY

A prevalent issue in industry is the cost of funding and time provided to testing robotic systems with physical prototypes. Through digital modelling of the device and its environment, virtual testing can be conducted to verify the functions of a system before a physical device is even built. Quadruped systems in the current market are very expensive.

A recent development for MATLAB Simulink is the integration of Unreal Engine scenes for testing dynamic systems in a 3D environment. This project aims to create a system using this new development to simulate realistic scenarios in aid for companies interested in robotic quadrupeds. Then, demonstrate the capabilities of this system in a simulated environment to ensure it works as expected.

In terms of key findings of this project, it was discovered that simulation and sensing data could be provided back from the Unreal Scene, simulating a real device taking in inputs from its environment, thus the controller can function as if it were part of a physical device. Furthermore, the Unreal Environment could be designed to simulate all types of terrain a quadruped would be expected to navigate, and potentially model conditions of other planetary bodies for testing robotics used in space missions.

The requirements of the system consisted of the quadruped tracking a ball with computer vision, picking up said ball at a starting position with its manipulator, and traversing uneven terrain using both computer vision and initial path planning. The system will also be able to climb a staircase and place the ball at the ending position. This was achieved through integration of a feedback controller for the quadruped gait, navigation modules (such as the path planning and computer vision), all functioning with a fully realised Unreal Engine model of a quadruped and the terrain.

CONTENTS

1 INTRODUCTION

Prototyping and testing are becoming more and more necessary as the demand for quadrupeds in industry rises. The present issue is that because quadrupeds are such a unique concept, they are incredibly expensive. By combining the hyper-realistic physics engine of Unreal with the computing power of MATLAB, the project aims to assist these businesses by creating a modelling environment that lowers the cost of prototyping.

To demonstrate the modelling system's capabilities, the project included a simulation of a quadruped and manipulator which can detect a ball using computer vision, navigate through tough terrain using computer vision and path planning, and then climb a set of stairs where the ball will be dropped by the manipulator at a predetermined drop off point.

The motivation behind the project was to firstly, reduce the costs in prototype and development for quadrupeds. This is necessary as robotic quadrupeds are a novel concept, the state-of-the-art manufacturers are both Boston Dynamics with their flagship ‘spot’ and Ghost Robotics with their flagship ‘Vision60’; which cost $75,000 and $150,000 respectively to purchase [1]. The exact cost to manufacture both devices isn’t publicly disclosed by either of the companies. However, due to the complexity of its design and the specialised components within them, these quadrupeds are likely to incur large costs to their respective manufacturer. Therefore, having a modelling environment would replace the need to use physical prototypes.

Another motivator for the project is to allow companies to validate real applications of quadrupeds via simulations. An example of this is a delivery company that may wish to test the effectiveness and efficiency of delivery quadrupeds. The quadruped can simulate picking up the parcel then using a satellite image that will be fed into the pathfinding algorithm to locate the final location and route the quadruped should take. The computer vision can be used to detect any unexpected obstacles that may appear in its path. They can then compare this simulation with current delivery times before investing large amounts of capital in physical quadrupeds and testing.

The final motivator is to simulate unobtainable environments. An example of this is extra-terrestrial terrains, for example sending a quadruped to the moon. Due to the different gravity and terrain conditions, this environment cannot be physically simulated in real time on earth. Therefore, the use of simulation software allows a system to be simulated under the physical effects of these unobtainable environments without being resource taxing or complex. The current quadruped systems are tested and prototyped using a variety of methods which will be assessed further in the report these include Physical prototypes, Unreal Engine, SimScape, Gazebo and Unity.

The project was carried out following the V-model as it uses a simple framework which makes delegating tasks and tracking progress easier. This model was abstracted into five sections: Con-ops and requirements, Detailed design, Implementation, Project integration and testing and Project verification and validation.

The report is broken down into:

- The introduction – introducing background, aims, objectives and motivation of the project.
- The systems engineering – Covering literature, Con-Ops, Requirements, Work breakdown and Organisation structure, and project risk register.
- The technical aspects of the project which will be broken down into the following chapters:
    - Environment generation – Creation of the terrain whining Unreal that the system will be simulated in.
    - Kinematic and simulation modelling – the modelling of the quadruped and manipulator
    - Gait modelling – the modelling of the movement
    - Path planning – Finding a collision free path from a chosen start position to a selected end position.
    - Computer vision - Detecting objects for traversal and selection decisions.
    - Stability analysis – Balancing of the quadruped.
- The integration of subsystems – How the subsystems were integrated together to make the final demonstrator.
- overview of final demonstrator – The final product and how the requirements were met.
- testing, verification, and validation – The testing that has been undertaken and the purpose behind them, followed by a critical appraisal.
- Conclusions and future work – key conclusions and ideas for future work.
- references.
- Appendix.

Most of the information needed to finish the assignment comes from the MathWorks and the Unreal Engine website. Furthermore, all the simulations and code mentioned in the document can be found on the following link on GitHub.

2 SYSTEMS ENGINEERING

Systems engineering is crucial within any engineering project, especially for group projects to ensure that every member has an equal understanding of the shared end goal.

2.1 LITERATURE REVIEW

The purpose of this section is to provide an overview of the current state-of-the-art solutions to the key topics that the project will branch into. This is done to acquire a better understanding of the potential challenges that are likely to occur, and the potential remedies for these challenges so that they can be implemented within the project. The main topics of research, which include Terrain generation, Kinematic and simulation modelling, Gait modelling, Path finding, Computer vision and Stability analysis, are to be elaborated upon in the individual chapters. For each of these topics, multiple documents will be utilised to compare different methods of achieving a similar goal.

The project makes use of the MATLAB 3D Simulation Block set in Simulink for the purpose of sending data between Simulink and Unreal Engine. Unreal Engine also required the MathWorks Interface plugin to connect the scene to the Simulink program. One paper [72] discusses the usage of these recent addons for the purpose of simulating automated grain offloading. The project in this paper greatly benefits from the LiDAR and camera sensors set up in Unreal to simulate a physical system using sensing devices, and the Unreal Scene allows for virtual verification of the system's functionality without needing to use a physical device, as this would take much longer to test and would be more expensive. Furthermore, in both papers [73] and [74] the process of bidirectional communication between a MATLAB system and the Unity game engine are discussed. This demonstrates further the possibility of linking model-based simulators into game engines for testing purposes, where the real environment is recreated in 3D software simulating realistic physics in place of physical prototyping. The modularity of these systems allows for seamless connection to a real device after testing the virtual model in the game engine.

Both the Vee/V-Model and the waterfall method are both sequential models. However, the V-model is more suitable for shorter projects where requirements are well defined [2] whereas, the waterfall method is more suitable for large changing teams to ensure a common goal is achieved [3]. The V-Model's drawbacks include its poor suitability for projects that last a while, yet this wasn't a major issue for the project because it only lasted a single academic year. The waterfall model has a high delivery time and often needs a hard reset in the event of a plan change [4], this is a significant drawback for the project due to tight time limitations. In conclusion, although the V-model is more complicated than the Waterfall approach, it does not prevent backtracking, which is a crucial aspect of the project due to the inevitable mistakes and changes during the

project [4]. Additionally, using the V-model provides a greater assurance of success and allows multiple engineers to work simultaneously on distinct areas that will eventually be combined. Finally, the V-Model debugging approach is preferred since it can be done in between phases instead of at the end, which results in fewer faults being discovered during the testing phase and speeds up implementation.

2.2 CONCEPT OF OPERATIONS

Con-Ops are a systematic approach when identifying the initial need for the system. These can be broken down into six main questions with regards to the stakeholders. These questions are who, what, when, where, why, and how; shown in Table 1.

Table 1 – ConOps Identification

Who	The project's stakeholders are companies that are interested in quadruped robotics. This can be a variety of industries including inspection, assembly, agriculture, search and rescue, and many more.
What	Stakeholders can simulate a modelled scenario of their choosing before physical prototyping and testing.
When	This model will be available to the stakeholder 24/7.
Where	The stakeholder will be able to access the model using PCs and laptops that are compatible with MATLAB, Simulink, and Unreal Engine 4.
Why	To reduce cost in prototyping and development of quadrupeds, allow companies to validate real applications of quadruped, and simulate unobtainable environments.
How	The Stakeholder will be able to change the environment through Unreal and change the implementation of the simulation using Simulink.

Throughout the project decisions needed to be made as to which approach to take. To do this Pugh matrices were used due to them being a powerful tool for identifying the optimal solution from multiple alternatives.

The initial decision that needed to be made was the software that the environment was generated in. The Pugh matrix in table 2 evaluates five possible simulation techniques based on their realism and complexity

within the scope of this project. These include Physical prototype, Unreal Engine, SimScape, Gazebo and Unity. The metrics these were being scored on included Physics engine (how well forces are simulated on an entity), Time consumption (how long the environment would take to create), Operational complexity (how complex the simulation technique is), adaptability to environment (how adaptable the conditions are to mimic different environments), and MATLAB compatibility (how compatible the simulation technique is with MATLAB).

Table 2 – Simulation Techniques Pugh Matrix

	Physical Prototype	Unreal Engine	SimScape	Gazebo	Unity
Physics Engine (2)	0	+1	-2	+1	+1
Time consumption (2)	0	+2	+1	-2	+2
Operational Complexity (1)	0	+2	+2	-3	+2
Adaptability to Environment (1)	0	+2	+1	0	+2
MATLAB compatibility (3)	0	+2	+3	+1	-1
Total	0	+16	+10	-2	+7

2.3 REQUIREMENT CAPTURE

To create the requirements for the simulation of a quadruped robot with a manipulator, an affinity diagram was created during brainstorming sessions from the perspective of the possible stakeholders. These ideas were then organised into categories relating to the basic and high-level objectives that have been stated in the introduction. The requirements were then ranked on importance (1 = very important -> 5 = not important) and feasibility (1 = very feasible -> 5 = not feasible) as seen in table 3 to ensure relevance and producibility in the final requirements.

Table 3 - Requirement topics from the Affinity diagram ranked by importance and feasibility.

Requirement Topic	Importance	Feasibility
Mobility	1	1
Balance	1	2
Manipulator	3	3
Sensing and Environment	2	3
Speed and Tuning	4	2
Sensor Performance	4	4
Manipulator Performance	2	2
Kinematics and Dynamics	1	2
Control Systems	2	2
Sensor Modelling	1	3
Terrain Modelling	2	1
Integration with Simulation Tool's	1	3

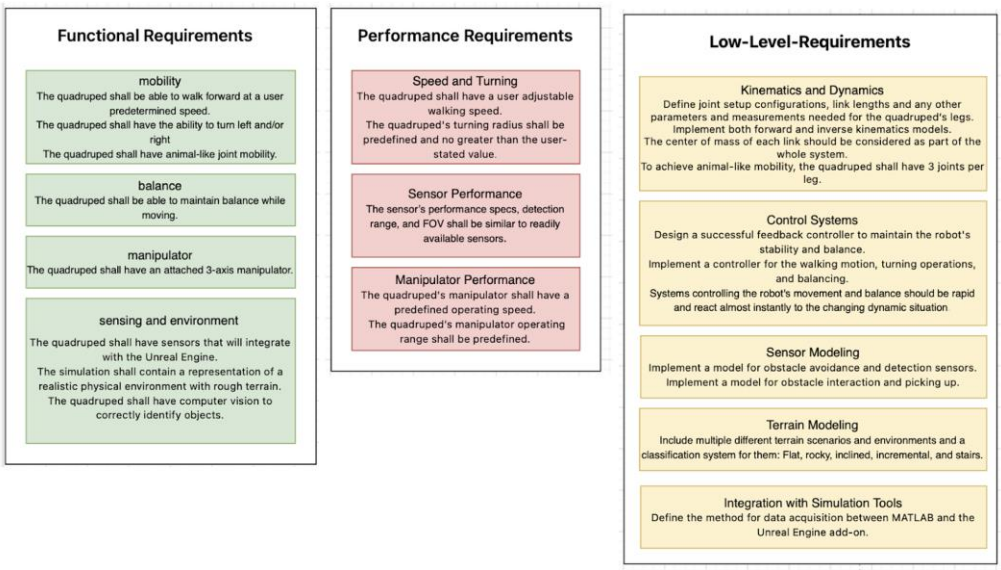

Figure 1 - Affinity diagram to brainstorm the requirements of the quadruped robot with a manipulator.

An affinity diagram was preferred for the requirement capture process over other processes such as use cases. This allows for the exploration of a wide range of requirements from diverse stakeholders, which is a priority for the project as it has a wide scope of possible future uses, therefore ideas could be taken from multiple possible stakeholders.

2.4 TECHNICAL REQUIREMENTS

Requirements are a very important part of systems engineering and can be split into two sections: Functional and non-functional. Functional requirements describe what the system should do and act as a roadmap for development to ensure that the system meets user expectation. Non-functional requirements on the other hand specify constraints on the functions offered by the system.

For each requirement an ID, Verification method, and Verification owner is assigned. An ID is useful for ease of referencing and traceability. A verification method is used to have a clear method in viewing if this requirement is completed. This is useful for stakeholders to easily see how the project is going. Finally, a verification owner is beneficial as it assigns clear accountability for the accomplishment or failure of that requirement.

2.5 WORK BREAKDOWN STRUCTURE

To create the work breakdown structure the V-model was used as previously stated. This was abstracted into project definition, project implementation, project integration and testing, and project verification and validation.

Figure 96 in the appendix shows the full work breakdown structure following the abstracted V-model.

The work Breakdown Structure is split into 4 key stages abstracted from V-model, Starting with Project definition, within this section project Con-Ops, requirements capture, systems engineering, and project management is conducted further breaking down each section with sub tasks if required. The second stage is Project implementation, the breakdown of the tasks of the project is encapsulated here, highlighting the core technical stages needed for a complete simulation and role allocation. The final two stages consist of Project Integration and testing and project verification and validation, conducting all testing and verification of requirements and how the systems integrate with each other.

2.6 GANTT CHART

The Gantt chart was produced following the previously stated abstraction of the V-model, with each section broken down into subsections then into further subdivisions all with allocated leaders, completion

percentages, and start and end dates. A further view of the Gantt chart can be found in figure 97 in the appendix and a link to the Gantt chart can be found here.

This is useful as it helps plan a project in advance as well as helping individuals stay on track. This is because you can clearly visualise the deadlines and which individual is responsible for each task. Normally tasks will be required to be done in a specific order to achieve the end goal, such as parallel goals where they can happen at the same time or sequential goals which require one to be completed prior to the next one being started.

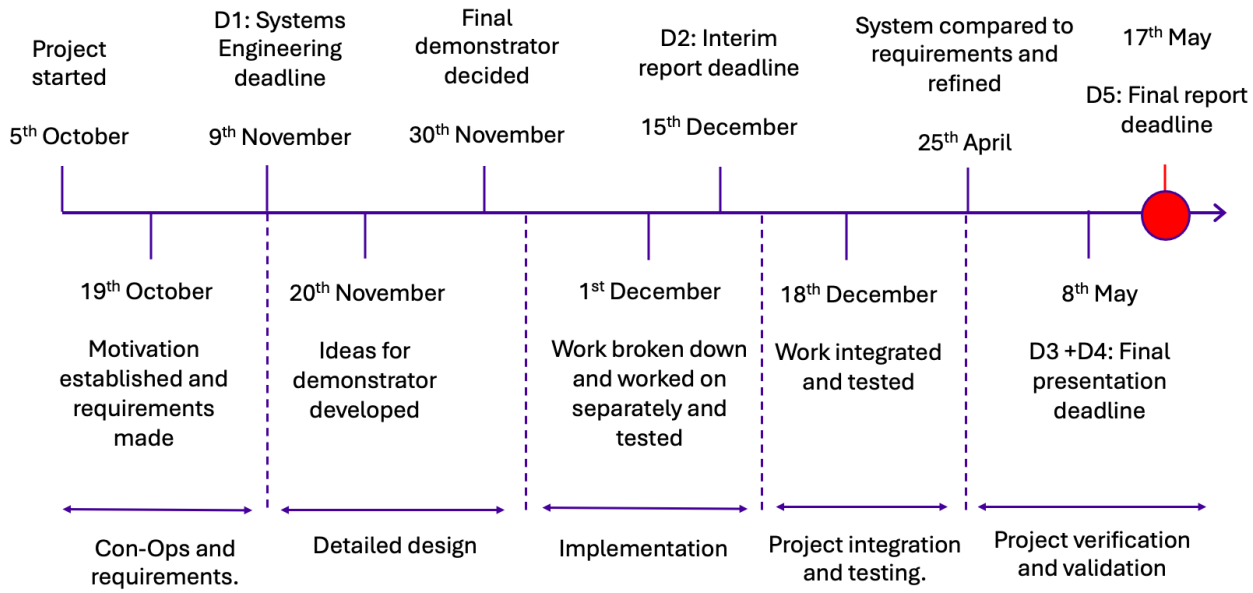

Figure 2 – Project Timeline

In addition, a project timeline was created, as seen Figure 2, that highlights key events from the Gantt chart and divides the project definition into two distinct v-model sections: Con-ops and requirements, and detailed design. This is helpful for monitoring important deadlines to guarantee productivity.

2.7 ORGANISATION STRUCTURE

To create an organisation structure, the project was first broken down into six main components shown in Figure 3. These were the subsystems of the project that were integrated together to make the final demonstrator.

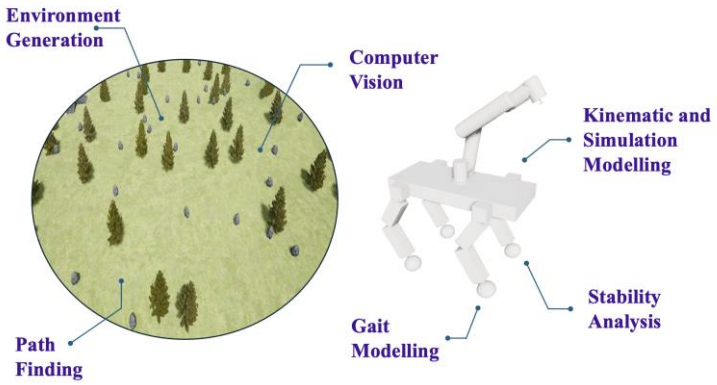

Figure 3 – Sub-system breakdown

With these subsystems thought out, roles were allocated to ensure clarity and organisation which reduced confusion and overlapping efforts. Furthermore, this breakdown also provides accountability, which builds trust among team members, creating a better working environment.

Like in every project, an operational leader was tasked with supervising the integration and coordination of each subsystem, this was to guarantee efficient operation and good team dynamics. This position was assigned to Lucas Argyrou based on his strengths in this role. The role of environment generation was applied to Ollie Rawlings. The role of path planning was applied to Jenna Hazard. The role of computer vision was applied to George Craft. The role of lead kinematic and simulation modelling was applied to Jamie Whitaker. The role of gait modelling was applied to Ollie Burge. Finally, the role of stability analysis was applied to Hari Nanda.

2.8 PROJECT RISK REGISTER

The risk register for this project clearly identifies both the physical and developmental risks that may arise over the course of the project. For each risk, the level of impact and the probability of the risk occurring is considered and the mitigation action required is provided. For the risk register shown in Table 5 in the appendix, the physical risks are given first, followed by the developmental risks.

3 TECHNICAL CHAPTERS

3.1 ENVIRONMENT GENERATION (OLIVER RAWLINGS - 200327097)

Introduction

Environment generation allows unlimited possibilities of creating a virtual environment to simulate a quadruped system within, whether that be simulating a busy urban area or simulating an uneven rural area. Environment generation provides an incredible advantage when testing systems due to its ability to simulate environments that may not be achievable otherwise, such as an extraterrestrial environment.

The environment generation was identified as a key area of the project as it would be providing the terrain and environment that the quadruped would have to traverse through. The terrain had to include enough complexity in order for the system to be simulated to the fullest of its capability, whilst not being so complex that it gave the system an impossible task. As previously mentioned, Unreal Engine was chosen for the environment generation due to its MATLAB compatibility as well as being equipped with a hyper-realistic physics engine.

Literature Review

Looking at current literature [5] it states that the way forward for product development is using virtual simulation tools and techniques within a computer-generated environment, one of these methods being virtual manufacturing. The use of these virtual tools and techniques allows more designs to be modelled that were not previously capable of being modelled, as well as reducing resource loss (time and cost). An article [6] addressing Virtual Manufacturing (VM) states that VM allows companies to simulate, improve and enhance manufacturing processes before they are carried out, whether that be in design, analysis, simulation, optimisation, or decision making. VM can be applied to numerous processes within numerous domains - such as automation or aerospace - due to its visualisation purposes and interactive manner, with one simulation software (Unreal Engine) being used in design driven development for systems in companies such as GMC HUMMER EV [7].

Similarly, another article [8] emphasises the significance in engineering for modelling a project in a computer-generated environment before execution. The proposed system would allow engineers to interact with an immersive environment via the use of a virtual reality headset, with the environment being created within Unreal Engine 4. In the context of this article [8], the system is being implemented in construction, architecture and engineering allowing the user to visualise and interact with a proposed idea without the worry of injury or wasting resources.

As previously mentioned, simulations have a variety of different uses and applications. One article [9] identifies the application of Unreal engine when simulating an earthquake, and how current simulation methods focus on single building damages rather than a more complex setting - capable in Unreal - such as an urban environment. In the case of this study real seismic waveform data is input in Unreal simulation with the data retrieved from this simulation being applied in the training of AI and robotic rescue missions. On the other hand, a different research article [10] looks into the use of Unreal for teaching purposes in practices that have a lack of hardware available. The article presents learning Weft-knitting Engineering based on Unreal Engine due to its algorithms and mathematical models, presenting a more convenient and immersive experience to that of a real life one.

Other studies show a capability for Unreal to be used for automation processes, with a paper [11] detailing this by implementing an autonomous system within the Unreal environment. The paper elaborates on numerous of Unreal's features such as its hyper realistic physics, where it explains the integration of external forces acting on an entity such as gravity, as well as touching upon the different physics engines employed in the software.

In addition, Unreal engine is compatible with numerous external software's, with one of these external software's being MathWorks. MathWorks allows data to be retrieved from an environment in Unreal (via the use of sensors [12]) as well as sending data back to Unreal, this allows Unreal to simulate a system with MathWorks computational power controlling the system. Articles [13][14] use a built-in MathWorks tool, known as driver-in-loop simulator, with Unreal engine 4 in order to create an immersive simulation that analyses human actions. With the use of Unreal an immersive interface was able to be created, with human actions being analysed via a deep neural network within MathWorks. The use of the immersion of Unreal allows the participant to act as they would do in the real-life situation, allowing accurate and precise data to be retrieved by MathWorks.

The literature and case studies reviewed above support because using Unreal Engine is the most appropriate software for the goal of the project, whilst providing some important context and relevant examples of Unreal Engine.

Requirements

Numerous requirements were identified in order to make sure the generated terrain met the required complexity and expectation of this project. These requirements (as previously seen in Table 4) are as follows: terrain shall include random uneven gradients, this shall allow the quadrupeds balancing algorithm to be implemented alongside the kinematics of the system; Terrain shall include obstacles, this shall allow the systems pathfinding/computer vision algorithm to be used to traverse through the objects; Obstacles shall

consist of rocks and trees, maintains immersion and realism of the terrain; Terrain shall include end structure, this provides the path planning with an end location allowing it to map a path through the terrain; End structure shall have stairs leading to a platform, allows kinematics controller to be used to ascend stairs, with balancing making sure quadruped is stable; End structure shall have a table to pick up and drop off ball, allows manipulator to be put to use to pick up and drop off ball; Terrain should have randomly placed obstacles and end structure, allows system to path plan to random end location whilst using computer vision to avoid randomly placed obstacles; Terrain should have dynamic moving obstacles, this allows for the computer vision to be used to avoid unexpected obstacles.

Main technical content

After identifying the requirements for the terrain an appropriate software had to be chosen in order to create, test and develop the terrain and environment. As seen in software Pugh matrix (Table 2), Unreal engine was a clear winner due to its software complexity/environment potential trade-off whilst also posing strong MATLAB compatibility, which is used for all computational processes.

In order to create a realistic environment, what type of texture should be used for the ground of the environment needed to be decided, with the texture of choice being grass with hills showing patches of rocks at certain heights. In order to make this possible Unreal Engines blend functions were implemented with its material attributes and landscape function to make the custom texture.

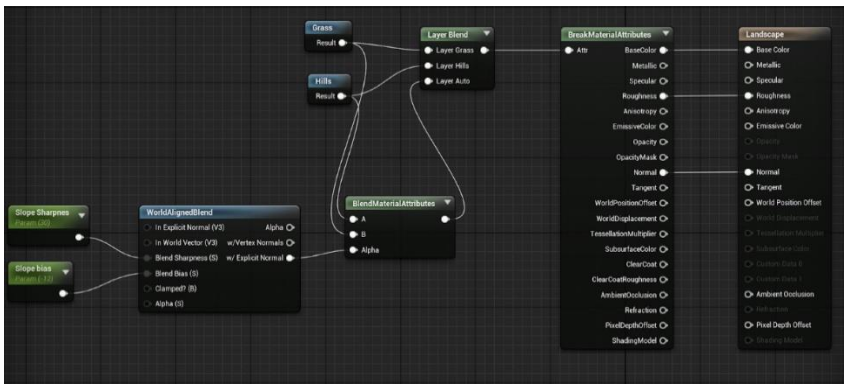

Figure 4 – Blueprint for Terrain Texture

As seen in Figure 4, Unreal's premade textures were used (stored in the Grass and Hills references) to create the visual look of the environment's terrain. The variables Slope Sharpness and Slope bias allowed control over how visible the hill texture was compared to the grass texture. In addition, it also allowed control over at what point the hill texture started to appear on uneven terrain. Once the texture of the ground was created, randomly uneven terrain was allowed to be created. Numerous tools could be used to create this such as erosion, which simulates uneven terrain via erosion caused by the movement of soil from high to low areas; hydro, which simulates uneven terrain via erosion caused by rainfall; or Noise, which raises and lowers

ground in a stochastic manner. Unreal's noise landscaping tool was applied due to, after testing all tools, producing the most sensible terrain for the given application. As previously stated, this tool randomly increases and decreases the height of ground in a certain radius stochastically, by a coefficient (tool strength) decided by the user. The larger the coefficient the more chaotic and harsher the peaks and troughs produced will be. Therefore, after trial and error testing, it was concluded a tool strength of approximately 0.01 was best suited for the creation of the terrain resulting in a completely random gradient terrain Figure 5.

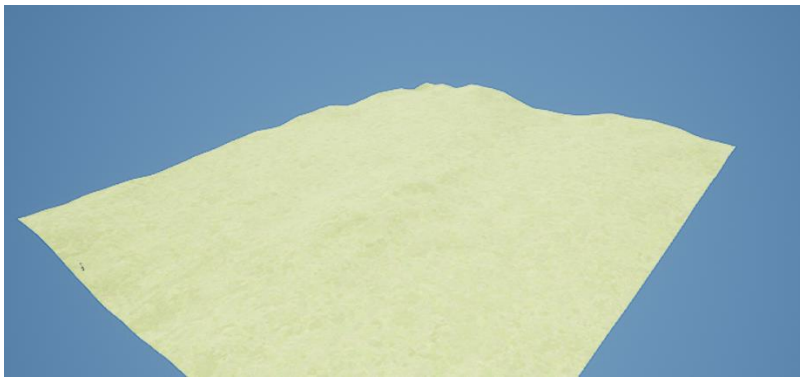

Figure 5 – Uneven Terrain Created

After the creation of the terrain the quadruped would be simulated on, static objects had to be inserted into the environment, as identified in the requirements, for the quadruped to traverse through to reach the end location. The objects were required to be appropriate in order to maintain realism and immersion during the simulation, therefore trees and rocks were chosen to represent the obstacles within the environment, as seen in Figure 6.

Figure 6 – Design of Static Obstacles Used

These objects, with the use of Unreal's foliage tool, were manually placed throughout the terrain, alternating whether a rock or tree was being placed. Upon placing a realistic amount of obstacles, making sure these obstacles were not sparse nor in abundance, the final terrain of the environment was created (Figure 7), meeting the requirement set whilst doing so.

Figure 7 – Uneven Terrain with Obstacles

As identified in the requirements, an end structure was required to be placed in the terrain for the quadruped to traverse to. This end structure was required to be an elevated platform with steps leading up to this elevated platform that the quadruped could ascend, upon this elevated platform there needed to be a table that the quadruped could place a ball on top of. To create this structure there were numerous methods, with each method being tested by creating part of the structure, then simulating and analysing the advantages and disadvantages of each option. One option was using numerous modified cubes to simulate steps; however, this method was deemed as too inefficient due the amount of time required to create and adjust the steps. Another option was to use predesigned static meshes (how an object will look during simulation) however during simulation tests due to these meshes being designed and not the actual physical object there were no collisions between the quadruped and the stairs, meaning the quadruped would pass straight through these. The final option, that was executed in the environment, was to use Unreal's built in geometry tool to place a linear staircase. This staircase allowed quick modifications to not only the size of the structure as a whole but quick modification to the steps (step length, step height etc.), whilst also containing collisions meaning the quadruped would not pass through it. Once the stairs had been finalised the same tool was used to insert a cube that would be used as the platform in the end structure, as seen in Figure 8.

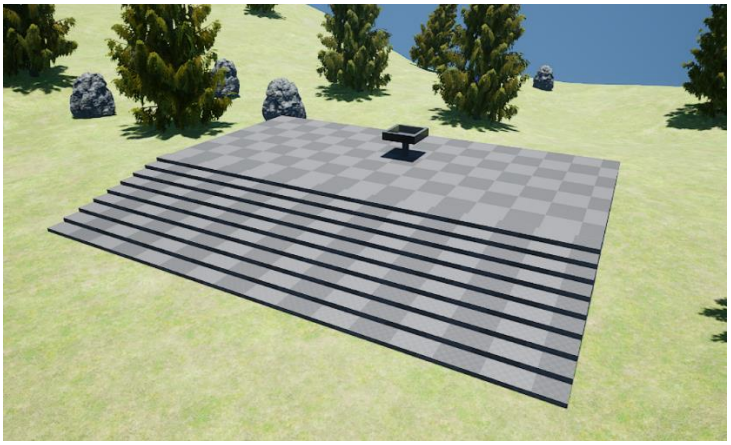

Figure 8 – End Location Structure

In addition to the structure, Figure 8 also shows the end table structure on top of the platform. Similar to the design of the structure there were numerous methods to design an end goal table, one option was to use a pre-made table asset however, after analysing the table it was concluded that it was not appropriate for this project, this being due to it not meeting the environment specifications and having an inability to be modified. The method opted in this project was to manually create the structure as these allowed modifications to the table during simulation. Furthermore, a similar modified table was used at the quadruped start position with a red sphere placed on top for the quadruped's manipulator to pick up.

Upon completing the environment as well as testing the final environment in simulations the advanced requirement of randomly placed static obstacles, rather than manually placed, was implemented. In order to complete this task each of the static objects required an extra blueprint as well as their current blueprint. This extra blueprint was required in order to place the obstacles in random positions across the terrain. This required an invisible cube to be created enclosing the terrain, which the extra blueprint will be attached to, to indicate where the obstacles could spawn; from there the behaviour of the obstacles spawning had to be defined in the blueprint.

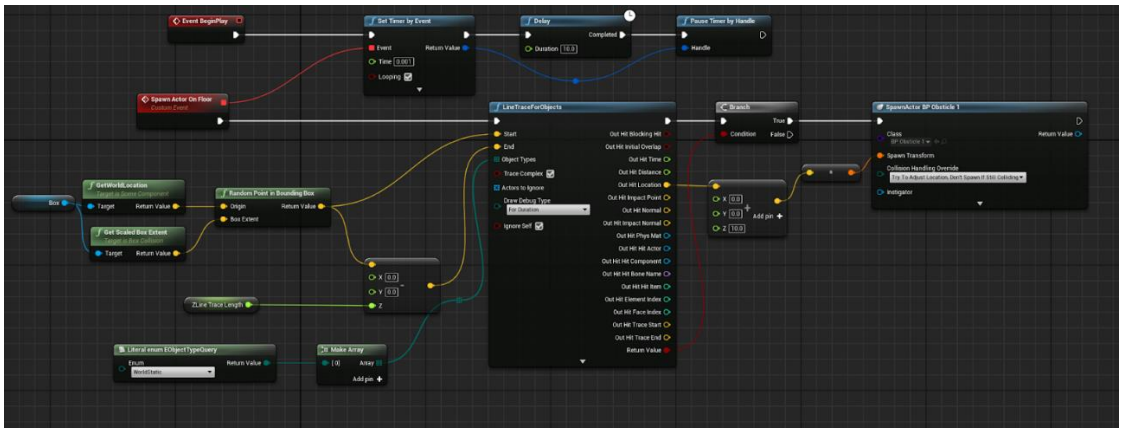

Figure 9 – Random Obstacle Spawning Generator

As seen in Figure 9, the obstacles are spawned based on a timer function set timer by Event, this function executes the attached event when the run time elapses the time specified, in this case 0.001 seconds. By ticking the 'looping' box in this function converted it into a loop function, where rather than executing the event once 0.001 seconds has elapsed, the procedure executes the event every 0.001 seconds. Using this procedure in harmony with the delay and Pause Timer by Handle a loop was created where the specified obstacle is placed every 0.001 seconds until the allotted delay is reached, in this case 10 seconds. The algorithm used on the bottom half of Figure 9 allows the obstacle to be placed onto the correct ground level; this is due to the terrain being uneven therefore the height of the ground at each point on the terrain will be different. The Spawn Actor On Floor event gathers the size of the domain (size of the invisible cube enclosing the terrain) that the obstacle could possibly spawn into, in vector form. Plugging this vector, as well as the

vector of the height of the floor, into the LineTraceForObjects function calculates the position vector the object is required to be placed at if the object is to be on the floor. The output of this function is then connected into the PlaceActor function where it is specified which obstacle will spawn (either a tree or a rock). In addition, by setting collision handling override to the correct setting prevents the obstacle being spawned and placed within the same radius as another object.

After this algorithm had been implemented, tested, and verified the same algorithm was implemented again except this time the algorithm would be used on the end location and the start location of the quadruped. Identical to the set up above an invisible cube will be placed encasing the area where the quadruped could potentially spawn, and another cube will be placed encasing where the end location could possibly spawn. However, the only difference being the 'looping' box in the set timer by event function will be unchecked, this means the desired object (start/end location of the quadruped) will be placed once after a certain amount of time has passed. Due to these objects and obstacles not being able to be placed in the same area, and the end location taking up a larger area than the obstacles, these locations had to be placed before any obstacle had been placed. Furthermore, due to the ground being uneven and the end location being a large area, there is a possibility that the stairs could be off the ground if the centre of the structure is on the peak of a hill. To combat this the centre of the object (what will be placed on the ground) was set to be the base of the stairs, meaning that the stairs will always be in contact with the floor.

Conclusion

As seen in this chapter the majority of the requirements were achieved allowing the desired environment to be created for this project. Unfortunately, due to running out of time, because of integrating numerous systems into the environment, the dynamic moving obstacles could not be implemented in time. Whilst this requirement was missed the final terrain contains enough complexity for all features of the system to be applied, whilst maintaining a high level of immersion and realism.

Furthermore, upon the conclusion of the environment, with use of Unreal's built in tools and compatibility this project was achieved to our desired level. If a different simulation engine was used this project could have been substantially more complex without producing the desired output.

3.2 KINEMATIC AND SIMULATION MODELLING (JAMIE WHITAKER - 200332345 AND LUCAS ARGYROU - 200121613)

Introduction

The quadruped and manipulator's kinematics and simulation modelling refers to creating a simplified structure that captures all of the quadrupeds' essential kinematic details, abstracting it down into a simple model. This can then be used for simulation modelling to then improve upon and modify, to achieve the desired performance, as well as to simulate and monitor the system's behaviour in different scenarios. The aim is to have Defined mathematical equations that can be used to describe and explain the structure of the quadruped and manipulator. This is the first and base step in the modelling process of the quadruped, once complete it can be passed onto the owner of the gait modelling section. The approach that was chosen was to divide these two systems (quadruped and manipulator) into independent subsystems to be merged at a later time; this strategy was chosen to facilitate validation and verification processes.

Literature review

Kinematic modelling is a key section with regards to the modelling of a quadruped and it is important to define a unified coordinate system to evaluate the interaction between neighbouring joints [15].

When creating the model, the quadruped's shape is significant. An example of this is choosing the appropriate leg length, since excessively long legs are more unstable and highly proportionate body structures are less agile and stable [16]. The quadruped's field of vision should also be taken into account [17]. If the quadruped is too low to the ground, it may be difficult for it to see the obstacles and the terrain.

The total amount of joints in the model is a significant factor for the project. Almost all of the research on quadrupeds, based on a variety of materials, found that having three joints per leg was the best way to allow stability when turning and walking across rough terrain in a straight line [15][16][18][19][20][21]. Additionally, from an evolutionary point of view, toed animals with three joints in their legs, like tigers and leopards, offer excellent benefits in terms of running speed and energy efficiency [16]. One article [19] uses the addition of a spine within the body of the quadruped adding an additional two degrees of freedom to improve stability however due to its complexity is not relevant for the project. Another article [22] uses only two joints per leg however makes up for the lack of the joint with a double acting cylinder and a wheel, which is not desirable to the project.

Many manipulators currently in industry are modelled with five degrees of freedom to mimic the movement of a human arm, as a lot of manipulators currently are made to aid the physically impaired, the elderly and workers within industry in general [23]. Manipulators with only five degrees of freedom can be flexible and

simple to assemble which are desirable traits. A study shown in article [24] describes how a manipulator with seven degrees of freedom cannot replicate the human arm movement and the overall modelling of the manipulator to be quite complex, moreover when compared to a manipulator with six DoF, the six DoF outperformed its rival as it was more consistent in multiple different scenarios. Therefore, the seven DoF manipulator is very rarely used within most industries. Paper [25] goes on to show that a six DoF manipulator has more efficiency, adaptability, and again accurate results, than a human arm (which is only five degrees of freedom). A current six DoF manipulator on a mobile platform that assists humans with a variety of tasks is the Kinova manipulator [26].

Requirements

There are a majority of functional requirements relating to this chapter including 1,2,4 and 7 from table 4 this is due to the model being essential for all gait modelling and stability analysis to occur. Furthermore, the functional requirements 6 and 8 are key to the project as they involve the manipulator and its kinematics to enable it to pick up and drop of a ball at a desired location, finally functional requirement 9 was a requirement that was initially set however due to project scope and lack of both time and documentation available was not completed.

There also included a variety of non-functional requirements relating to kinematics and simulation modelling shown in table 4. These were all used to ensure validation and to track progress throughout the project.

Main Technical Content

This chapter of the project is broken down into a systematic approach separating each section into core areas with verification and validation stages between each core area.

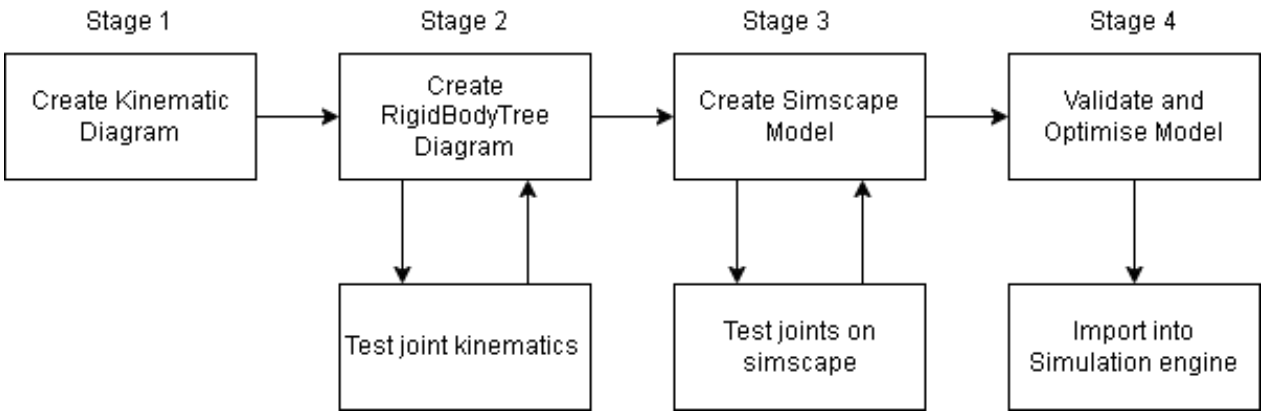

Figure 10 – Chapter Systematic Approach

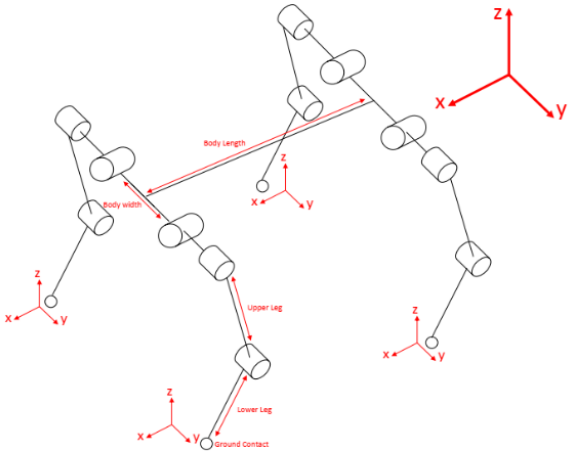

Figure 11 - kinematics diagram of quadruped

Starting with the quadruped, the development begins with the kinematics diagram shown in Figure 11, which is essential for understanding quadruped joint interactions and analysing the joint configuration and orientations to obtain DH parameters of the quadruped.

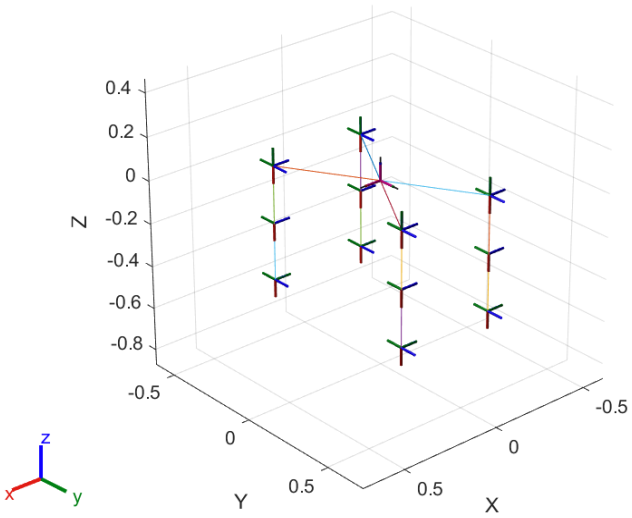

Figure 12 - RigidBodyTree diagram based on kinematics diagram

The kinematics diagram was established using data and information learnt through the literature review, the reason for choosing three joints per leg, two for sagittal movement and one for pitch is to provide the quadruped with the full mobility of currently active quadrupeds in use today. The RigidBodyTree Diagram was created in MATLAB using the Robotics Systems Toolbox, shown in Figure 12. This provides the ability to complete comprehensive robotic visualisation and analysis. This acts as a basis for all further modelling of the quadruped and how the joints interact.

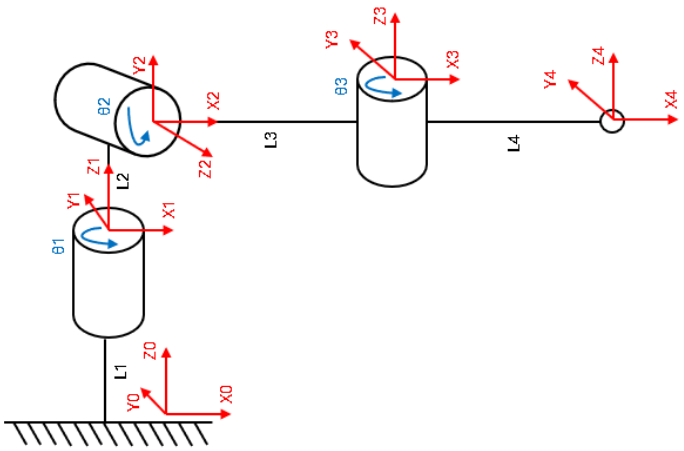

Figure 13 - A diagram of the single leg kinematics diagram

Table 4 - DH table of single leg configuration

i	a_i	α_i	d_i	θ_i
0 - 1	0	0	L1	0
1 - 2	0	$\pi/2$	L2	$\theta 1$
2 - 3	L3	$-\pi/2$	0	$\theta 2$
3 - 4	L4	0	0	$\theta 3$

A diagram of the single leg kinematics diagram was produced as seen in Figure 13, followed by the generation of a DH table based on the legs configuration that can be used to model and move the leg shown in Table 6. Due to the Nature of the Denavit-hartenberg approach the pitch joint and the first sagittal joint have swapped positions in the Leg kinematics diagram, due to these joints overlapping it does not matter the order in which they occur and for the sake of simplicity the joints have been swapped.

$$\begin{bmatrix} \cos(\theta2+\theta3)\cos(\theta1) & -\sin(\theta2+\theta3)\cos(\theta1) & \sin(\theta1) & \cos(\theta1)(L_3+L_4\cos(\theta2)). \\ \sin(\theta2+\theta3) & \cos(\theta2+\theta3) & 0 & L_2+L_4\sin(\theta2) \\ -\cos(\theta2+\theta3)\sin(\theta1) & \sin(\theta2+\theta3)\sin(\theta1) & \cos(\theta1) & L_1-L_3\sin(\theta1)-L_4\cos(\theta_2)\sin(\theta1) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 14 - homogeneous equation from base to end effector

From this DH table the homogeneous transform matrix can be acquired and tested within the MATLAB environment, found in Figure 14.

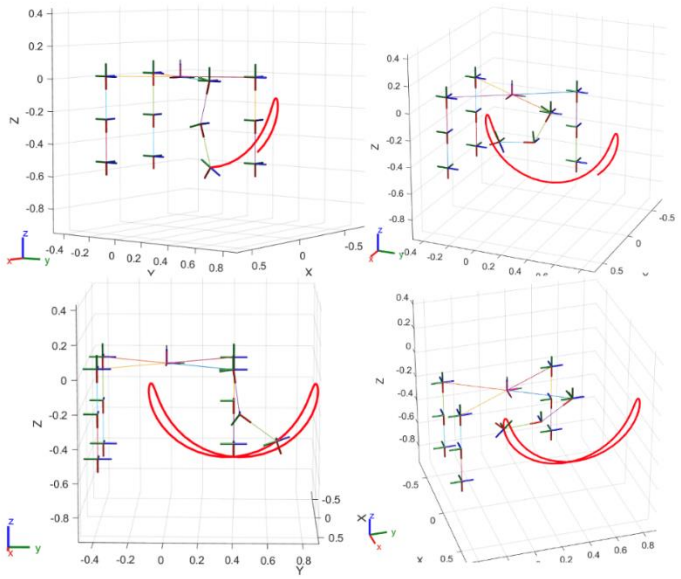

Figure 15 - visual representation of the quadrupeds foots trajectory following a sinusoidal motion

To verify the DH table and the joint configuration, a simple sine wave can be fed into the leg joints of the model, this will allow the leg to move in a sinusoidal motion in the sagittal plane. The end effector (foot in terms of the quadruped) plots the red trajectory shown in Figure 15 as it moves, allowing a visual representation of the movement path of the joint. Thus, verifying the joints configuration and structure to be correct.

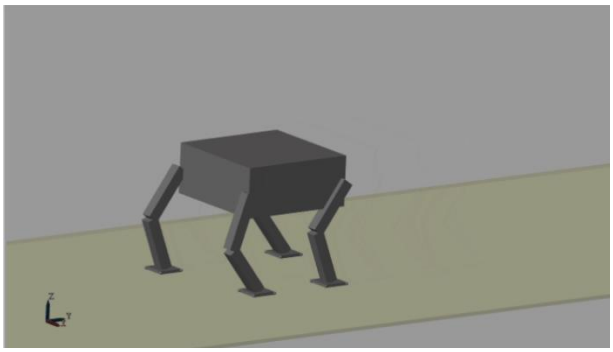

Figure 16 - Quadruped modelled in Simscape

Once the Kinematics and RigidBodyTree Diagrams of the quadruped were verified and validated, they were passed to the Gait modelling owner to further develop and begin work on gait movements mentioned later in the report. The next stage was to model the Kinematic Diagram into MATLAB's Simscape Multibody. This is a MATLAB Simulink additional tool that allows users to rapidly create models of physical systems within the Simulink environment. [27]. The reason for using Simscape Multibody was to prepare a model that would be ready to interact with the MathWorks Unreal Engine Integration Blocks Included within the Vehicle Dynamics Toolbox's block set. Additionally, this platform is perfect for verification and validation of the

kinematic modelling, to visualise whether the joints have been correctly modelled. This phase required vast research and development of Simscape and Simscape multibody, starting with a comprehensive MATLAB tutorial that showed how to produce a bipedal robot within the environment [28]. Following and understanding these steps allowed for the creation of a simple four-legged quadruped within Simscape as shown in Figure 16.

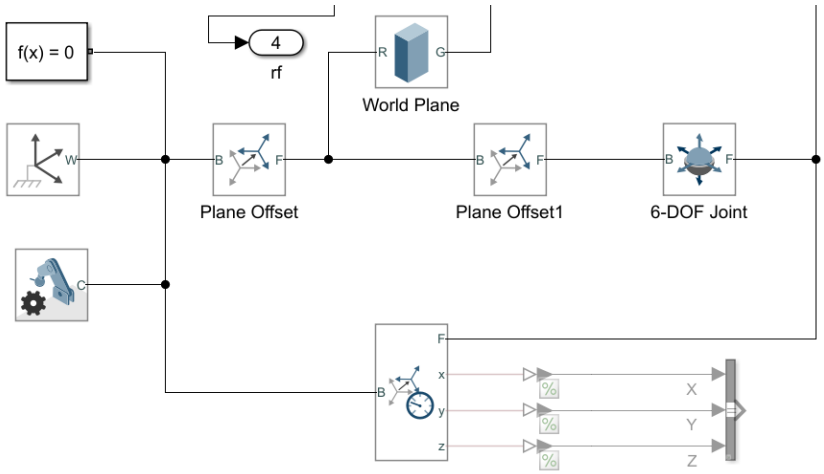

Figure 17 - Simscape of the world plane

The kinematics diagram was then implemented into Simscape. This requires the generation of the base plane with the offset of the quadruped's body shown in Figure 17. The quadruped is modelled as an object with six degrees of freedom. The reason for this choice is due to the body itself having no movement, therefore needing four legs to enable movement within all planes. With the Plane established, the legs were made within a subsystem block within Simscape to help maintain clarity within the model, as well as future debugging and evaluation.

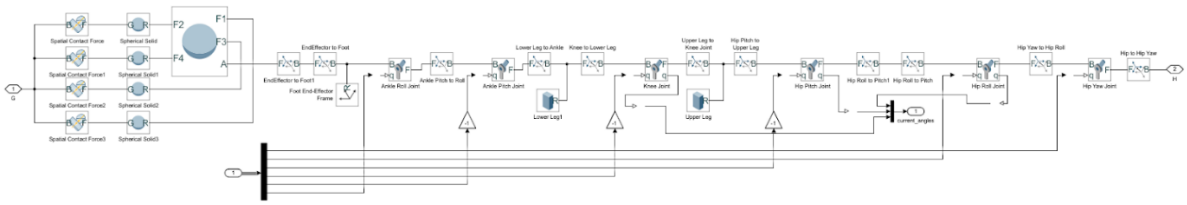

Figure 18 - Simscape leg structure design

Figure 18 shows the leg structure in Simscape which was produced using the logic provided by the Kinematics Diagram. This was done to ensure that the homogeneous transformation matrix for the base to end effector of each leg in Figure 14 will be uniform with the Simscape Multibody Model.

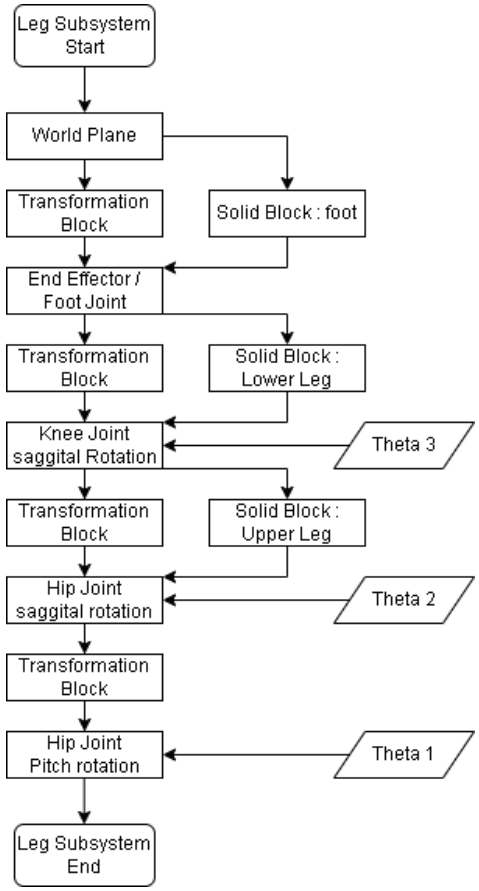

Figure 19 - flow diagram of leg structure design

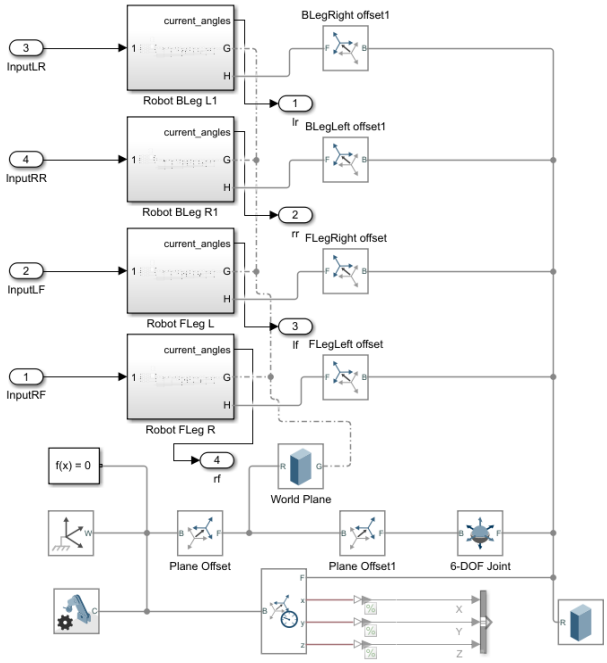

Figure 20 - full Simscape model of the quadruped

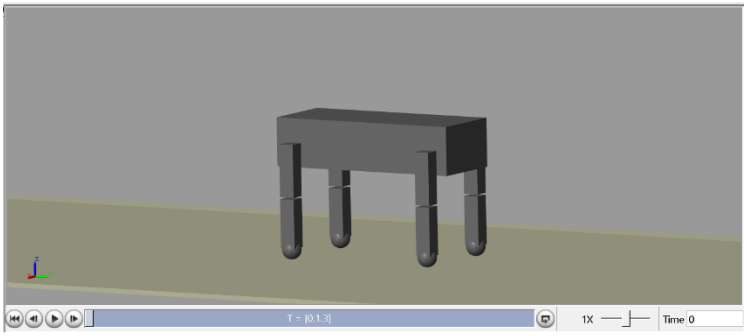

Figure 21 - Model loaded without angle data

The previous steps were integrated resulting in an individual leg with the structure of the Kinematic Diagram abstracted and shown in Figure 19.

In order to Produce a reaction force between the world plane and the quadruped model, spatial force blocks where required. Four spatial blocks were used to provide a small square like force zone. This provided a more accurate representation of the force simulation on the ball joint. Without the inclusion of four force spatial blocks, Simscape simulates the model with a single point. If this point misaligns, it can Clip and phase through the world plane simulation floor. Figure 20 shows the final Simscape model made by duplicating the four leg subsystems and naming them appropriately.

Running the simulation produces the quadruped modelled within the environment in a static Home Position Pose as defined in the rigid body tree shown in Figure 21.

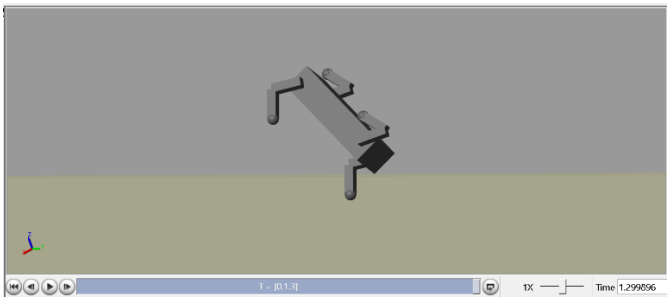

Figure 22 - first iteration of the simulator after feeding basic angles into model

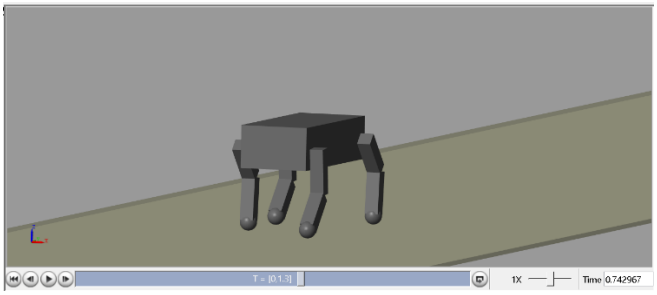

Figure 23 - Fixed quadruped model with basic angles working

The Simscape Multibody was then tested and verified, this was done to verify and validate the original modelling Justifications, such as leg degrees of freedom, link lengths, link rotations, quadruped height. This simulation was fed basic cycling Angles Provided Via MATLAB's Bipedal Walking Robot and Adapted for Use Within the Quadruped. The first iterations of testing Resulted in Some Errors with the modelled simulation running an example is shown in Figure 22. This was due to the lack of joint constraints within the joints. Figure 23 shows the fixed model which performed as expected.

Figure 24 - (a) Model A with reduced thickness and leg length testing (b) Model B wide body testing

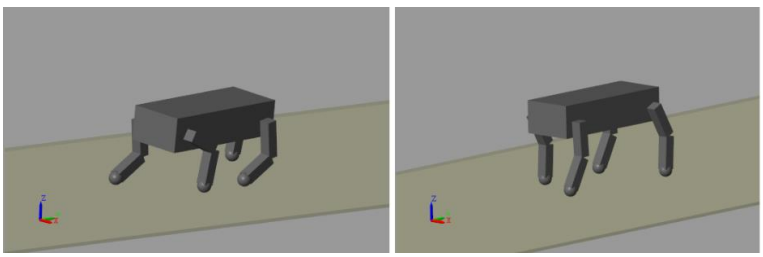

Figure 25 - The final quadruped after parameter tuning and testing

Shown in Figure 24 (a) is Model A with reducing Leg length and thickness. Shown in Figure 24 (b) is Model B with a wider Body, this led to instability when walking and reduction in balance. Using the testing and the literature review it was concluded that quadrupeds with lower centre of mass and thin bodies are the most optimised for walking and maintaining balance during movement [11] with 1 or 2 legs raised off the world plane. This led to the final design of the Simscape model shown in Figure 25 that was then Implemented in MATLAB's Unreal Engine Block sets. The Final stage for the quadruped is to integrate the MATLAB Unreal Block set within the Simscape model, with reference to [29] to replicate a similar process within our quadruped system. The aim is to transfer the Simscape model directly into the environment to have all the simulation modelling within MATLAB, using the Unreal 3D environment block which would open an instance of Unreal within MATLAB. This would allow any user with MATLAB Installed the ability to use and run the simulations without the requirement of having Unreal engine installed on the device.

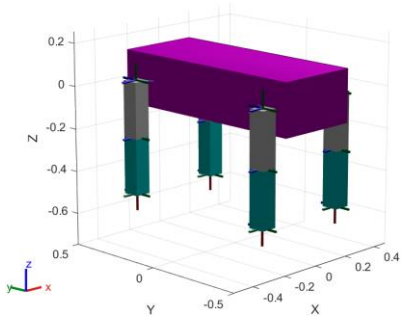

Figure 26 - URDF of quadruped

Figure 26 shows the Simscape model of the robot in a URDF file, URDF stands for Unified Robotics Description File, this format refers to the robots XML specifications of the multibody system storing both the structure and the meshes of each link. Using MATLAB, the URDF was acquired from the Simscape model, however the solids included within the model are not transferable to this format, requiring the manual mapping of the meshes.

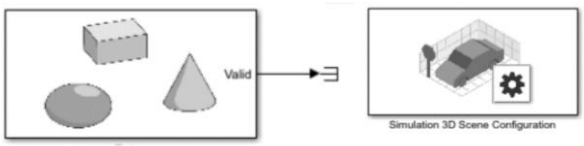

Figure 27 - 3D scene configuration and Actor Block

After the generation of a URDF, the model was then linked to the 3D actor Block within MATLAB, shown in Figure 27, this would Identify the URDF file of the quadruped as the actor within the scene allowing the simulation within the configured scene.

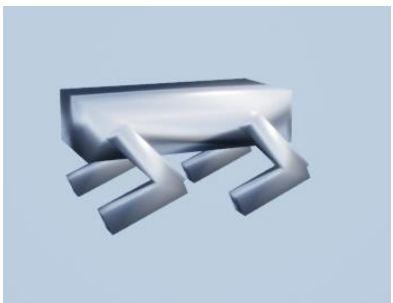

Figure 28 - Static Quadruped Model Imported via the Actor Block

It was discovered that the ability to import a multi jointed quadruped was more complex than the scope of the project, this was due to only being able to import the static shape and mesh of the quadruped shown in Figure 28. Because of the relationship between each joint and the non-grounded base. Each time a leg moves this would impact the quadruped's body, which would in turn adjust the locations of the other legs, this would result in lots of data needing to be transferred across. There was also identified to be a lack of

documentation with this toolbox to be used in this way. All results from this method were collated into a miniature report document appended in the appendix of this report. [reference appendix] This resulted in an alternative solution being used in order to achieve a similar result. With the model of the quadruped being modelled within the Unreal environment itself. This differs from the original approach by not being solely accessible via MATLAB and would now require the user to install Unreal Engine Although the physics simulation capabilities are still identical.

Moving onto the Manipulator Kinematic Modelling and Simulation Modelling, begins with the development of a Kinematic diagram in order to Produce a Denavit-Hartenberg table subsequent parameters, in order to produce the inverse and forward kinematic equations for manipulator control.

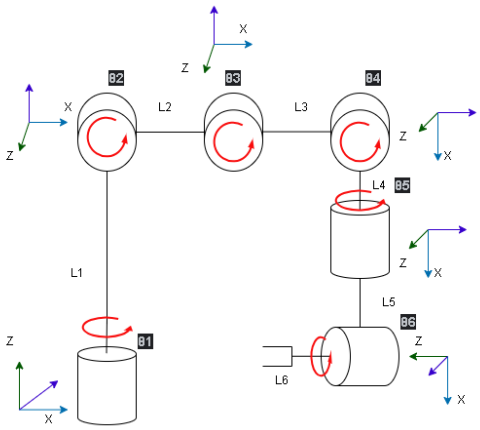

Figure 29 - Kinematics Diagram of the Manipulator

Deciding on a 6 Degrees of freedom Arm in order to provide full mobility to the manipulator in order to allow it to pick objects up whilst being on the top of the quadruped, meaning if an object was under the quadruped the manipulator must have the ability to access this workspace. With the Manipulator design decided shown in figure 29 A DH table can be created.

Table 5 - DH table of manipulator

Joint	a_i	α_i	d_i	θ_i
1	0	$\pi/2$	L1	$\theta 1$
2	L2	0	0	$\theta 2$
3	L3	0	0	$\theta 3$
4	L4	$\pi/2$	0	$\theta 4$
5	0	$-\pi/2$	L5	$\theta 5$
6	0	0	L6	$\theta 6$

From Table 7 the Transformation matrix can be derived, this will be used in both the Forward and Inverse Kinematics Equations in order to control and move the manipulator between two points. This will allow the entry of a desired end position and the arm will move towards that position adjusting all joints if required in order to achieve the position. The next step is to model the Arm within CAD in order to have the meshes that will be mapped over the arm and displayed within the Unreal Environment.

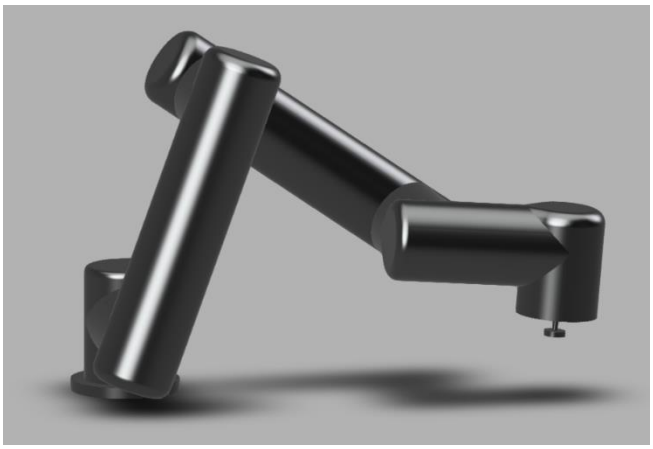

Figure 30 - The manipulator in CAD

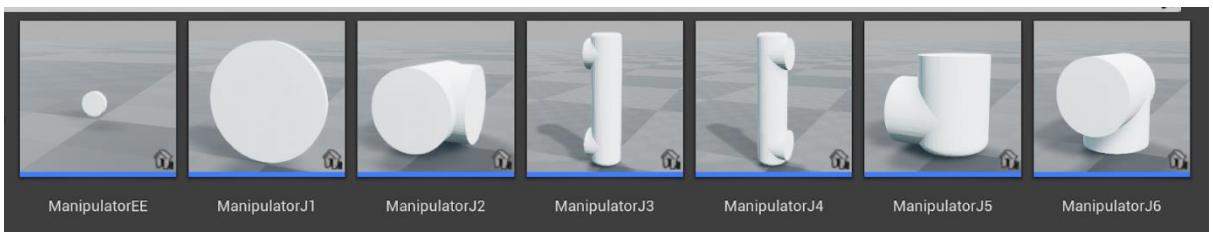

Figure 31 - the individual links of the manipulator

The CAD software SolidWorks was used to create a six degree of freedom model of the manipulator shown in Figure 30. taking inspiration from the Kinova link 6 [4]. This was done by making six individual links shown in Figure 31. Once the model was completed each individual link was exported as an .fbx file into Unreal Engine and attached together using physics constraints as stated in the integration of subsystems section.

Figure 32 - The gripper using one prismatic joint to open and close

A modelled gripper was made in replacement of the end effector. This was generated with a single prismatic joint to open and close the gripper. This concept was based on a claw gripper at an arCADe and a view of its operation can be seen in Figure 32.

The gripper was not able to be exported into the simulation due to complexity and time restraints however this is a possible future aim for the project.

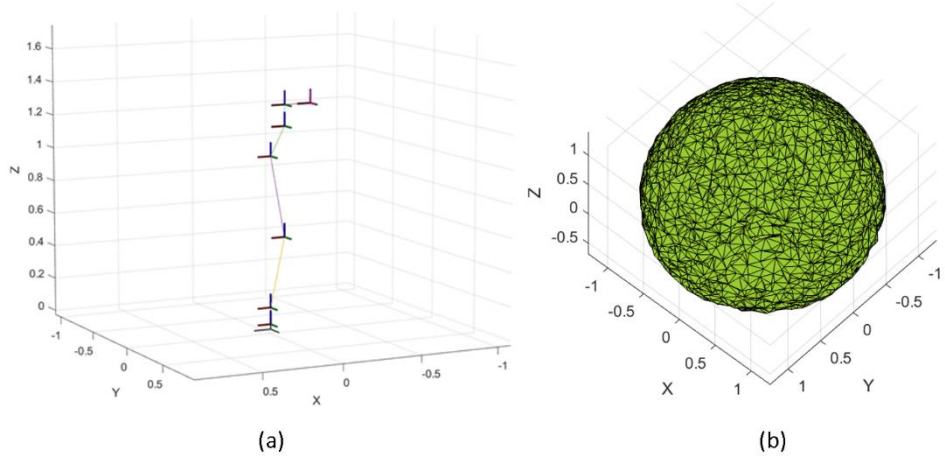

Figure 33 - (a) RigidBodyTree Diagram of manipulator (b) manipulators workspace with joint restrictions

With a finalised Idea of the Manipulator and link movability testing, a rigid body tree was created taking into account the new adjustments and parameters of manipulator the workspace was then verified with joint constraints enabled limiting the range to make sure it can reach the desired locations if necessary, all shown in figure 33.

Using the rigidBodyTree and the Homogeneous Transformation Matrix the inverse Kinematics can be generated, however opting for a much simpler and computationally less demanding method is to use the inbuilt MATLAB IK solver a part of the Robotics Systems Toolbox. This allows the code and computational requirement of the system to be drastically reduced, increasing performance. The IK solver works by entering The desired Coord location within the XYZ coordinates, entering the weights which adjusts weights associated with each joint, this contributes to the likelihood of using that joint over another joint, due to this being an iterative approach, there can be numerous configurations to meet the same target location, choosing appropriate weights can help make the arm more reliable and produce more desired and viable solutions. Choosing the ability to enter the coordinates allows the arm to act universally within the code and system as a whole, compensating and adjusting scales to accommodate for any discrepancies between MATLAB and Unreal platforms. After careful testing the weights decided where to be [1 1 1 1 1 1] in order to provide priority equally to all joints, this also produced the most desired solutions, without entering singularities or collisions between links. The next stage was to feed the IK solutions into the Rigid Tree Body.

showing the end position after the IK solver has been called. Setting the desired location as varying numbers between -2 to 2.

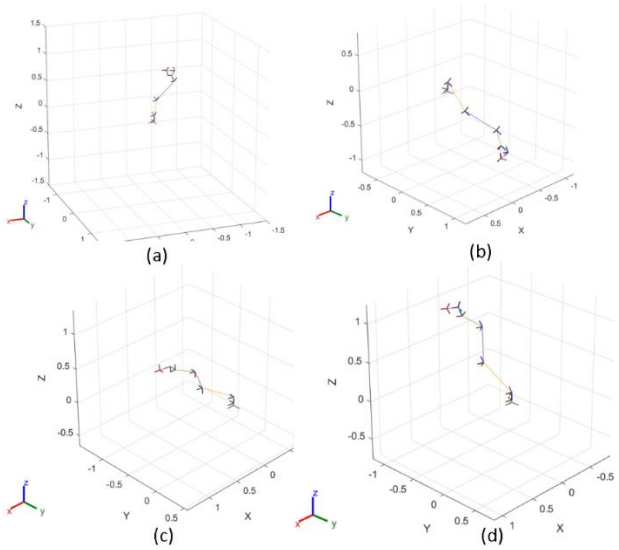

Figure 34 - Manipulator Target position configuration (a) x=1.4, y=1.8, z=0.5 (b) x=-0.4, y=1.8, z=-1, (c) x=0.4, y=-1.8, z=-0 , (d) x=2, y=-1.8, z=1.4

Shown in figure 34 is the Manipulator in varying configurations validating that it reaches these target positions entered into the IK solver. Now that the inverse kinematics has been validated, a trajectory between the desired points was required, along with a trajectory smoothing process, as the manipulator requires the reduction of jerk that may occur and cause quadruped to lose stability, therefore a trajectory smoothing was selected, originally Cubic was selected however this did not provide control or adjustability of acceleration and velocity between the points, and so quintic trajectory smoothing was used and implemented into the Inverse kinematics.

```
M = [1 t0 t0^2 t0^3 t0^4 t0^5;
     0 1 2*t0 3*t0^2 4*t0^3 5*t0^4;
     0 0 2 6*t0 12*t0^2 20*t0^3;
     1 tf tf^2 tf^3 tf^4 tf^5;
     0 1 2*tf 3*tf^2 4*tf^3 5*tf^4;
     0 0 2 6*tf 12*tf^2 20*tf^3];
```

Figure 35 - showing the Quintic trajectory smoothing matrix

Quintic Trajectory smoothing works by differentiating two different fifth order polynomials with an initial variable for acceleration velocity and time and a final value for acceleration velocity and time. These two equations are used to determine the speed and acceleration of the arm as it moves between the start and end position, setting the initial and final velocity and acceleration values as 0 to provide a smooth transition between stopping and starting when moving to the desired end effector location. Inputted in MATLAB as a matrix of the fifth order polynomial differentiated down to a third order polynomial. Figure 35

Dividing this matrix by an additional Vector B containing all Variables to calculate the coefficients for each variable and thus providing the acceleration and velocity at desired times as the arm progresses. Placing all of this into a MATLAB Function and integrating into the inverse Kinematic, the next stage was to test the inverse kinematics with the trajectory smoothing.

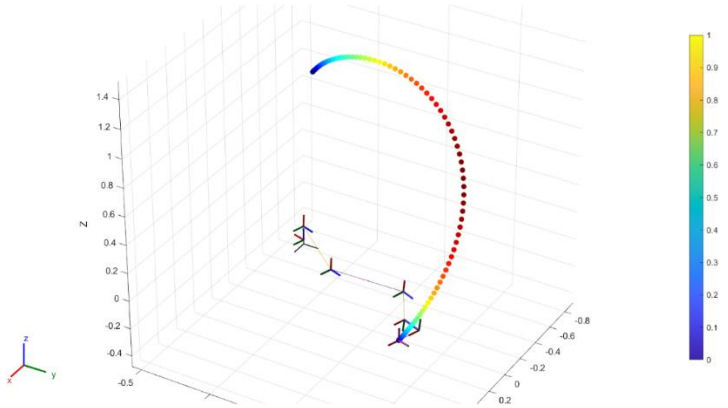

Figure 36 - Manipulator with trajectory smoothing between start and end position within 8 seconds

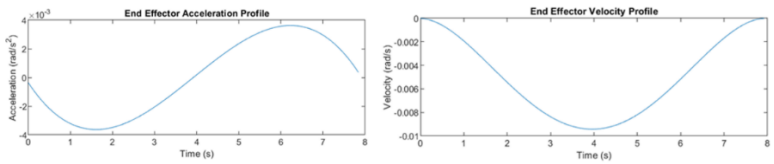

Figure 37 - Graphs of Manipulators end effector acceleration and velocity Profiles

Setting the time as 8 seconds to move between start and target position and plotting points showing the end effector's trajectory path with a colour heat map of the velocity of the end effector at that point, Blue being the slowest to red being the fastest velocity seen in Figure 36. Additional graphs were generated in order to verify the basic principle and expected results of the acceleration and velocity of the manipulator against time. Figure 37 Analysing the graphs to be the expected result and structure, as the acceleration will act in one direction then once it has reached the peak velocity and median time, a deceleration will occur in order to slow the velocity and reduce the value to 0 within 4 seconds, achieving a smooth motion with minimal jerk. As seen, the velocity is fastest in the middle of the trajectory path starting slow at the beginning and velocity decreasing as the end effector approaches the desired target position.

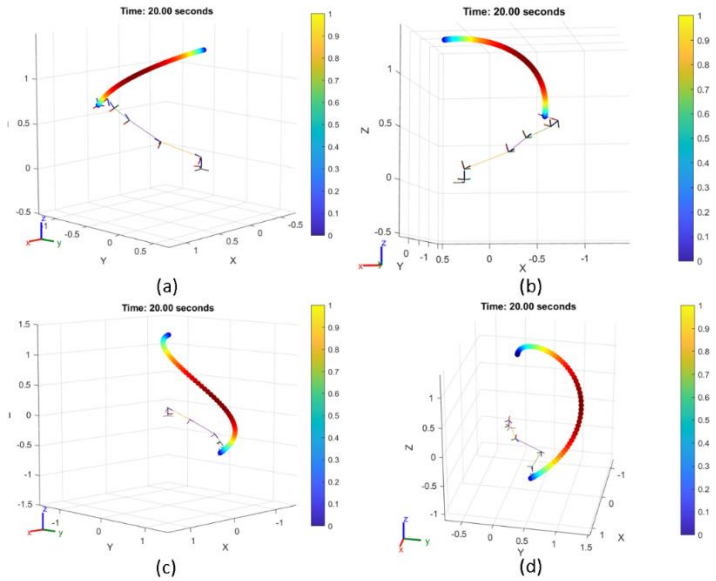

Figure 38 - Manipulator With trajectory smoothing and visual velocity trail path of end effector Target position (a) x = 2, y = -1.8, z = 1.4, (b) x = -2, y = -1.8, z = 1.4, (c) x = -2, y = 1, z = -1.4 , (d) x = 1, y = 0.6, z = - 0.4

To verify and validate the trajectory smoothing and inverse kinematics, testing was conducted and recorded in order to be logged and compared to the expected result setting the trajectory time as 20 seconds Figure 38.

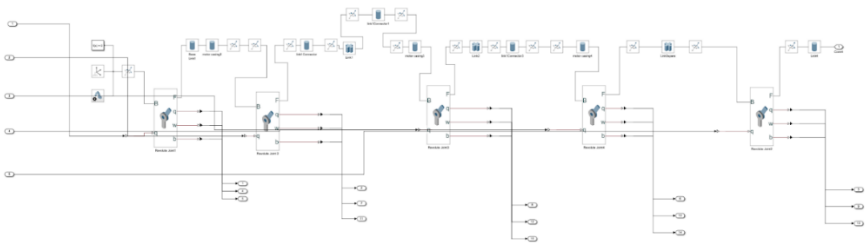

Figure 39 - Simscape of the Manipulator

The next stage is to create a Simscape model of the quadruped arm Figure 39, used to test the kinematics in motion before testing in the Unreal simulation. This was done to validate the Inverse kinematics and the joint configuration before modelling within Unreal using the CAD model and porting the data across using the 3D scene configuration block and Unreal Communication Block.

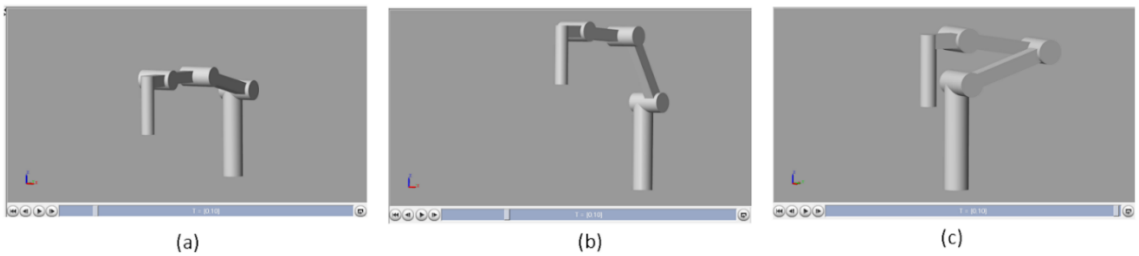

Figure 40 - Simscape of the Manipulator Simulation Outputs a to c

With the Inverse Kinematics and Simscape modelling working, and motion verified shown in Figure 40.

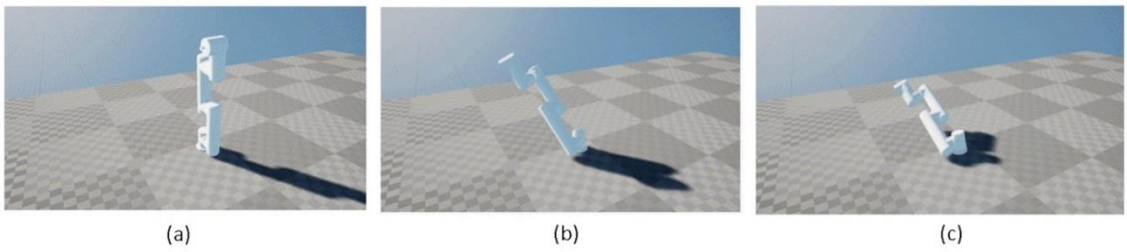

Figure 41 - Unreal simulation of the Manipulator moving to target position a to c

The final stage is to model the arm within Unreal and test the kinematics and trajectory smoothing on the modelled manipulator in the Unreal environment. Figure 41

Conclusion

To conclude, the kinematic and simulation modelling is an essential step within the project. This is because the kinematic quadruped and Manipulator modelling is a base for all other technical aspects to build on and be verified with. Throughout the process of this chapter, all technical knowledge analysed during the literature review was used to provide a comprehensive understanding of the key features which make quadrupeds and six degrees of freedom manipulators successfully, and their individual abilities to be adaptable. This allows all additional technical aspects to achieve their requirements and a pivotal area that can be passed to the gait modelling owner. The simulation modelling is a pivotal stage of this project, with the initial plan of running the simulation solely powered via Matlab without the need of the additional installation of Unreal Engine and relying on the Matlab Actor Block to propagate the environment and simulation. However, due to the scope of the project and lack of documentation, the project was adjusted and thus the requirement was not met. Due to this, an alternative solution was generated with identical simulation effectiveness by modelling within the Unreal Environment itself and transmitting the Matlab data across. A future plan for the project if additional time was provided and more comprehensive documentation on MATLAB to Unreal interaction would be the task of adapting the simulation to be solely run via MATLAB, using the inbuilt Unreal Simulator.

3.3 GAIT MODELLING (OLIVER BURGE - 200326595)

Introduction

The foundation for the movement and control of the quadruped itself are the analysis of the kinematics and the generation of inverse kinematic formulas for the system. The internal balancing and walk control of the quadruped rely on these formulas to return the required target angles for each joint in the leg, and furthermore in the manipulator arm discussed later in this report.

The quadruped legs will consist of three joints each for the purpose of 3D positioning, including a roll axis shoulder joint, a pitch axis shoulder joint and a pitch axis knee joint which connect the two links of the leg together with the quadruped body. Furthermore, this chapter will also touch on the walk cycle control utilising the calculated kinematic formulas. The controller will be capable of directing the quadruped to walk forward on uneven terrain or up a staircase and turn left and right across uneven terrain.

Literature Review

To begin producing the kinematics for this project, the analysis of existing systems can help profusely in understanding and developing the formula required. Article [35] shows the use of forward kinematics, inverse kinematics (IK), and skeleton manipulation for animating quadrupeds. By exploring these techniques, the article provides a solid foundation for understanding how kinematics plays a crucial role in creating realistic motion for quadrupeds. While it primarily focuses on animating quadrupeds for entertainment and simulation purposes, the techniques and methods discussed are highly relevant to the project by creating more realistic and adaptive locomotion strategies that respond to external forces and environmental conditions.

Paper [36] introduces a combination of dynamic and kinematic strategies to control the arm-mounted quadruped robot. By utilising joint PD controllers for arm manipulation coupled with the mobile base on the kinematic level, the study demonstrates a thoughtful approach to managing the kinematic aspects of the system. The use of a dynamic whole-body controller for handling the interaction between the arm and the quadrupedal platform is a novel approach. This methodology shows promise in enabling the robot to perform manipulation tasks efficiently while maintaining stability during locomotion.

The literature above [35] aids the project by providing insight into how the kinematics of the quadruped should be designed through information on how realistic quadrupeds manoeuvre and how a similar quadruped to the planned design operates with the manipulator shown in the paper [36].

Paper [37] provides the process from the ground up of forward and inverse kinematic development for a three-joint per leg quadruped, as well as walk cycle development through the calculated workspace of the leg end effector, which gives a clear structure to kinematic development which will influence the steps used in this project. Paper [38] also lays out the process of kinematic development for a similar quadruped design. This is further discussed in Paper [39], where a Python simulation is developed to test the kinematics on a prototype leg.

The Book "Quadruped Locomotion" [40] provides an in-depth look at dynamic control strategies for quadrupeds built to traverse uneven terrain, as well as climbing slopes and stairs, which will also be tackled in this project. The control across terrain is also discussed and handled in one Paper [41]

Finally, the kinematic process is again discussed and shown more insight into with Paper [42], where instead SolidWorks Motion is utilised to render the quadruped design and test its control.

Requirements

Considering the requirements for this aspect of the project, there has been little change to the requirements originally stated in the Interim Report. The only addition was the clarification that the quadruped could function correctly on completely flat terrain as well as uneven terrain. The verification methods for each functional requirement involve visual verification in the Unreal Engine scene.

Main technical content

In the early stages of the development stage, inverse kinematic formulae were produced for the two pitch axis joints, which initially limits the positioning to two dimensions. The formulae take in predefined values for the link lengths which allow the system to be adjustable to different specifications of a quadruped as the decision on the model size would change throughout development. The formulae take these values as well as the X and Y positioning in 2D space and output the required joint angles to achieve this. The inverse kinematic equations are seen in Figure 42.

```
q2 = acos((x1(i)^2 + y1(i)^2 - l1^2 - l2^2)/(2*l1*l2));
q1 = -( atan(y1(i)/x1(i)) + atan((l2*sin(q2))/(l1 + l2*cos(q2))) );

if x1(i) < 0
q1 = q1 + pi;
end
```

Figure 42 - Inverse Kinematic equations for the two pitch axis joints

The resulting angles can be visualised using a graphical demonstration of the quadruped legs in 2D space, utilising the MATLAB plotting tool. The demonstration is set to update each second, setting the angle joints

such that the end effector of the legs follow a devised walk cycle pattern of a semicircle. A frame from this demonstration is seen in Figure 43.

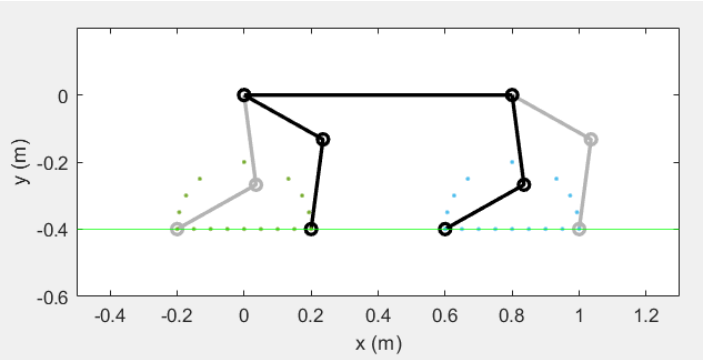

Figure 43 - Kinematic test program display

With this basic walk cycle set up, a 3D environment test could be produced in the MATLAB Multibody Simulator. As this project builds upon a previous year's work, their model can be reused for the purpose of this test. The multibody model takes in target joint angles and using PID controllers adjusts to these new positions naturally. The walk cycle controller output is fed into their respective joints, and the controller is set to cycle at a certain speed alongside the simulation time. This was tested with the quadruped in a completely flat environment in the simulator. The result was a fairly stable gait for the quadruped, but with an increased speed could travel at a fast rate without collapsing. A frame from this test can be seen in Figure 44.

Figure 44 - Kinematic Test in MATLAB Simulator

Now that the kinematics had been tested in 3D space, they could now be tested in the Unreal Environment itself. With the physics constraint set up for the model's joints (discussed in the MATLAB and Unreal Engine Communication section of the integration of subsystems chapter) the walk cycle can be tested to validate the behaviours and physics of the quadruped model in Unreal before the actual controller is adapted for full control, thus any adjustments can be made to the model.

The walk cycle code is provided with a continuous value of the Unreal simulation time, so that it can run in a loop and provide angle data to the model consistently. The first test displayed the quadruped struggling to

keep itself up whilst slightly moving its joints in the walk cycle rhythm. The method of providing joint data as well as a frame from the initial movement test are shown in Figure 45 & 46.

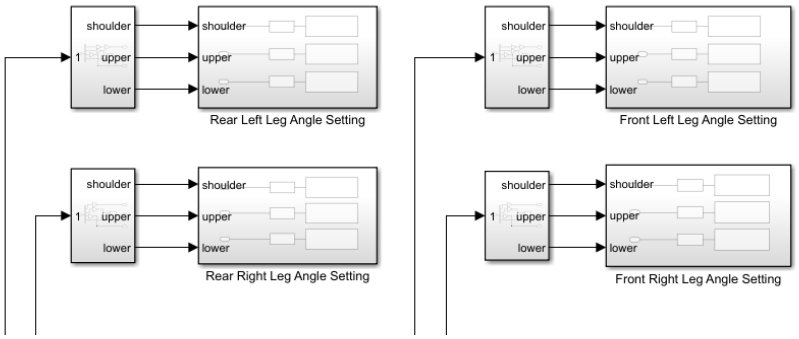

Figure 45 - Simulink sub-systems for processing and sending joint data to Unreal

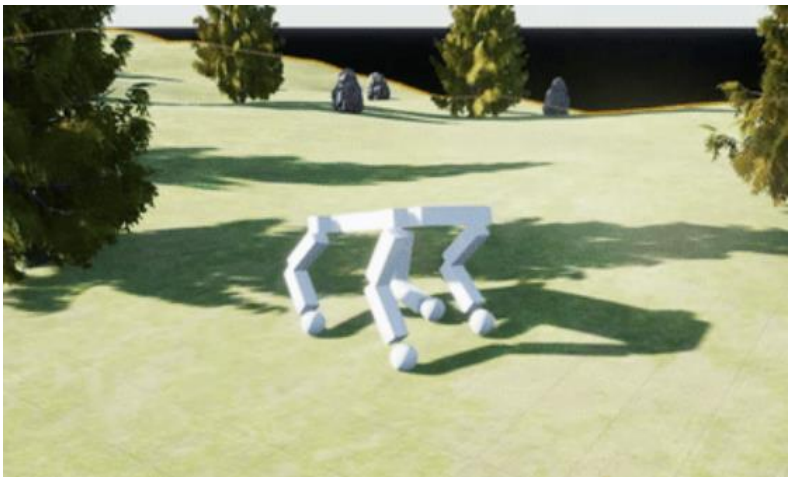

Figure 46 - Initial Kinematic Test in Unreal

This result can be explained from the minimal values for joint rigidity and angular setting strength. By increasing both of these values, the test presented the quadruped walking competently across a fairly flat ground. As shown in Figure 47.

Figure 47 - Modified Kinematic Test in Unreal

From this test, it can now be validated that the physical properties of the joints have been set up correctly and the quadruped can now be linked to a real closed-loop controller to test the walk across uneven surfaces.

From this point the idea was to produce a feedback system from scratch, involving research into quadruped control methods and determining what data to receive from the Unreal model. However, this task was deemed to be too advanced for an individual aspect, especially within the project time provided. This was due to a lack of initial experience working with systems in Unreal Engine and would be considerable to its own advanced project.

Instead, as previously mentioned, the project builds upon an existing quadruped system, so a much simpler task is to utilise the existing closed-loop controller in this preexisting quadruped system and modify it to work with an Unreal model, reacting to changes and data from the new environment. The controller previously produced makes use of the Raibert strategy to ensure stability in the quadruped's locomotion, where a dynamic trotting gait model is created from the acquisition of velocity, body rotation, and height components of the system at any given time.

The original walk cycle controller was replaced with this advanced controller, where the output was connected to the quadruped joints, and the necessary input data was provided from Unreal using the communication system (again, see section on MATLAB and Unreal Engine Communication). The data provided for the Raibert strategy to function is as follows: the velocity of all axes of the quadruped body, the rotation of the body provided as a rotational matrix, the global position of the body, and finally Boolean values for whether each foot is touching the environment ground. From these inputs the controller can determine all the required target joint angles for the quadruped to traverse in a required direction with stability. The set-up for this acquisition of data from Unreal within the Simulink system is shown in Figure 48.

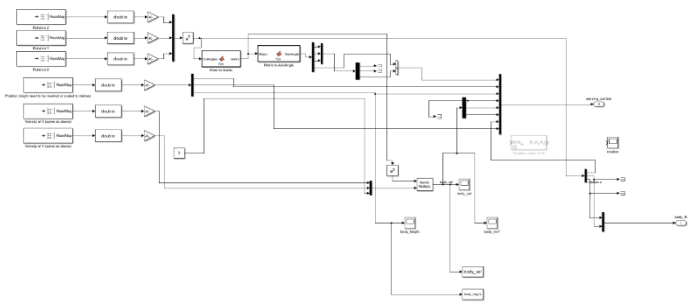

Figure 48 - Simulink set-up for receiving and processing data from Unreal

The feedback control takes in the provided existing data, as well as the angles from the previous sample, and uses this to determine the rate and positioning of each leg swing. The paper on the Raibert Strategy [30] provides a very in-depth explanation of the process. The Simulink system for the main controller code and set-up is shown in Figure 49.

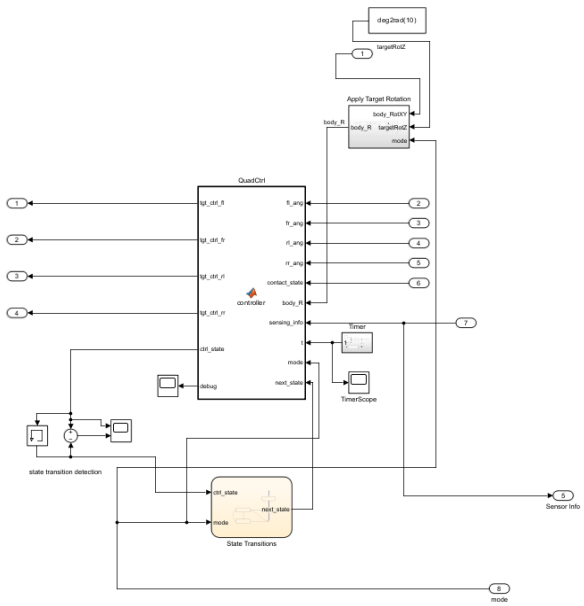

Figure 49 - Simulink system for Controller, taking in current data to provide target angles

The initial test of this new controller set for forward walking displayed good results. The quadruped kept a stable gait and produced a more natural movement than with the original walk cycle test. The turning states, both left and right, were also tested on the model and produced acceptable results. The quadruped completed turns at a slower rate in Unreal compared to within the MATLAB Simulator, however this may be due to the difference in physics engine causing friction to act differently and the quadruped sometimes visibly slips slightly when placing feet to turn. A frame from this test is shown in Figure 50.

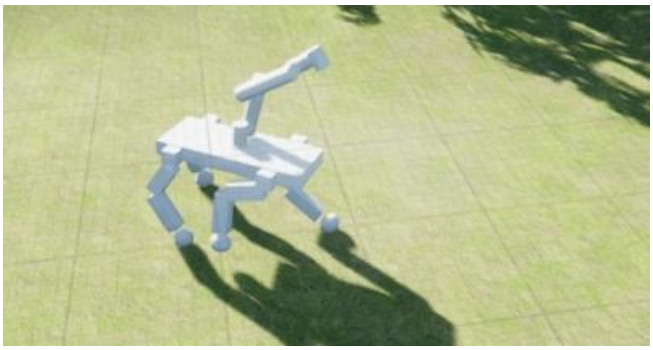

Figure 50 - Test of Controller set for Turning Left

The traversal states of forward and turning have now been successfully established for the quadruped, now making use of all three joints in each quadruped leg to ensure stable locomotion. However, an extra requirement of stair climbing is a major goal for this project, so this is a separate traversal state that needs to be worked on.

Initially, the hope was to utilise the computer vision module to detect the presence of a staircase and the length of each step, however this was not achieved in time. Thus, the current system works as follows: the

climbing state is a modified version of the forward traversal state. The length of steps need to be predefined in the code so that the kinematics can be adjusted in such a way that the quadruped swings its legs the same distance as the step length, thus avoiding the legs slipping off the step's edge. The test of this new state showed good results as long as the quadruped was aligned perpendicular to the staircase. As shown in Figure 51.

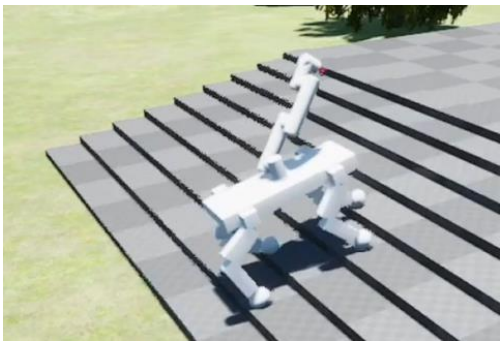

Figure 51 - Test of Controller set for Climbing Stairs

With all necessary states produced, the quadruped was now capable of traversing all aspects of the environment, as well as capable of receiving and acting on directions from a navigational system. An extra state was added where the quadruped will remain stationary, thus allowing for manipulator actions to take place.

The navigational system is composed of the combination of path finding and computer vision modules. The path finding provides the states the quadruped must move to follow the determined path via checking whether the quadruped is off the course of the path and if it needs adjustment to the correct bearing. The computer vision will override this input to set to the turning state if it determines the quadruped is approaching close to an object not accounted for by the path finding module. A basic flowchart of this system is shown in Figure 52.

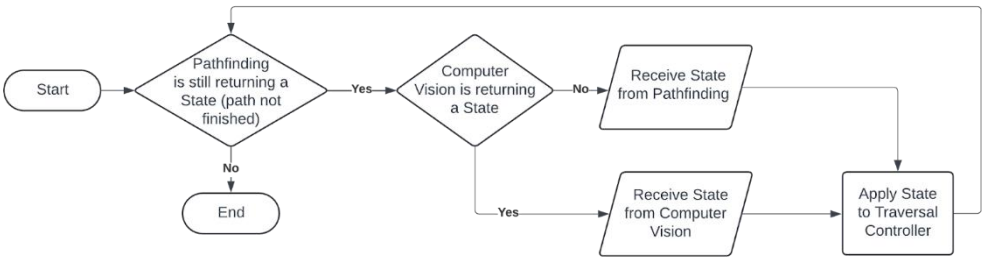

Figure 52 - Flowchart of State Controller

The development of the quadruped leg control started with the basic 2D kinematics, devising a simple walk cycle to test the behaviour and movement of the quadruped model in the early stages. From this came the modification of the feedback controller to operate within the Unreal environment, and further modified to allow for basic stair climbing. The controller was then linked to the navigational state system which makes use of other modules in the overall system.

The balancing module also acts as an extension to the control. Details on how this integrates are written in the integration of subsystems section.

Conclusion

The leg kinematics and control were a fundamental aspect of this project, as without successful quadruped traversal the other technical aspects (apart from the arm kinematics) would have no method of validation as they function as additions to the quadruped's movement. The production of this aspect was successfully achieved by the end of the project and fulfilled all requirements initially set.

3.4 COMPUTER VISION (GEORGE CRAFT - 200326469)

Introduction

When developing any autonomous robotics, it is vital that the system is able to sense and understand the environment it is placed in, in order to make the system effective, safe, and robust. Multiple sensors and methods are available with today's technology that provide autonomous robots with this information. An Example would be the usage of LiDAR. LiDAR has been receiving considerable attention, due to its ability to collect data quickly with a very high level of accuracy, however it is currently unsuitable for widespread application due to its high costs and its complexity [43][44]. Therefore, for a large range of applications, computer vision is utilised to provide live data to help make informed decisions, making it suitable for this quadruped project. Within the scope of this project, computer vision was required to guide the quadruped around objects in a random terrain and additionally guide a robot arm to pick up a red ball.

Literature Review

When deciding on how to go about implementing computer vision for this project, previous projects and literature were used to gain insight and justify any decisions made. The decision of the choice of object detector was particularly researched, due to its fundamental importance in the accuracy and complexity of the computer vision implementation.

Among the number of current methodologies, YOLO object detection emerges to be a practical choice, supported by literature that highlights its success in real world applications. Comparative studies have shown the strength of YOLO in object detection tasks. For instance, a study comparing Blob Analysis with YOLO object detection revealed that whilst blob analysis exhibited higher sensitivity, YOLO showed higher precision, an important factor in discriminating objects accurately within dynamic environments [45]. Moreover, an experiment focusing on person detection demonstrated YOLO v2's ability to classify objects with high accuracy [46], providing evidence of its reliable performance in real world scenarios. With respect to monetary considerations, YOLO v2 is favourable over other methodologies as well. In comparison to Fast R-CNN, YOLO v2 provides a cost-effective solution without compromising on accuracy [47]. While detection accuracies between YOLO and Fast R-CNN vary marginally, YOLO's inference time is 3 times faster [48] making it more attractive for real-time applications such as the one in this project.

Further reinforcement for YOLO's viability for quadrupeds comes from its usage in previous projects [49][50]. Whilst these instances use YOLO v3 and v5, the underlying principles of YOLO remain the same across all versions, emphasising its versatility and efficacy as an object detector for robotic systems. These examples

not only demonstrate YOLO's power in accurately identifying objects but also its ability to do so with quick efficiency, an important requirement for the dynamic environments inhabited by quadruped robots.

However, for the scope of this project, it's also important to evaluate the colour thresholding abilities of the object detectors, to be used for the guiding of the robot arm. Blob analysis has proven to be the most suitable for this scenario, and has been used previously to guide robot arms, using colour thresholding and shape recognition [51].

Therefore, the choice of YOLO object detection for the main object detection in the quadruped was influenced by literature that proves its success in real-world applications, precision, cost effectiveness and previous usage in similar projects. Additionally, literature helped the selection of blob analysis for the detection of red balls due to its efficiency at colour thresholding and shape recognition.

Requirements

Multiple requirements were written up to help aid the creation process of the computer vision, giving clear aims on what the object detector should detect and what to do with that information. The overall requirement was 'The quadruped shall detect unexpected objects using computer vision'. This was then broken down into 'The quadruped shall detect trees in the terrain' and 'The quadruped shall detect rocks in the terrain'. Next requirement was 'The quadruped shall choose a path depending on the objects it sees'. This was created in order to utilise the computer vision output to help guide the quadruped around any objects efficiently. The final requirement was 'The manipulator shall detect a red ball'. This was made in order to help guide the robot arm to pick up a ball.

Main technical content

After the requirements were outlined, research and testing was done to discover which methods of Computer vision would be best suited for the detection of the unexpected objects and also which method would be best for the detection of red balls. Table 8 provides a Pugh matrix showing the evaluation of the 4 possible methods. The literature review specified earlier that YOLO object detection would be a suitable detection method for detecting the unexpected objects due to its live detection capabilities, therefore testing was carried out to decide which YOLO version would be most applicable, whilst also being compatible with MATLAB. Initially testing with YOLO v4 and v3 was done, as these are the highest level of YOLO detection that are available within the computer vision toolbox in MATLAB [52]. However, when running exemplary training code for YOLO v4 or YOLO v3 using powerful PCs, the training couldn't be completed due to insufficient Video memory. Additionally, when this query was brought up to a MathWorks Research Scientist, it was suggested to use Yolo v2 as it is less complex and would save on training times, also allowing for finer

hyperparameter tuning within a reasonable time frame. Whilst YOLO v4 and v3 may have output more accurate object detection, given the time frame and machinery available, YOLO v2 was the perfect balance between accuracy and computational complexity so was therefore selected for this project.

Table 6 – Computer Vision Pugh Matrix

	Blob Analysis	Histogram of oriented gradients	Yolo v2	Yolo v3+
Live detection capabilities (3)	0	0	+3	+3
Computational power (2)	0	-1	-2	-3
Accuracy (2)	0	+1	+2	+3
Complexity(1)	0	+1	-2	-3
Colour thresholding (1)	0	-1	-1	-1
Total	0	0	+6	+5

After the choice of object detector was made and requirements were laid out, next was to begin training the detector. The first step to train a computer vision detector is to obtain data, in this case, images. Given that the simulation of this project was run completely through Unreal engine, many images from the created terrain were used to train the object detector. Initially a small dataset of 100 images was used to train the first of many object detectors. Throughout the progression of the project more and more images were used to train the detector until the final number of over 500 images was used to train the final detector. All these images were resized to 224 x 224 as this is an optimal network input size that YOLO v2 detectors can be trained on [53]. The method for labelling the images was simple. Using MATLAB's inbuilt *Image Labeller* app [54], the images were loaded in, and Region of interest (ROI) Labels for Trees and Rocks were drawn over every photo, ensuring the boxes perfectly captured just the Objects in question, as seen in Figure 53. An extra point to note is that only objects that were in the middle ground or foreground of the images were labelled, this was done to ensure the detector was only detecting objects that had an influence on the quadruped within a defined proximity. Once all photos of a set were labelled, the data was exported in Table form (where the first column is the location of the images on the hard drive and the second column is the location of all bounding boxes in matrix form) and then saved as a .mat file to be used within the training data code.

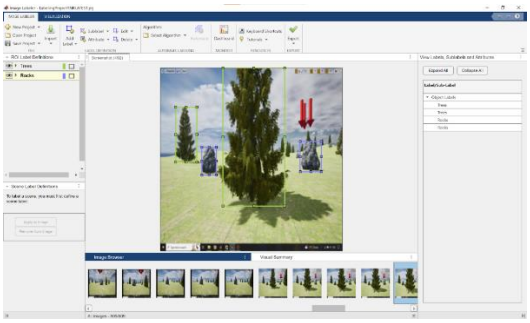

Figure 53 - Image Labeller App

Once the data had been collected and exported into the correct format, the data was then augmented. This step of training was added after multiple iterations of the object detector. It became apparent that the live feed, that the object detector was being implemented on, output a video that was significantly different than the images that the object detector was being trained on, making the detector not accurate. Therefore, data augmentation was implemented to attempt to match the training data closer to the live feed data. This was done using the *jitterColorHSV* [55] function, helping to randomly alter the training data's brightness and saturation, as shown in Figure 54.

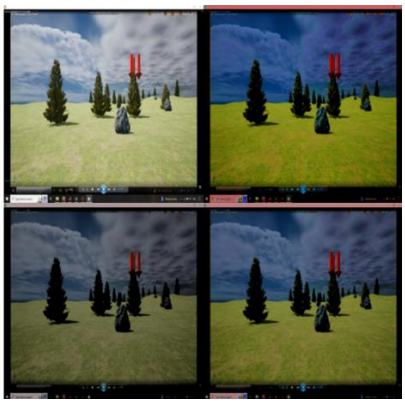

Figure 54 - demonstrates different iterations of jitterColorHSV applied to the original training data image (top left)

This greatly increased the accuracy of the finished detector, halving the RMSE of the previous detectors. Now that the training data was augmented and ready for training, the detector layers had to be defined in order to create the Convolutional Neural network (CNN) for YOLO v2. The first layer is the input layer. This layer simply accepts the size of 224 x 224 pixels with 3 RGB channels. Next were the middle layers which include Convolutional layers, Batch normalisation layers, ReLU layers and Max pooling layers. Convolution layers are fundamental for feature extraction as they apply a set of learnable filters to the input images, producing feature maps that capture different aspects of the input. In this example, 4 convolutional layers are used with an increasing number of filters. The batch normalisation layers are then used to stabilise and accelerate the training of the deep neural networks. Again, there are 4 of these and they correspond to each convolution layer, normalising the output of each. Then, there are 4 ReLU layers that introduce non-linearity to the

network, which helps to promote learning of complex patterns. Finally, are the three max pooling layers. These simply down sample the feature maps, reducing the spatial dimensions by a factor of 2. The complete layer graph is shown in Figure 55.

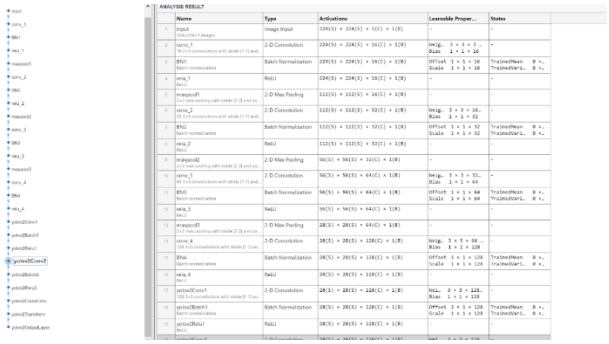

Figure 55 - Complete Layer Graph for YOLO v2

After these layers have been created, the anchor boxes then had to be estimated using the *estimateAnchorBoxes* function [56]. This function uses the bounding box data to analyse the distribution of object sizes in the training data and estimates anchor boxes that can effectively cover the variations. This is critical to ensure that the detector only detects objects that will be in the middle and foreground of the live feed, so as not to detect small background trees that could potentially cause the traversal algorithm, mentioned later, to malfunction.

Next up the training options that the YOLO v2 Detector would be trained on were defined. This had multiple parameters that significantly changed how the detector would perform, including initial learning rate, mini batch size and max epochs. Many different versions of the detector were created with different values until an optimal set of values were found. These were as shown in Table 9.

Table 7 – Training Option Parameters

Variable	Value	Reason
InitialLearnRate	0.001	Ensure stable training
MiniBatchSize	4	Reduce memory requirements and helps prevent local minima
MaxEpochs	80	Good balance between training time and model performance

Once these were defined, the object detector was trained using the *trainYOLOv2ObjectDetector* function [56]. Training of the detector took ~10 minutes each time, making it easy to adjust and change parts of the code that had a significant effect on the performance of the detector, such as the parameters defined in

Table 7. After an accurate Object detector was created, it then had to be implemented into Simulink and applied to a live feed from Unreal engine. This was done alongside the creation of a traversal algorithm as shown in the Simulink model in Figure 56.

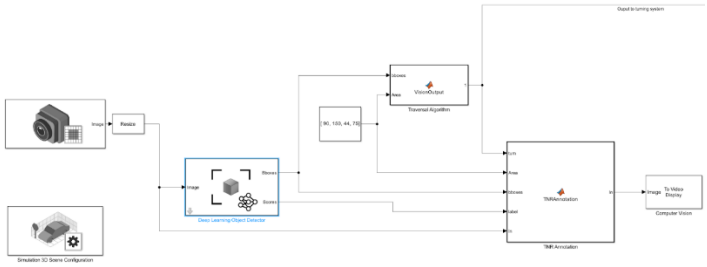

Figure 56 - Simulink Model created for Computer vision

The Simulink model uses a simulation 3D camera placed on the front of the quadruped, which is the input live feed to apply the computer vision on. This video output is then input into a 'resize' block that changes the 1280x720x3 input to a 224x224x3 output, in order to be run through the 'Deep learning object detector' block. This is the block where the YOLO v2 Object detector is present and begins to detect the objects. This block outputs the bounding box location of all (if any) current objects present in the video feed, as well as outputting the confidence scores of each of these labels. The bounding boxes output is then fed into a MATLAB function block custom made for this project called 'Traversal algorithm'. This algorithm takes all current bounding boxes and checks if any objects are within a specified 'danger zone' *OR* if the overall size of an object is larger than a specified 'safe area'. Both the 'danger zone' and 'Safe area' were decided after doing lots of testing using many different values and using judgement over when the quadruped should turn, ensuring it has enough space to turn but also ensuring it doesn't turn as soon as it sees an object in the distance. Initially, the algorithm only used the danger zone to decide if it should turn however this caused a few problems with objects on steeper gradients as they wouldn't enter the danger zone on the camera feed due to the angle of the camera, therefore the addition of the object size checker helped to ensure the detection of nearby objects, without increasing the sensitivity of the algorithm. Additionally, after the original algorithm would detect an object, it would simply turn right. Although this did work often, it was obvious that the algorithm could be made more practical, by deciding when to turn left or right. Therefore, in the final algorithm, once an object has triggered either of the *OR* statements, all the Objects on the left are summed as well as all the objects on the right. They are then compared and if there are less objects on the left, it sends a signal to the quadruped to turn left or vice versa. This algorithm is shown in the flowchart in Figure 57.

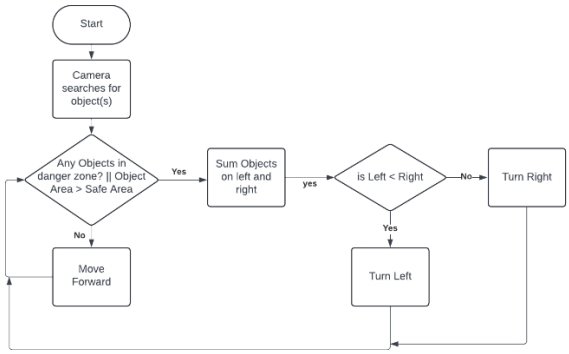

Figure 57 - Flowchart for Traversal algorithm

The final part of the Simulink model is displaying all of the information onto a single live feed display. This is done within the custom-made function called 'TNR Annotation' shown in Figure 56. Within this block, the original live feed is overlaid with the bounding box output from the Deep learning Object detector, as well as the traversal output algorithm, displaying either 'Left', 'Straight' or 'Right'. Figure 58 shows an example of this Live feed. The left photo shows when the Tree is far away enough to not trigger the traversal algorithm, however the photo on the right shows when the tree is much closer, triggering the algorithm. It decides to turn right as there is a Rock on the left, demonstrating the algorithm's ability to choose the most efficient route. The numbers shown on the bounding boxes are the confidence scores that were discussed earlier on. In order to make this detector more accurate in the future, a larger training dataset could be used, as well as more data augmentation, finer training options parameter tuning and perhaps upgrading to a newer version of YOLO detection, if higher spec machinery is available.

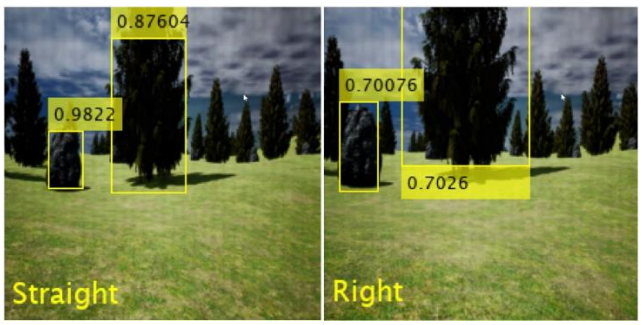

Figure 58 - Simulink Video output

Despite the use of the object detector in this project being used to detect trees and rocks, depending on the terrain the quadruped will be simulated in, the object detector could be trained to detect many different objects to traverse or track. For example, if the quadruped's goal was to be a delivery robot, it would be important for it to be able to detect any unexpected objects such as humans, cars, fences etc. to ensure it doesn't cause harm to either these objects or the robot itself. Different traversal algorithms would need to

be put in place for these complex situations, as well as additional sensors to help the quadruped understand its environment better, to aid it to make more informed decisions.

The other implementation of computer vision within this project was within the robot arm manipulator. Computer vision was utilised to track an object of choice and help feedback the location to this object. The object of choice in this project was a red ball placed among other different coloured balls, therefore colour thresholding ability of the chosen detector was the most important factor for this detector. At first, a YOLO v2 object detector was trained using 100 labelled images of red balls. However, once this was implemented, it struggled to differentiate between the different coloured balls. Therefore, Blob analysis was chosen for this object detection, due to its superior colour detection abilities and ease of implementation. Figure 59 shows the 3 simple steps of Blob analysis used. An image was imported of the scene in which the red ball needed to be detected in (far left image of Figure 59), into MATLAB's colour threshold app. This was used to filter out all colours that weren't present in the red ball (middle image of Figure 59). Next any noise was removed and finally a bounding box was placed around the detected region.

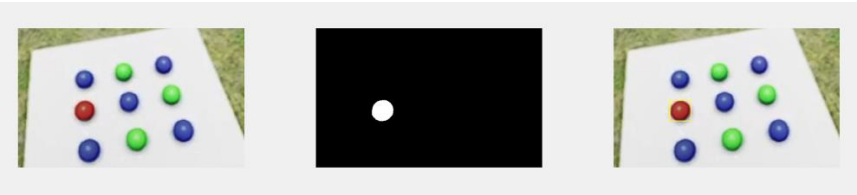

Figure 59 - Blob analysis demonstration

Although Blob analysis was the most appropriate method for object detection in this section of the project and its given requirements, a YOLO v2 object detector could also be used for the robot arm manipulator detector, in cases such as the delivery robot example mentioned earlier. Therefore, the detector used for this part of the project is entirely dependent on the objects to be detected and the context / use case of the quadruped.

Conclusion

Overall, computer vision was successfully integrated into the quadruped to help enhance its autonomy and safety. Extensive research and testing led to the decision of Object detectors, using the requirements to decide what capabilities were required for each task. The traversal algorithm was effective and demonstrates the quadruped's ability to navigate through dynamic environments. In the future, the object detector could be enhanced by expanding the training dataset, retuning the training parameters, or exploring newer versions of object detection algorithms, helping to improve the accuracy of the system. Additionally, the computer vision could be combined with an ultrasonic sensor to improve the traversal algorithm, making it even more reliable and more efficient.

3.5 PATH FINDING (JENNA HAZARD - 200326872)

Introduction

Path finding enables an autonomous robot to navigate through an environment filled with various obstacles from a start position to a desired end location along an optimal and collision-free path. This project utilises path finding with a quadruped traversing through an uneven rural terrain with the knowledge of the locations of these obstacles, as well as the start location and desired end position. This chapter will discuss the different path finding algorithms explored and explain why the one chosen was most appropriate for this project.

Literature review

The website [57] shows a step-by-step guide to using path finding within the Unreal engine. This is very useful for the project as it starts from basics finding a path between a start location and an end location by using a Navigation Mesh which is generated from the Unreal world's collision geometry and should be used to get an understanding of using the Unreal engine.

The A-Star algorithm and its variants, as discussed in the article [58], can be adapted for quadruped robots. However, it is essential to consider that quadruped robots have different motion paths compared to wheeled vehicles, which may require modifications to the path finding algorithms. The emphasis on obstacle avoidance in the article [58] is crucial for quadruped robots operating in dynamic environments and uneven terrain.

Some methods used in article [58] can be used within the project such as making a detailed analysis of the rough terrain that the quadruped will encounter by Identifying key features such as uneven terrain and steep inclines so that they can be avoided. This will be used alongside the website [57] which will provide in-depth knowledge on using Unreal to get the actor object to path plan towards a predetermined end position.

The RRT method can be used in a variety of different environments, one of them being tight spaces seen in article [205], which is useful when industry would like to use the system produced in this project in many different scenarios, in real applications. In addition, looking into article [60] sampling- based finding algorithms, such as RRT, perform better in large-scale problems, which is beneficial for the rural area that is to be simulated in this project. There are obviously negatives that come with this algorithm such as the time taken to find a feasible path may take a while and due to the search of nodes being randomised the path found may not be optimal.

Requirements

The main functional requirement is that the quadruped shall traverse to the end position using pathfinding, this in turn shall have several sub-requirements. These requirements are as follows: the algorithm shall be able to detect obstacles and in turn produce a collision-free path; the planned path should be optimal, this could include the robot navigating the shortest path or the path which is the flattest, therefore not applying a lot of pressure on the kinematics of the multi-joint robot; there shall be a complete path from beginning to end if a solution exists; the implementation of the algorithm should not be too complex.

There are extra requirements which don't directly apply to our simulation due to our environment being fixed and static, such as the algorithm should be robust, for example towards dynamic obstacles during simulation or changes to the terrain every simulation; or the algorithm should be able to adapt to different environments, such as the moon, indoors or tight spaces, however, as the simulation is intended to be used by different industries before applying to real life scenarios, then this would be advantageous to have this specific attribute.

Table 8 – Pathfinding Pugh Matrix

	PRM (Probabilistic Roadmap)	RRT (Rapidly-exploring Random Trees)	A*
Optimality of path (1)	0	+1	+2
Efficency (1)	0	+1	+2
Completeness (3)	0	+1	-1
Dynamic environments (1)	0	+2	-2
Complexity (2)	0	0	+1
Total	0	+7	0

The Pugh matrix in Table 10 compares three algorithms based on their ability to perform different requirements in MATLAB. The algorithms included are Probabilistic RoadMap (control), RRT and A*. The criteria they are being scored on is the optimality of the Path planned (the shortest path), The efficiency of

the algorithm (how fast it can calculate a path), Completeness (the ability to plot a path if one is available), how the path planned responds to dynamic environments (more aimed towards being used in real-life situations) and last but not least Complexity of implementation.

Originally looking into path finding algorithms, one of the most popular was the A* algorithm. This algorithm uses a heuristic function to explore routes between nodes, starting at one and moving through adjacent nodes until the goal node has been reached. It adds the calculated cost (the distance between the current point to the next point to travel to) to the current cost already travelled. It finds the next surrounding node with the least amount of cost and continues to do this from the start position to the end position and eventually creates the shortest path available. However, if the original cost provided is not enough to reach the end point, then the A* algorithm may never converge to a solution and may not provide a full path to the end as it will never overestimate that cost. Due to this being weighted very highly in the Pugh matrix, it was concluded that another option had to be considered. Therefore, even though A* did perform better in a few attributes, overall, for this project, and keeping in mind future applications of this product being used within industry, the most appropriate path finding algorithm to use was RRT.

Main technical content

To get started with path finding and understand the basics of how it worked, a simple PRM (Probabilistic Roadmap) algorithm was used in MATLAB. This algorithm randomly generates nodes (points on the map) and if there is no obstacle between the current node and the next node it will connect them together. This algorithm does not necessarily find the optimal path, however by increasing the number of nodes available it is possible to find a more efficient path than the previous one, as this means there are more nodes available to search through.

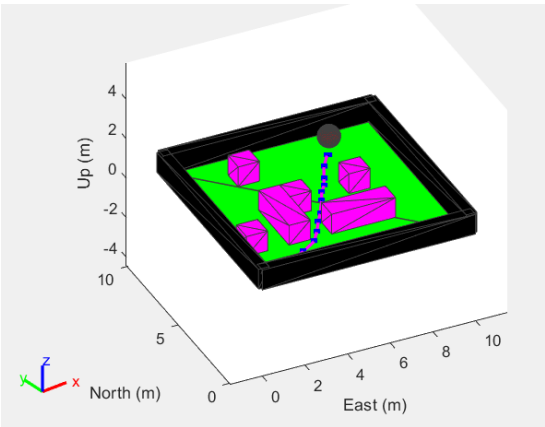

Figure 60 - 3D binary occupancy map with obstacles

Due to being at the beginning of the project there wasn't a quadruped available to use for testing and therefore, a robot was imported from MATLAB workspace. Similarly, there was not an available environment to work off of, however, in MATLAB it is generally quite simple to create a small map to test out this algorithm.

This 3D map was created as a binary occupancy grid, which represents an occupied workspace (meaning there is an obstacle in this grid) as the value true and unoccupied workspaces represented as false. As seen in Figure 60 the obstacles are represented as the pink shapes and the desired destination is the red sphere (for the purpose of testing the algorithm, the sphere was represented as an unoccupied workspace allowing the robot to actually reach the final grid the sphere was placed in). The planned path is represented as the pink and blue line, this could be slightly different each time, due to the probabilistic nature of the random sampling and number of samples taken but overall, the end location is reached each time if there is an available path.

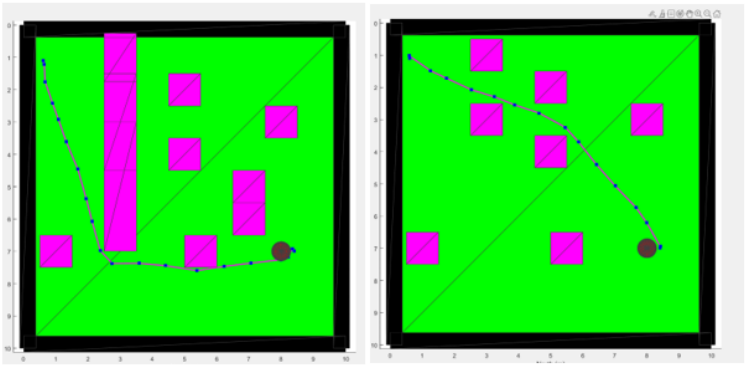

Figure 61 - Birds eye view of the 3D map with different obstacle arrangement

To visualise how the path finding algorithm works, the robot initially started in the top left corner of the map and the beacon in the bottom right corner of the map with some obstacles in between the two points. Multiple simulations were run with the same start and end location, however, the obstacle in between had a different arrangement shown in Figure 61 and therefore the planned path was rerouted to avoid the obstacles each time as the previous path had been blocked.

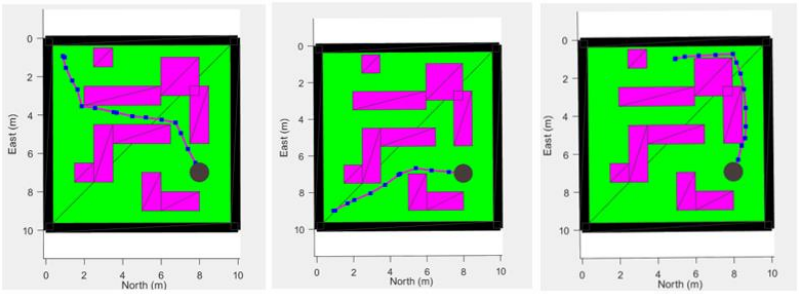

Figure 62 - Birds eye view of 3D map with different robot start positions

After observing the path finding was working, the obstacles and the sphere remained in the same place each time, however, this time the robot would randomly spawn at different locations each time a simulation was run. This was to represent how in real scenarios the robot may not start in the same place each time but

would still have to reach the same endpoint, for example if the robot was to be used in search and rescue it could collect people from different places and lead them all back to the same safe location. This simulation involved producing a matrix to generate a random x and y value to represent the start location for the robot, this was of course not a coordinate that was occupied by an obstacle as the path would have been invalid. Successfully, the simulations were run and each time they produced a path of the robot navigating its way to the sphere, shown in Figure 62.

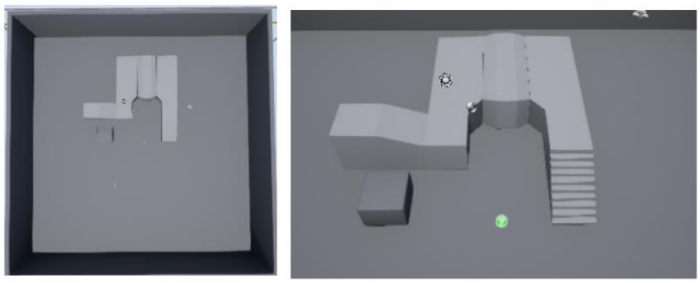

Figure 63 - pre-generated multilevel map in Unreal Engine

To start path finding in Unreal engine, there were some pre-made maps available and a 'third person character blueprint', which in this simulation was modelled as a human but for the purpose of testing was there to represent the quadruped actor as there wasn't one currently available. The map included a flat space for the actor to roam around and a set of stairs available shown in Figure 62. The availability of the stairs assisted in showing that the path finding does not only work on flat levels of the environment, such as the x and y coordinates as previously seen in the PRM algorithm, but can also work on multiple levels, which is needed for later testing of the uneven terrain for the project.

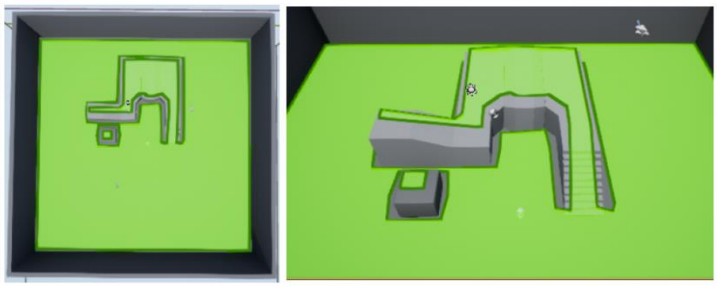

Figure 64 - NavMeshBoundsVolume actor in Unreal Engine

So, using this map and the character, first a NavMeshBounsVolume actor shown in Figure 64 has to be placed into the environment, this represents the accessible space for the character to navigate through and therefore has to be scaled and positioned appropriately to the areas that require navigation in.

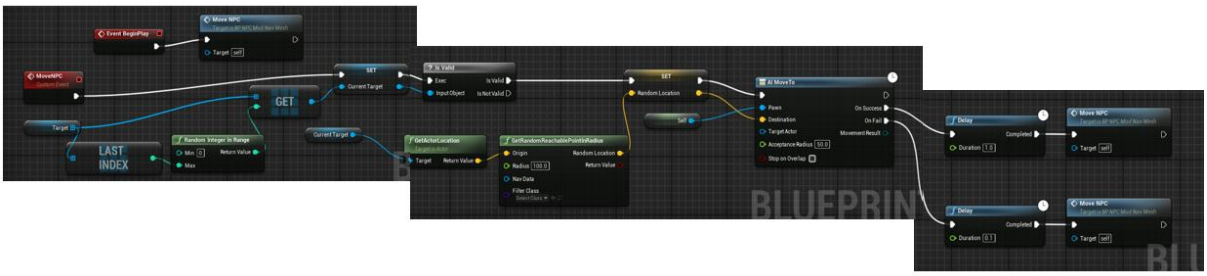

Figure 65 - Blueprint for randomised movement in Unreal Engine

After this, the actor is dragged and placed into the scene and its actions are edited in the blueprint section. Figure 65 is the initial code used to test the movement of the character and this allows the actor to roam around the map randomly, which wasn't the goal of path finding and due to there being no objects within the map, it was hard to know if the path finding algorithm actually worked, therefore a new map and code was required.

Figure 66 - Birds eye view of new map with obstacles and target points

The new map, shown in Figure 66 has different shapes and sizes of obstacles, shown in white, and stairs with a platform, shown in blue, with the target points selected for the character to navigate to, shown as pinks spheres. This was very much improved from the first attempt of path finding in Unreal Engine, as a certain end point was chosen using physical target actors on the spheres and the character navigated towards it as well as avoiding all obstacles in between the different locations.

The final step was to achieve path finding in the uneven terrain that had been made for this project. This seemed possible based on the previous achievement, however, when simulated the actor managed to reach the end goal, however, the trees and rocks weren't being avoided, but instead it was possible to pass through them. This was due to there being no box collision around the objects and therefore the actor did not recognise the obstacles. To fix this issue, it would have required editing the whole environment again, which would have been time consuming, however, by just placing a few blocks in place inside the trees and rocks the basics of the problem was resolved.

At first this method of path finding could have been an option due to how quick it was to implement, however due to the low level of coding required for this method; the environment needed adaptation again; and there wasn't a visible line of the planned path, it was decided against. This method was still beneficial though, as it could still be utilised for testing purposes of the quadruped model traversing across uneven terrain along some sort of path.

Moving back to path finding algorithms within MATLAB, due to Table 10 the next approach was to try the RRT method. The rapidly exploring random tree algorithm (RRT) works by randomly sampling points in the workspace provided and when one is found it will continue to do this from start to end connecting the points until a full path is found, even though it may not be the optimal path.

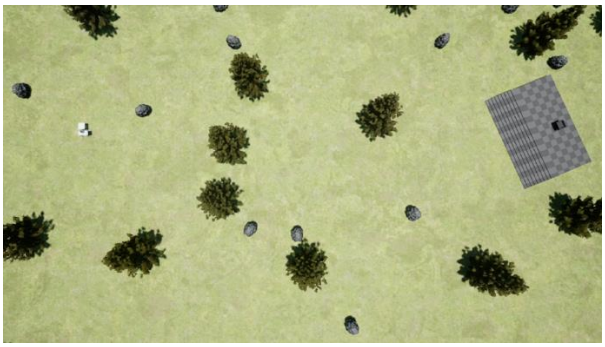

Figure 67 - Birds eye view of rural terrain with obstacles

In order to implement this algorithm a few steps had to be done first. Now that the terrain had been fully generated the first step was to obtain a high resolution, birds eye view image of this terrain shown in figure 67, which shows the placement of rocks, trees, and stairs, which in this case was the end position that the quadruped would end up at. This involved opening Unreal Engine through the scene configuration for 3D simulation environment block in Simulink and placing a camera actor in Unreal Engine. For real world applications, this could be an image taken by a UAV or a satellite image.

Figure 68 - Conversion of image into a costmap

This image is then loaded into MATLAB using the function imread() and first was converted into a gray image, using the functions im2gray(), then a Black and white image, and finally into a costmap by the function vehicleCostamp(), with all images displayed in Figure 68 respectively. The Red parts of the cost map are known as inflated areas, which displays the parts of the terrain which are occupied, so here the areas in red are the rocks, trees and again the stairs, which you can see are clearly distinguishable from the rest of the terrain. The pixels in the costmap are given a number (or a cost) based on how likely the area is a free space in the range [0-1], 0 being the white spaces and 1 being the dark red areas.

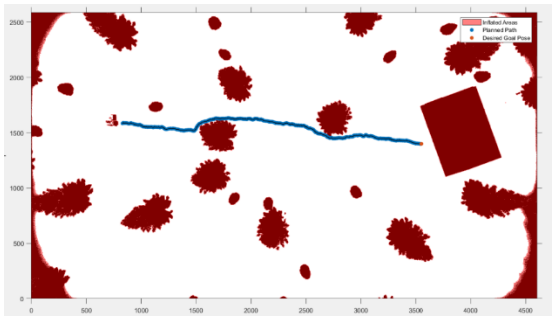

Figure 69 - Planned path on costmap

This cost map was then used to plan the path using RRT. Start by choosing a start and end position of where the manipulator will travel, in Figure 69 the start position was chosen as [829,1585,0] and the end position [3550,1400,0], with the numbers in the row vector representing the x coordinate, the y coordinate and the degree of rotation of the quadruped with respect to the x axis. The path planner works by searching many different nodes within the map and uses the numbers in the cost map (the range of [0-1]) to determine the free spaces. Once a clear path has been found from start to end with no collisions, the path is plotted on the map for us to visualise.

Conclusion

Many of the requirements stated in this chapter were achieved and eventually a successfully planned path was produced after attempting different methods to fully develop an understanding of path finding in different forms. Due to requiring a complete path from start to end, the RRT algorithm had to be used, however it did not produce the optimal path, in the sense of shortest or flattest. This could be changed if there was more time available and perhaps use RRT* algorithm or a combination of algorithms for optimality and completeness.

If this project was to actually be used by industry members, then the extra requirements could have been explored more. However overall, the algorithm wasn't too complex to implement and works well to achieve the task provided.

3.6 STABILITY ANALYSIS (HARI NANDA - 200121196)

Introduction

A principle advantage of quadruped robots is their ability to emulate the stability and agility of four-legged animals, allowing them to operate in dynamic environments unsuitable for wheeled or tracked robots. A critical component in ensuring this is possible is stability analysis, the focus of this chapter. Stability analysis aims to ensure the stability of the quadruped is always maintained, even in varying complex environments. Without stability analysis, the overall functionality, agility and ability of the quadruped to operate in real-world applications (and simulations) would be greatly diminished.

Literature Review

Fundamentally, literature [61] provides an introduction to the process of stability analysis that begins with inputs that monitor the quadruped. These inputs are then processed using a mathematical model, to simulate the quadruped , literature [62] provides a comprehensive framework for this process and provides useful information on the associated theory. These predictions can then be used by control algorithms to respond to changes in the inputs to maintain balance, literature [63] provides a case study showing the successful use of control to balance a quadruped, although the lack of real world validation takes away from its reliability. Another use of the model is by engineers seeking to gain insight into how changes to the design, walk cycles and control algorithms affect stability, guiding development literature [64] details how models can be used to analyse walking gaits of four legged animals, similar to quadrupeds. The creation of a mathematical model for stability analysis in quadruped robots presents several challenges, due to the complexity of quadruped dynamics, real-time constraints and the fact that there are a variety of methods available, with different levels of abstraction, using varying inputs discussed in literature [65]. Different evaluation of dynamic stability can be done with many different approaches, such as the static stability criterion [66], force-angle stability [67], and the zero moment point method [68], these literatures give insight into the diversity of strategies to assess the complex interactions of robotic motion and environment. Each method offers distinct advantages and limitations, influenced by the intended use, mechanical design of the robot and the specific need of the application. This process of method analysis is further complicated by the simulation nature of this project, which presents challenges to achieving the goal of maintaining the quadrupeds balance over rough terrain.

Requirements

The functional requirements that the quadruped shall move forward and turn on uneven terrain was set. This had to be achieved to not only enable basic operations to be completed but also to enable other areas

of development to be implemented. The method or approach used to achieve this was not mandated. However, the program produced had to meet the non-functional requirements of running in real time, so it would be useful in a real-world environment. It also had to be accurate the vast majority of the time and provide explainability of the stability analysis, to enable it to be utilised to guide development and control. Finally, only realistically available inputs could be used, allowing for real-world usage.

Main technical content

Extensive research was carried out to determine the most effective approach to model the stability of a quadruped. This research suggested that the optimal strategy would be to first develop a static model, which simplifies understanding the fundamental balance principles without the complexities of motion. From there, moving to a dynamic model allows for the integration of movement and external perturbations. Following this, a method for dynamic analysis will be identified, with the subsequent development and testing of the dynamic model. Finally, this model will be refined and utilised to prevent falls.

In static stability analysis for quadrupeds, various methodologies can be employed based solely on static data like the positions of feet and joints, the centre of mass (COM) of individual links, the base plate, and the associated masses. One approach involves geometric analysis to evaluate how limb arrangements affect stability, focusing on limb length and orientation relative to the body. Another method calculates the COM for different sections of the quadruped to understand how weight is distributed relative to the base of support. However, the most robust method involves computing the overall COM of the quadruped and confirming its placement within the support polygon, defined by the feet's contact points with the ground. This technique provides a comprehensive view of stability by ensuring the entire body's mass is adequately supported by its stance, this method was chosen due to its simplicity and effectiveness.

The static stability model takes an input of the feet positions, joint angles, link lengths, link weights, body weight and which legs are on the ground and outputs whether the centre of mass of the quadruped is inside the polygon of support, indicating stability. The first step to achieve this is to find the 3D coordinates of each point for each leg. The nested function “calculateAdjustedEndpoints” does this by calculating the points in a 2D space (ignoring the shoulder angle) and then rotating all points by the shoulder angle. This is repeated for each leg to produce a 12x3 matrix of points.

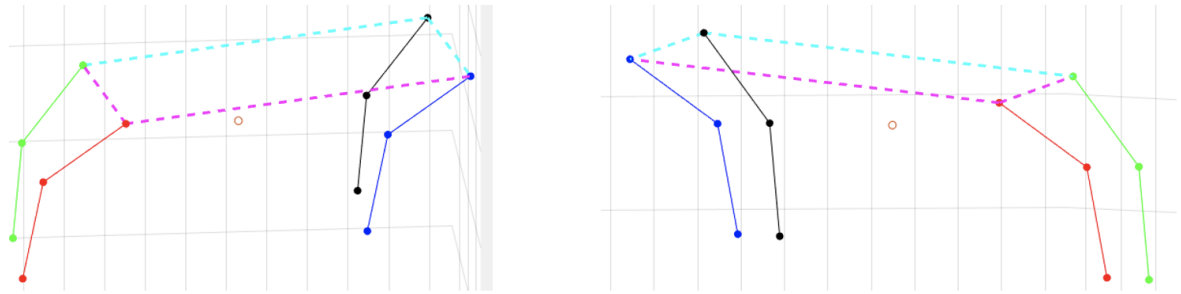

Figure 70 - Output visualised, verifies the points are correctly generated

Then, the next step is to calculate the centre of mass of the entire quadruped, the “calculateMidpoints” nested function is an intermediary step to this. This function finds the midpoint between each point in a leg chain, for example for leg 1 with points 1, 2 & 3. Two midpoints would be found, between point 1 and 2 and between points 2 and 3. Using data from the input matrix which specifies the link weight per metre the weight is calculated. This allows us to find the centre of mass for each link and its mass. There should be 8 points for 12 inputs, however a 9x4 matrix is produced. This is because row 9 contains the centre of mass of the “base” plate and its mass. The final step to finding the overall centre of mass of the quadruped is combining these 9 points and the masses into a single centre of mass, this is calculated using a weighted average inside the “calculateCenterOfMass” nested function. This point is plotted along with the position points by slightly modifying the code used to visualise the “calculateAdjustedEndpoints” function. The results visually confirm the centre of mass is in the correct position. The “isPointInTriangle” nested function is the final stage of assessing whether the quadrupeds stance is stable. It takes three foot positions (which foot is not touching the ground is specified from the input matrix) and identifies the corresponding points (ignoring the Z position). Then use the MATLAB inbuilt function “inpolygon” to determine if the centre of mass is inside the polygon of support. An output of “YES” or “NO” is produced. The results can be verified by plotting the plane from these three points and plotting the centre of mass with a vertical line going down. If the vertical line is inside the triangle created by the three points on the plane it means the centre of mass is inside the polygon of support. Expected results were obtained, validating the model.

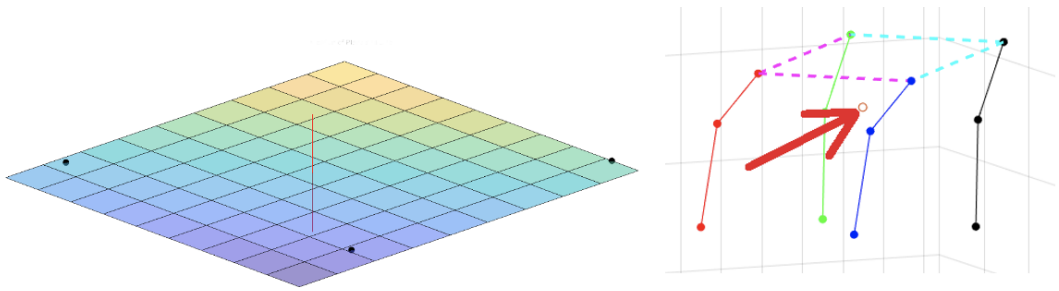

Figure 71 – (a) Output plot used for validation (left) (b) Plotted points, arrow pointing to COM (right)

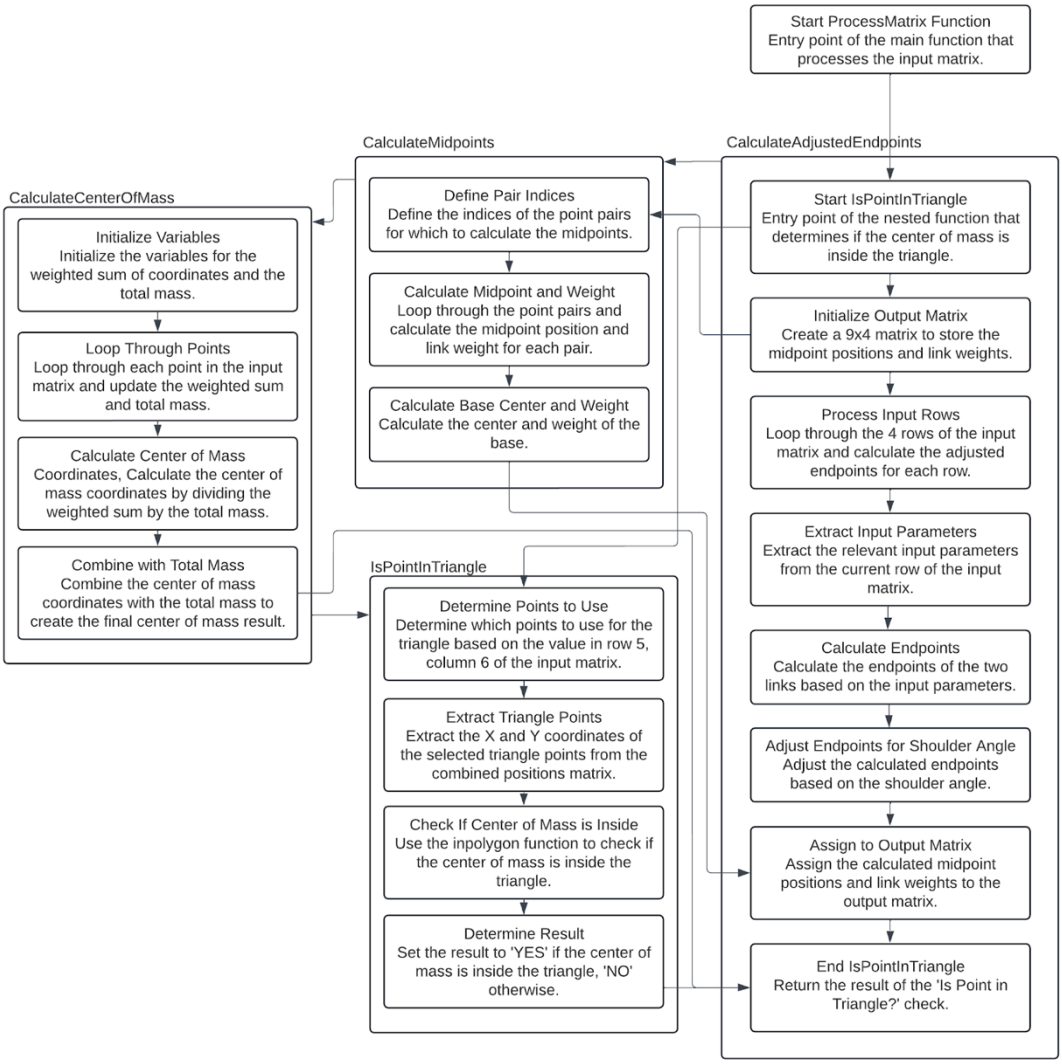

Figure 72 - FLOW DIAGRAM FOR static STABILITY analysis

The development of a dynamic model for the quadruped builds upon the static model, necessitating a range of precise new inputs. These inputs include the inertial properties of links, dynamic joint properties such as velocity, acceleration, and torque, alongside ground reaction forces and environmental data. These parameters are crucial as they enable the model to realistically simulate how the quadruped will react to physical forces. However, the data from simulations in Unreal Engine contains significant inaccuracies that arise due to the engine's limitations, sometimes parameters like velocities, acceleration and torques are infinite or indeterminate. Moreover, several required inputs and environmental factors such as friction coefficients and ground reaction forces are not captured, a very significant problem.

To address the problems caused by unknown or erroneous dynamic parameters which are required to produce a remotely accurate dynamic model, they are estimated. Two methods capable of estimating were compared. The finite difference method is particularly suited for scenarios where quick estimations are required with minimal computational overhead. Matlab, with its powerful array handling and built-in

functions, streamlines this process. Using diff to calculate the difference between successive data points and gradient to find derivatives directly, this provides a straightforward path to estimating velocities and accelerations. Since simulation data can be noisy, employing Matlab's smooth data function can help mitigate noise effects before differentiation, enhancing the reliability of the results. This method's simplicity makes it very appealing; however, its accuracy heavily depends on the quality of the data and the rate of change. Rapid dynamics or abrupt changes typical in Unreal simulations might lead to derivative estimates that are incorrect due to numerical instability and error propagation. For more precise and robust estimations, the geometric Jacobian method is preferable. This approach uses the kinematic relationships of the robot's structure, modelled via the Jacobian matrix, to relate joint velocities to the end effector velocities. This method yields joint velocities and accelerations from known end effector velocities. This method is less susceptible to noise and provides more accurate estimations by taking the physical constraints and relationships into account. However, it is unsuitable as the end effector velocities are not known. As a result, the finite difference method was chosen and utilised. The required estimated data did not appear to be realistic or feasible. To ensure development time was utilised efficiently, only the estimated joint accelerations were tested, before further, potentially unnecessary work was carried out. It was found that the joints estimated acceleration varied significantly depending on the sample rate. Further investigation suggested the unreal simulation caused the joints to "snap" to their target values as soon as the command was received, yet without transferring inertia, as long as the change in angle was small. Changes in angles greater than 2 degrees between two sequential target values were found to produce a huge force that flung the quadruped into the sky. These results strongly suggest that the degree to which physics is accurately modelled in the unreal engine is inconsistent. Subsequently, it is not possible to test or validate a dynamic model in unreal, due to the lack of data and the fact a dynamic model would be based on real-life physics which is different from simulated physics.

Ensuring the quadruped robot's stability during dynamic movement without direct measurements or accurate estimates of joint velocities, accelerations, torques, motor outputs, inertial properties, ground reaction forces, or environmental data presents a significant challenge. However, there are still conceptual approaches and simplified methods to approach the problem. One approach is to avoid dynamic modelling entirely by designing movement sequences that maximise stability by using known configurations and movements that historically maintain balance. Additionally, a series of kinematic responses could be designed to actively adjust the robot's posture preemptively during high-risk movements such as turning. The problem with this approach is its lack of adaptability to unforeseen situations, making this method unsuitable. Another potential approach is producing a machine learning model which would be trained to predict stability from the known inputs, whilst this approach is promising, the amount of data required for training from the relatively slow unreal simulations made this method unfeasible. Another option is to use

the available inputs to continuously assess whether the current stance of the quadruped would be stable if it was static, using the existing static model. However, after testing this approach in unreal, it became clear that this method was flawed. In dynamic stability, the ability to maintain or regain balance is not strictly tied to keeping the COM within the static support polygon at all times. When the COM moves outside the support polygon, if the robot has enough momentum and the movement is part of a controlled manoeuvre, it can shift its body to move the COM back into a stable position before a fall occurs.

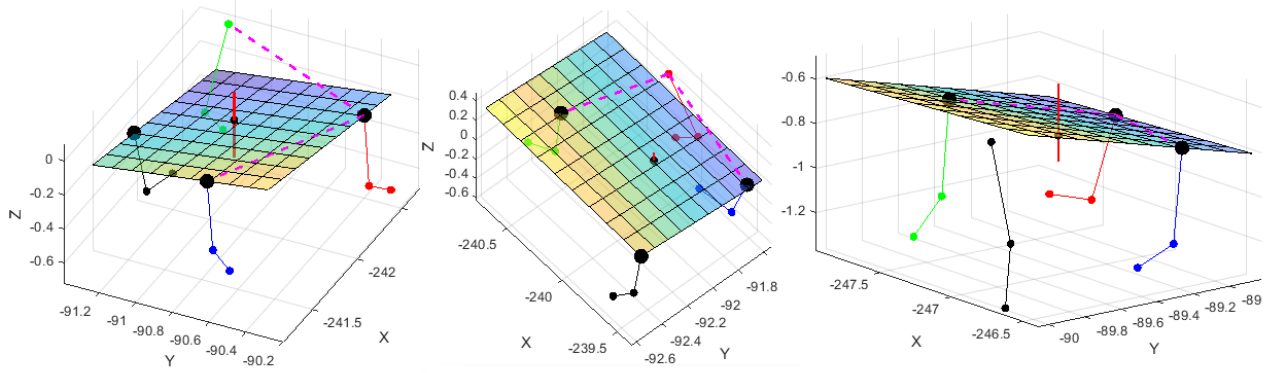

Figure 73 - Image showing COM outside polygon

To address this is to modify the static stability criterion to allow for controlled temporary excursions of the COM outside the support polygon. This can be achieved by defining a boundary around the support polygon where the COM can temporarily move into without initiating corrective action, acknowledging the robot's ability to recover. Whilst also taking into account the speed and direction of the COM movement, relative to the distance from the edge of the polygon. Limits on what is acceptable and for what duration can then be found from simulations, with a margin of error built in. This method was chosen as the only viable option.

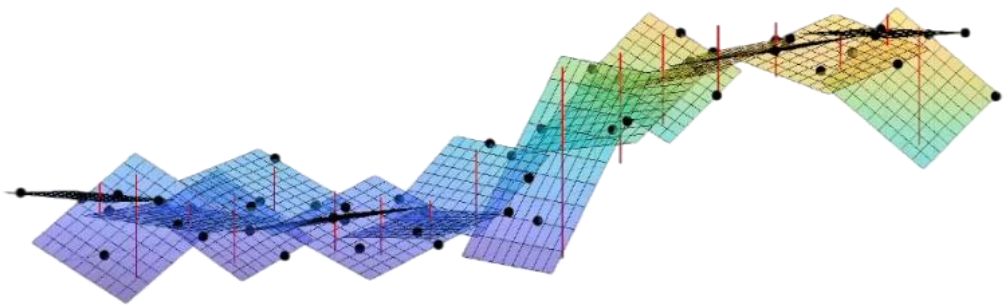

Figure 74 - change of COM, feet positions and polygon of support over time

Continuous data was imported from unreal simulations as a 3D matrix in the same format as the original 2D input matrix. Continuous static analysis was performed, saving relevant data.

To dynamically change the stability criterion based on dynamic factors they must first be found.

Matlabs in built functions were heavily utilised, to efficiently achieve this. The distance from COM to polygon edge was found using p_poly_dist1, The distance from COM to polygon centre using centroid, the same method used to find the centre of "overlap" polygons and base plate. The angle between COM and the polygon centre was found using atan2, a 2D projection was used, since the COM always points down, atan2 was also used to find the direction of COM change. The Polygon area was found using polyarea, the same method for finding areas of "overlap" polygons. To find the base plate angle relative to ground the cartesian vectors are first converted to a rotation matrix then to Euler angle using rotm2eul, this method was also used to find the direction of change of the base plate centroid. The rate of change of many parameters" found with differentiation, the direction of Polygon change was found with centroid and atan2. The Intersection of past and current polygons was found using intersect, the rate of change of the base plate centroid was found using pdist2. The next step was to use these parameters to implement the framework for assessing stability based on dynamic, varying boundary conditions. A configurable function was used to continuously perform this analysis for each "frame", using simple logic such as "if", "else", "for", etc. The parameters previously calculated, and the relationships between these parameters were used, for example the direction of COM change relative to the direction of polygon centroid change, and the rate of COM change relative to the direction of COM change, relative to the angle between the COM and the polygon centroid, relative to the distance between the COM and polygon edge. Relationships like this enable inference of the dynamic, for example if the COM is rapidly moving away from the polygon centroid and is already outside the boundary then it could be unstable. The final step was to configure the aforementioned functions rules. This was done based on what I expected logically, the behaviour seen in simulations, real-world quadruped dynamics, and research. The process used was to bypass all the undefined rules to begin with, then slowly add rules one by one, tuning them in the process. Using the large sample data set from simulations, cases where the COM was outside the polygon of support, yet stability was maintained were analysed visually. Code was written to display the frames continuously leading up to the erroneous result, with the whole quadruped and its centre of mass and polygon of support being displayed over time. This helped make insights into the causes of the false result.

This new method of dynamic stability assessment was tested by comparing the estimated stability to the actual stability seen in unreal simulations, using pre-defined movement patterns. The results showed that the new approach yielded significantly fewer false predictions of instability than when only the static analysis was used, however there were still some false predictions. It was difficult to test the extent to which false

predictions of stability were produced given the relatively few examples of this from simulation, due to the relatively flat terrain used in the simulation. Overall, this approach was useful and represented a significant improvement over the static only approach, however there were still limitations, likely caused by the method itself and the need for further rule refinement and tuning. It is also worth noting that this method was not computationally intensive, allowing for real time analysis.

The new method was utilised to meet the design requirement of being able to walk over rough terrain. The algorithm was used to enhance the predetermined walk cycles. This is because it allows for testing without time consuming simulations in unreal, enabling rapid iterative changes to be made and tested. As well as allowing for efficient fine tuning, the analysis can also guide the tuning process, adhering to the "rules" that govern the stability assessment is the aim of this process. However, the factors that are used such as the rate of change in support polygon area over time relative to direction of change are not obvious when designing the walk cycle manually. To compensate for this, visualising the outputs of the stability assessment alongside its stance, precise analysis of the walk cycles can be made, the point at which stability is lost is known, as well as the conditions leading up to that point. Inferences can then be made about how the walk cycle's design affects the complex factors, allowing for optimisation to prevent instability. The analysis could also be used for real time feedback control, to dynamically adapt the movement to prevent a loss of balance; this was not implemented as the walk cycles provided adequate stability for the terrain in the simulation, as well as the lack of terrain information and time constraints. Finally, the stability analysis could also be used to optimise the mechanical design of the robot, since these are configurable.

Conclusion

The process of estimating stability of the quadruped faced many challenges as a result of the simulation based nature of the project, critical thinking, trial and error and a willingness to try new methods allowed for these challenges to be overcome. The end result was not perfect but crucially it was good enough to meet the needs and requirements of the project. The development goal of enabling movement over uneven terrain was achieved, the algorithm was lightweight and highly configurable, ensuring it could be used in real time and be easily tuned and changed. The stability analysis program was able to be utilised to help improve the walk cycles, a crucial part of the final design. The extent to which the stability analysis impacted the final design was somewhat limited due to the fact the simulation environment was predictable and the terrain was fairly flat, removing the need for dynamic feedback control based on stability analysis.

4 INTEGRATIONS OF SUBSYSTEMS

4.1 MATLAB AND UNREAL ENGINE COMMUNICATION

Through the decision to utilise Unreal Engine as the modelling and simulation software, a comprehensive and continuous link between the MATLAB Simulink Controller and the Unreal Engine Environment needed to be established so that integration of the modules could begin. The Simulink side runs all control and processing that would be present in the microcontroller of a real quadruped robot, and the Unreal side runs all modelling and simulation of this system as if it were in a real physical environment. This set-up is important for the project because the functional requirements of the system cannot be verified without the display of its actions in a simulated environment.

The Vehicle Dynamics Block set Addon was installed to MATLAB for use in sending and receiving data with an Unreal Engine scene. The Addon is built to control and model different vehicles in a simulated environment, utilising the built-in Unreal Engine simulator to run basic physics for the model and enable visual validation of a vehicle system. The initial plan was to produce the model and terrain within this built-in simulator; however, the physics were too limited to fully recreate a physical environment for the quadruped and its many joints, links, forces, and sensors which could not be simulated here. Instead, the Addon was able to link directly to an existing Unreal Environment save within the actual Unreal Engine program, thus this was chosen for the sake of convenience, as many features such as physics constraints with target angle control, friction modelling, and terrain friction were included and simple to modify.

As MATLAB is communicating outside of the Simulink program, the Unreal save requires configuration to link to the program. The MathWorks Interface plugin should be installed, and the MathWorksSimulation Dependency needs to be built to the project file. The instructions for this process can be found on the MATLAB Help Centre page [34].

With these plugins set up, the Simulink program could be modified to send the controller angle outputs into Unreal. Each piece of data requires the following set-up to be communicated to Unreal:

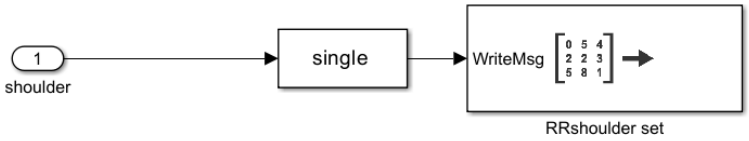

Figure 75. Simulink sending data for Rear Right Shoulder Joint

The continuous data must be converted to a single-precision value otherwise the Unreal Environment cannot recognise it. This is achieved with a simple data type conversion block. The data is then sent through a

"Simulation 3D Message Set" block, where it is defined with a certain tag. This tag will also be defined in a GetData actor in the Unreal Engine.

The Unreal Engine level blueprint is a global event graph in Unreal that allows actors to send and receive data as well as trigger actions for the 3D environment. From this blueprint the data sent to specific actors can be received and provided to other aspects of the environment, like for this case the target angle in a physics constraint joint. As shown below:

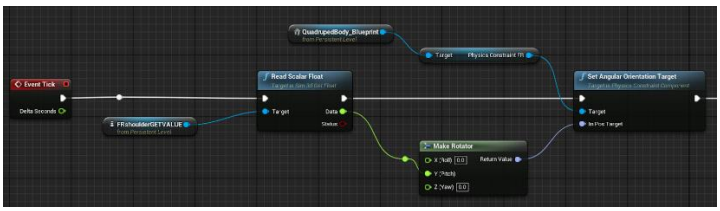

Figure 76. Level Blueprint receiving data for Front Right Shoulder Joint

The graph shown here displays the event tick block, such that the data can be updated every sample of the program. The graph then reads the scalar float from the GetValue actor, which had been defined with the same tag as that in Simulink. This float value is then sent to a rotator matrix block, assigning this joint to the pitch axis, which is then set as the angular orientation target for the quadruped actor's joint. This is then continued for all joints so that they are all updated at every sample tick. For sending data from Unreal to Simulink, the process is reversed.

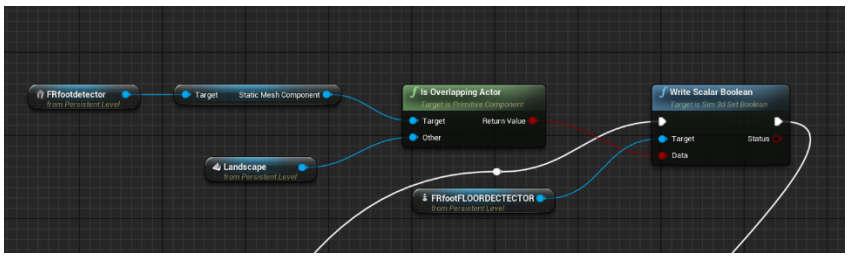

Figure 77. Level Blueprint sending data for Front Right Foot ground detection

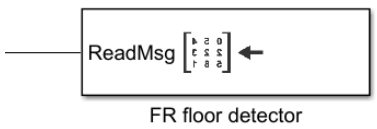

Figure 78. Level Blueprint receiving data for Front Right Shoulder Joint

The Boolean value of whether the front right foot is touching the ground is produced and written to a SetValue actor which is assigned the same tag as that defined in the "Simulation 3D Message Get" block

shown above. The value appears in this block at each sample and can then be used by the controller. This process is used for all separate data values being sent between Simulink and Unreal Engine.

4.2 QUADRUPED ON TERRAIN

The quadruped is required to be implemented into the environment created in the Unreal engine. The quadruped is modelled identically to the initial model used in the testing of kinematics in simulink, using premade structures in Unreal for each link and the body the quadruped consists of. These links were connected to each other and eventually to the body using Unreal's built-in feature *physics constraints*.

By changing certain values of these *physics constraints* these constraints were able to model rigid servo motors, which are used in real applications of quadrupeds. Once the model had been completed with rigid joints rotating in the correct plane and connecting the correct links, these joints had to be mapped to MATLAB and Simulink. This would allow the kinematics to be applied to the Unreal quadruped model using a data link between the two softwares.

4.3 KINEMATICS WITH UNREAL MODELLED QUADRUPED

With the communication between the two primary software complete, connection of the kinematics to the modelled quadruped can be achieved. The communication process is set up to provide all Unreal joints with angle values from the Simulink controller. The full level blueprint for this process is shown below:

The graph is set up in a 3 by 4 layout, covering all 12 moveable joints in the quadruped model. Each sample the Unreal Program receives these joint values, and updates the target angles of each joint, providing natural movement similar to that of a rigid servo motor.

4.4 PATH PLANNING WITH QUADRUPED

After the path planning process is completed, a reference path is produced, which contains information about which direction and the angle required by the quadruped to turn in order to reach the next node on the planned path. The path planning algorithm uses the same start position coordinates that the quadruped is placed in the terrain at the simulation's beginning. The quadruped walks forward, following the reference path until a large enough angle is found to represent an obstacle in the path, then the quadruped is told to stop within a radius of that point and turn to the new correct bearing. The quadruped continues to walk forward until it enters the next turning point and will continue to do this until the end goal has been reached. If the quadruped is detected to be far enough outside a straight section it stops and turns towards the original path. This is achieved by sending the quadruped body's positional data to the state controller which determines which walk state the quadruped should be in. A flowchart of this process is shown below:

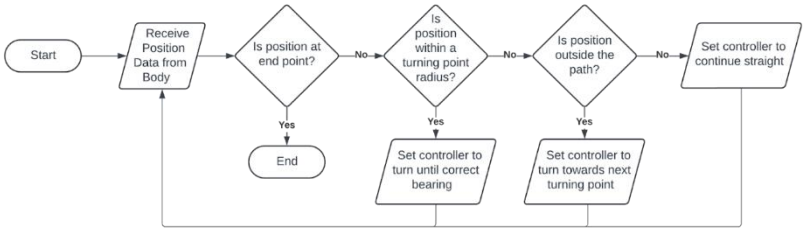

Figure 79. Flowchart for Path Finding state algorithm

4.5 COMPUTER VISION WITH QUADRUPED

The Input video for the computer vision is from a 3D simulation camera that has been placed on the front of the Unreal engine quadruped model. The corresponding 3D simulation camera block is then used to extract this video into Simulink. The traversal algorithm simply outputs either a '0', '1' or '-1'. These correspond to straight, left and right respectively and are then fed to the main quadruped controller which turns the robot.

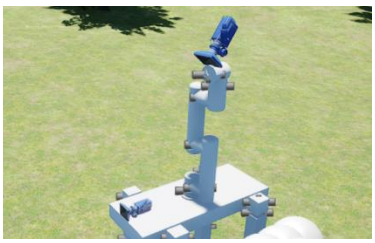

Figure 80. 3D Simulation Cameras on Quadruped

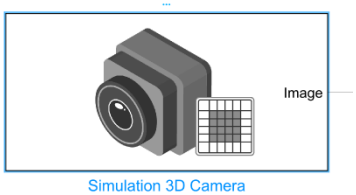

Figure 81. Simulation 3D Camera Block

4.6 MANIPULATOR WITH QUADRUPED AND KINEMATICS

In order to collect, carry and drop off the desired object a manipulator had to be modelled and implemented onto the quadruped in Unreal, with each joint angle being able to be modified via MATLAB & Simulink. The manipulator's links were initially modelled by premade cylinders in Unreal, that had been augmented in order to resemble the correct links of the manipulator model. This rough model was created in order to verify each joint was rotating in the correct plane, as well as making sure each joint resembled a rigid servo motor, with

the links being connected and joints being modelled using the same method as explained in Quadruped on Terrain.

Due to the manipulator mimicking the design of a Kinova link 6 [69], realistic joint models were created using CAD to create a realistic model of the manipulator. These CAD models were substituted into the model for the temporary cylinders used for verification, giving us the final design of our manipulator, Figure 82.

Figure 82 – Final Manipulator

As the manipulator has six joints the Simulink Program receives six Sim3D blocks to provide the data per sample to Unreal Engine. Each block defines a unique tag (a1 to a6) for the manipulator.

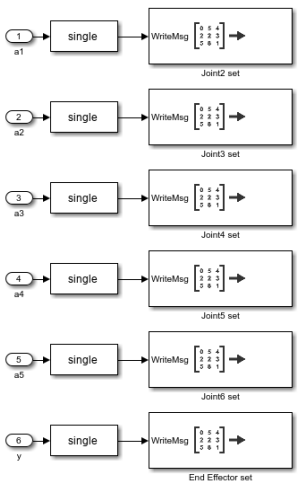

Figure 83. Simulink program for sending angles to all six joints

This data is then received by the level blueprint in Unreal Engine, applying these target joints to joint actors in the manipulator model.

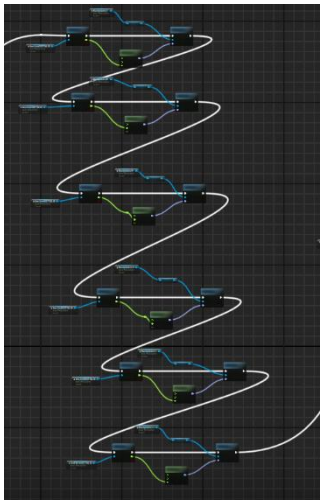

Figure 84. Level Blueprint for receiving angles of manipulator

This enables smooth movement of the manipulator as the controller which was previously discussed guides the end effector to target positions, such as the starting ball location and placing it at the end of the simulation.

4.7 COMPUTER VISION WITH MANIPULATOR

for the manipulator to pick up and select the correct ball, computer vision and the manipulator were combined, in order to achieve this a universal coordinate system was described within the MATLAB environment, attaching a MATLAB Camera Block and connecting with unreal and using the blob analysis developed during the computer vision development to highlight the red ball, placing nine balls within a three by three grid and assigning each balls location with a reference coordinate location. with the arm in the static home configuration positioned above this grid, the blob analysis from the computer vision will identify the red ball, transmit back to MATLAB that the ball has been identified, selecting the correct coordinate system to move too, once the ball is no longer detected by the computer vision the arm can identify that the ball has been connected, and the arm can pick the ball up and return back to its home configuration. This system could be further developed to rely solely on the camera's data to pick up the Object.

4.8 BALANCING WITH QUADRUPED

Integration was primarily used to improve the movement cycles. In this case a sample of data from unreal was imported into MATLAB as an array, with thousands of sample points. The program then analysed this data and provided meaningful insights and enabled fine tuning. The program can also directly run alongside the simulation, this was done for testing and verification.

5 OVERVIEW OF FINAL DEMONSTRATOR

5.1 FINAL DEMONSTRATOR ANALYSIS

The modelling environment's capabilities are demonstrated through the simulation of a quadruped with a manipulator which was tasked in picking up a ball using computer vision, navigating a tough terrain using computer vision and path planning, and then climbing a set of stairs where the ball was then dropped by the manipulator at a predetermined drop off point. This was successfully validated visually through a video which can be found using the following link here[71] a closer look at the simulation content can be seen in figure 85.

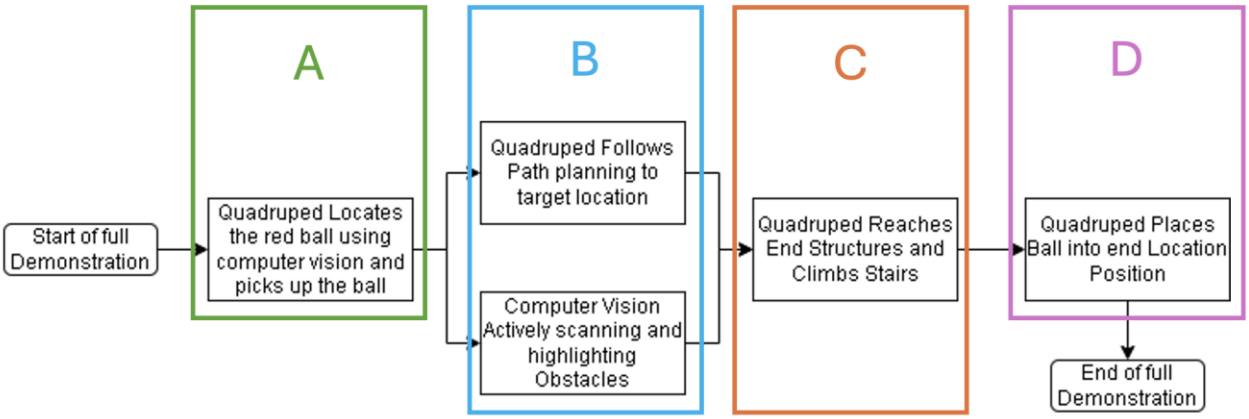

Figure 85 – Demonstrator breakdown

Manipulator picking up ball using computer vision (Figure 86 Sec A)

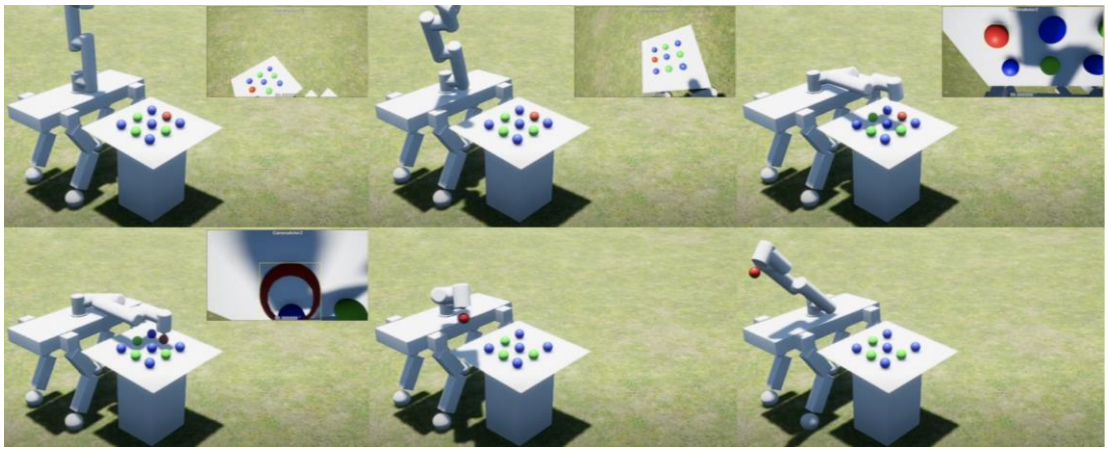

Figure 86 - Include photos of it happening with time stamp in caption 0:07-0:24

The manipulator locates the red ball out of an array of 9 balls (consisting of 1 red ball and 8 alternating blue and green balls) via the applications of computer vision. From there the manipulator's end effector is moved

- via inverse kinematics - to the location of the red ball. Each ball in the array holds coordinate values, when computer vision identifies the red ball in the array the correct coordinates are selected, and the manipulator moves the end effector to these coordinates.

The computer vision accurately identifies the correct colour of the ball using a blob analysis, with the manipulator smoothly and accurately transitioning to the identified location of the ball. To improve this system, rather than each ball containing coordinates the live feed can direct the arm to move in 3D space, making it more applicable for spontaneous manipulator use.

Quadruped using path finding and computer vision to traverse to end position (Figure 87 Sec B)

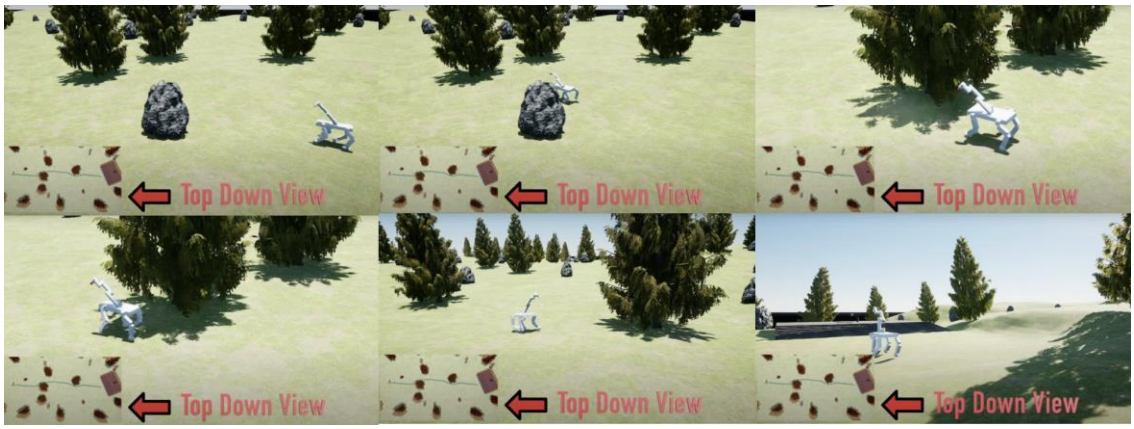

Figure 75 - 0:25-1:50

The quadruped is traversing across the terrain following the reference path in the bottom left corner. The quadruped walks forward, closely following the reference path until a large enough angle is found to represent an obstacle in the path, then the quadruped is told to stop within a radius of that point and turn to the new correct bearing. The quadruped continues to walk forward until it enters the next turning point and will continue to do this until the end goal has been reached.

The algorithm planned a collision-free path from the start point to the end point provided. The path planner could use the gradient of the terrain and plots a path that is flat, so it doesn't have to rely as heavily on the kinematics of the quadruped

Quadruped reaching end structure and climbing stairs (Figure 88 Sec C)

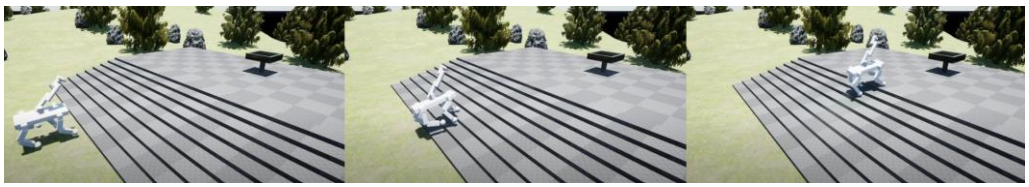

Figure 88 – Quadruped Climbing the stairs

Once reaching the end location, the quadruped ascends the stairs to reach the end location where it can complete the systems requirement and finish the simulation. By application of kinematics the quadruped is able to ascend the staircase of the end structure, with the balancing algorithm working alongside the kinematics, in order to ensure the quadruped remains stable and balanced.

The quadruped quickly and efficiently transcended the staircase, without any questionable movements or emergency balancing movements occurring. To show off the algorithms and capabilities of the quadruped further, turning on the staircase could be implemented such as a spiral staircase. Furthermore the quantity of stairs/height of steps could increase to test the system further.

Manipulator dropping the ball at the end position (Figure 89 Sec D)

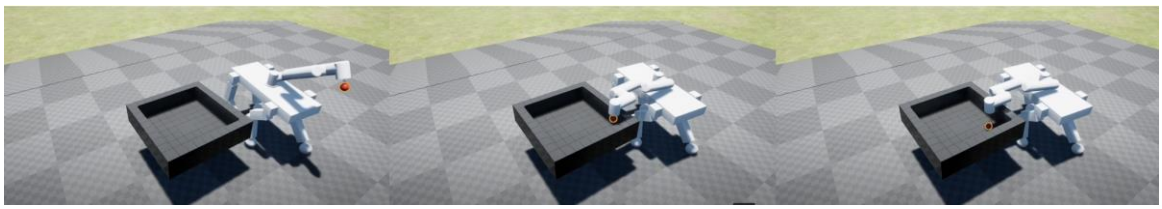

Figure 89 – Quadruped placing the ball

Upon ascending the staircase, the quadruped approaches the end destination where it is required to drop off the red ball, on top of a table, that it has been carrying whilst traversing through the terrain.

The quadruped stops next to the table due to the quadruped reaching the final location as identified by the path planning algorithm. Once in position, similar to collecting the red ball, the quadruped's manipulator will extend out over the table via inverse kinematics and release the ball dropping it onto the table.

Similar to the collection of the red ball, the manipulator is extended smoothly over the table. Upon extension of this manipulator (due to the manipulator having mass) the quadruped centre of mass is shifted requiring rebalancing to take place in order to ensure the quadruped remains stable. To improve this part of a demonstration a different, more complex end structure - such as reduced surface area - may be used where the quadruped would have to place whatever object the manipulator is holding onto. For example, in the case of the red ball, the end structure may be a tube. This would require high levels of accuracy from the manipulator in order to place the ball correctly in the tube.

5.2 REQUIREMENT ANALYSIS

The requirements have changed significantly from the feedback received from D1 due to a lack of understanding of the end goal of the project. A list of the initial requirements can be found in the appendix in Figure 99. A clear project aim was established changing the of scope of the project from making a physical

model to making a modelling platform that fuses the computational power of MATLAB with the modelling capabilities of Unreal Engine in order to provide a greater understanding and development in using the Unreal Engine's Simulation 3D Message Set in MATLAB's Simulink that companies can now use to expand on the project after completion.

Further to this, the project was separated into 7 clear subsystems as shown in Figure 3 which enables clearer, easier to integrate requirements. Assigning an owner to each of these subsystems reduced confusion and enabled functional and non-functional requirements be established as seen in Table 4.

The simulation can be compared to the final functional requirements of the system shown in Table 4. Each individual requirement is visually validated in a video found by the following link here [70]]. As evident in the video [70]] the majority of the set functional requirements were accomplished, with only one functional requirement being incomplete due to complexity issues and time constraints.

Requirement 1 states that the quadruped shall move forward and turn on flat terrain, with use of the shoulder joints and kinematic controller the quadruped is able to turn [70, 0:05 – 0:16]. Requirement 2 states that the quadruped shall move forward and turn on rough terrain, the quadruped use identical methods to the previously explained requirement, however this time the quadruped uses a balancing algorithm to stay balanced whilst traversing uneven terrain [70, 0:17-0:42]. Requirement 3 identifies the quadruped shall detect unexpected objects using computer vision, the quadruped uses computer vision to identify obstacles in front of it, with an algorithm then used to indicate the optimal way the quadruped should turn in order to avoid obstacles [70, 0:43-1:02]. Requirement 4 states that the quadruped shall traverse to the end position using path-planning, the quadruped uses a path planning algorithm with the angle the quadruped is required to turn being outputted to the quadruped at certain points of the journey [70, 1:03-1:32]. Requirement 5 states that the manipulator should be able to locate ball with computer vision, with use of computer vision the quadruped is able to identify a red ball out of an array of 9 coloured balls [70, 1:33-1:44]. Requirement 6 states the manipulator shall be able to pick up a ball at desired location, using the computer vision to identify which ball should be picked ip, the manipulator moves the end effector to this location and retrieves the ball [70, 1:45-2:02]. Requirement 7 states that the quadruped should walk upstairs, with the use of kinematics and balancing the quadruped is able to ascend the stairs efficiently and with improved stability [70], 2:03-2:12]. Requirement 8 states that the manipulator shall be able to drop off a ball at desired location, with use of balancing and the manipulator the quadruped is able to drop off the ball and the end location, as identified via path finding [70], 2:13-2:23].

As mentioned previously not all functional requirements were completed, with the one being incomplete being the simulation shall be solely run in MATLAB. This is due to running the simulation was more complex than Unreal as well as being extremely time taxing. However, although the whole simulation was not able to

be simulated within MATLAB, all the computational requirements and controllers were modelled and implement into the simulation via MATLAB.

6 TESTING VERIFICATION AND VALIDATION

The simulation testing involved validating all the previously listed functional requirements in both the Unreal Engine display and in any Simulink outputs. This is so verification of all decided functions of the system are achieved, and that the demonstrator video could display a comprehensive combination of all these actions.

Test 1: The Quadruped shall move forward and turn on flat terrain

The bare minimum that was set for the Quadruped behaviour was to traverse across completely flat terrain without any path finding control. The state controller is set on a timer such that it moves forward, turns right, continues forward, turns left, and so on. The purpose was to confirm the quadruped was not specifically designed just for uneven terrain and could function just as or even more efficiently across even terrain.

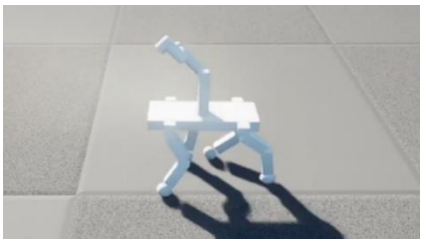

Figure 90. Testing on Flat Terrain

The controller provided stable gait for the quadruped through both the forward and turning traversal. The quadruped body stays relatively parallel to the ground throughout its walk, visually identifying the success of the stability. The turning state is successful however it appears to take longer compared to uneven traversal, which is potentially due to slopes aiding the turn when the quadruped falls onto each leg.

Test 2: The Quadruped shall move forward and turn on uneven terrain

Testing the quadruped on the uneven terrain was as simple as providing the same set up as with the previous test, and moving the quadruped onto uneven ground to compare stability. This was completed to verify that the quadruped could function across the generated terrain so that path finding and computer vision could be tested after.

Figure 91. Testing on Uneven Terrain

The quadruped displayed efficient traversal across slopes and hills, however was less stable along steeper gradients. As long as the quadruped was not in extreme terrain it could behave effectively, thus this functional requirement is verified and more advanced modules could be integrated.

Test 3: The Quadruped shall detect unexpected objects using computer vision

Computer vision was validated using the simulink model and the input from unreal engine. The terrain created was used to test the capabilities of the computer vision. The Simulink model overlays all the detected bounding boxes over the live feed as well as the confidence scores of these boxes. Although it was able to detect objects well, it wasn't very good at identifying what the objects were, often labelling rocks as trees.

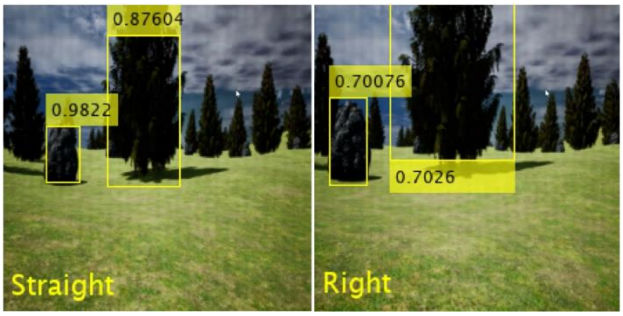

Figure 92. Demonstration of Computer vision

Test 4: The Quadruped shall traverse to the end position using path-finding

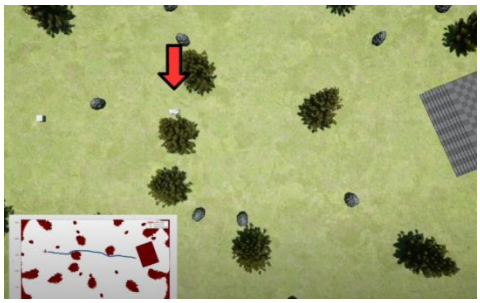

Figure 93 – path finding

The path finding algorithm was tested in matlab, shown in the bottom left corner, showing the path from the start to the end avoiding the red areas. This was further validation in unreal engine, using the quadruped to traverse through the terrain, again from beginning to end, avoiding the obstacles.

Test 5: The Manipulator should be able to locate the ball with computer vision

The blob analysis was tested in MATLAB, and proved to be extremely accurate, with the bounding box not flickering and no false positives being sensed.

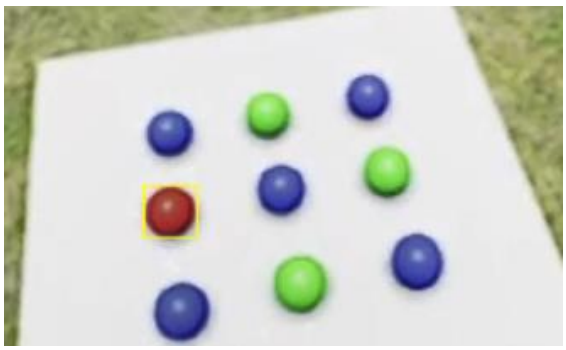

Figure 94 – Colour vision

Test 6: The Manipulator shall be able to pick up a ball at the desired location

With the computer vision working correctly, the manipulator system was provided the correct positioning of the red ball. With this the kinematic system was used to guide the end effector to this position, so that then the ball could be attached to the arm and returned to the normal position.

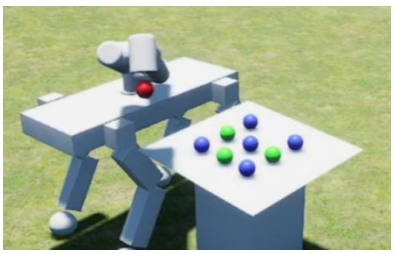

Figure 95. Manipulator picking up ball

The manipulator positioned itself directly above the ball and carried the ball into the position for traversal. This verifies that the ball can be carried from the start of the simulation and whilst traversing the terrain.

Test 7: The Quadruped should be able to walk up stairs

The quadruped was required to balance its walk up a staircase to reach the end point in the traversal. The quadruped was placed in front of the stairs and the state controller was set to climb. The balance module was necessary here to ensure the quadruped did not collapse walking up the stairs.

Figure 96. Testing Stair Climbing

The test showed the model climb the stairs successfully and finish at the top of the platform. Thus it was verified that the quadruped system could traverse the terrain towards the platform and then complete the climb to the ball drop-off point.

Test 8: The Manipulator shall be able to drop off the ball at a desired location

Once the quadruped finished full traversal to the end platform, the quadruped was required to drop the ball within the table. The quadruped was placed by this table and the manipulator was set to configure itself using the kinematic system into a position above the table. The ball was then detached from the end effector to complete the simulation.

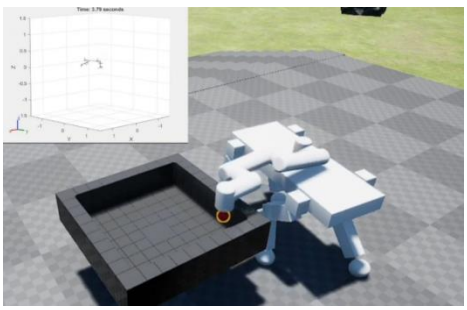

Figure 97. Quadruped dropping ball at drop-off point

As with Test 6, the manipulator arm movement was successful and configured smoothly to the target joints. The ball was placed in the table and thus the final function of the simulation was verified.

7 CONCLUSIONS AND FUTURE WORK

To recap, the aim of this project was to create a system to integrate MATLAB and Unreal Engine to simulate realistic scenarios of modelled quadrupeds demonstrating the capabilities of the system. The motivation behind this was to reduce the cost of prototyping in development of quadrupeds. This was achieved by creating a model of a quadruped with a manipulator which performed a task of picking up a ball, traversing over rough terrain, then placing the ball.

Future aims for this project could include:

- Simulating quadruped on a variety of surfaces, such as icy and Rocky terrains.
- Implementing an end effector of the manipulator to collect various shaped objects.
- Pathfinding using the gradient of the terrain as well as avoiding obstacles.

These future aims were not included within the project due to both time and financial restraints.

To conclude, the project has taught a variety of skills such as time management which made easier with the use of the V-model. A solution to the current problem of prototyping quadrupeds has been created, that is flexible and can be implemented to a wide range of real-life applications such as delivery quadrupeds and extra-terrestrial navigation. This can help during rapid prototyping allowing companies to adjust features on the fly without incurring costs and wasting time, which is financially attractive to companies interested in quadrupeds and robotics.

8 REFRRENCES

[1] M. Petrova, "Where four-legged robot dogs are finding work in a tight labor market." CNBC https://www.cnbc.com/2021/12/26/robotic-dogs-taking-on-jobs-in-security-inspection-and-public-safety-.html#:~:text=Boston%20Dynamic's%20entry%2Dlevel%20"explorer,included%20in%20the%20price%20tag (Accessed 16th may)

[2] S. Mudassar," V-Model Used in Software Development. ", June 2023. [Online]. Available: **https://www.researchgate.net/publication/371902849_V-Model_Used_in_Software_Development** **(Accessted 16th may)**

https://safetyculture.com/topics/waterfall-methodology/[3] R. Paredes, "Waterfall Methodology: The Pros and Cons." Safety Culture. **https://safetyculture.com/topics/waterfall-methodology/** (Accessed: 16th May)

[4] E. Auvee "WATERFALL VS. V MODEL: THE ULTIMATE COMPARISON" Impala Intech. **https://impalaintech.com/blog/waterfall-vs-v-model/#:~:text=Waterfall%20is%20a%20sequential%2C%20linear,focus%20on%20testing%20and%20verification.** (Accessed: 16th May)

Terrain references:

[5] M. C. Becker, P. Salvatore, and F. Zirpoli, "The impact of virtual simulation tools on problem-solving and new product development organisation," Research policy, vol. 34, no. 9, pp. 1305–1321, 2005, doi: 10.1016/j.respol.2005.03.016.

[6] V. G. Bharath, and P. Rajashekar, “Virtual Manufacturing: A Review” in National Conference Emerging Research Areas Mechanical Engineering Conference Proceedings, 2015, pp.355-364

[7] Unreal Engine, “USA’s largest car maker builds HMI systems in Unreal Engine.” Unreal Engine. https://www.Unrealengine.com/en-US/blog/usa-s-largest-car-maker-builds-hmi-systems-in-Unreal-engine (Accessed: 13th May 2024)

[8] T. Hilfert and M. König, “Low-cost virtual reality environment for engineering and construction,” Visualization in engineering, vol. 4, no. 1, p. 1, 2016, doi: 10.1186/s40327-015-0031-5.

[9] Y. Sun, H. Wang, Z. Zhang, C. Diels, and A. Asadipour, “RESenv: A Realistic Earthquake Simulation Environment based on Unreal Engine,” 2023, doi: 10.48550/arxiv.2311.07239

[10] X. Zhang, J. Liu, Q. Chen, H. Song, Q. Zhan, and J. Lu, “A 3D virtual Weft-knitting Engineering learning system based on Unreal Engine 4,” Computer applications in engineering education, vol. 26, no. 6, pp. 2223–2236, 2018, doi: 10.1002/cae.22030.

[11] E. Pukki, “Unreal Engine 4 for Automation”, BEng. Thesis, Elec. and Automation Eng., Metropolia Univ. of Applied Science, Helsinki, Finland, 2021. [Online]. Available: https://librarydevelopment.group.shef.ac.uk/referencing/ieee.html#headingDissThesis

[12] MathWorks, “Choose a Sensor for Unreal Engine Simulation.” MathWorks. https://uk.mathworks.com/help/driving/ug/choose-a-sensor-for-3d-simulation.html (Accessed: 13th May 2024)

[13] S. Ansari, H. Du, and F. Naghdy, “Driver's Foot Trajectory Tracking for Safe Maneuverability Using New Modified reLU-BiLSTM Deep Neural Network,” in 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC), IEEE, 2020, pp. 4392–4397. doi: 10.1109/SMC42975.2020.9283169.

[14] S. Ansari, F. Naghdy, H. Du, and Y. N. Pahnwar, “Driver Mental Fatigue Detection Based on Head Posture Using New Modified reLU-BiLSTM Deep Neural Network,” IEEE transactions on intelligent transportation systems, vol. 23, no. 8, pp. 10957–10969, 2022, doi: 10.1109/TITS.2021.3098309.

Manipulator/quadruped section references:

[15-29]

Gait references:

[30] M. H. Raibert and E. R. Tello, "Legged Robots That Balance," in *IEEE Expert*, vol. 1, no. 4, pp. 89-89, Nov. 1986, doi: 10.1109/MEX.1986.4307016. keywords: {Book reviews;Legged locomotion;Mobile robots;Service robots;Intelligent robots;Kinematics},

[31] Chufan Jiang, Shveta Dhamankar, Ziping Liu, Gautham Vinod, Gregory Shaver, John Evans, Corwin Puryk, Eric Anderson, Daniel DeLaurentis, 'Co-simulation of the Unreal Engine and MATLAB/Simulink for Automated Grain Off Loading', IFAC-PapersOnLine, Volume 55, Issue 24, 2022,

[32] Andaluz, Víctor & Chicaiza, Fernando & Gallardo, Cristian & Quevedo, Washington & Varela Aldás, José & Sánchez, Jorge & Arteaga, Oscar. (2016). Unity3D-MATLAB Simulator in Real Time for Robotics Applications. 246-263. 10.1007/978-3-319-40621-3_19.

[33] Akharas, Ismail & Hennessey, M. & Tornoe, Eric. (2020). Simulation and Visualization of Dynamic Systems in Virtual Reality Using SolidWorks, MATLAB/Simulink, and Unity. 10.1115/IMECE2020-23485.

[34] Mathworks "Get Started Communicating with the Unreal Engine Visualization Environment" https://uk.mathworks.com/help/vdynblks/ug/get-started-communicating-with-the-Unreal-engine-visualization-environment.html (Accessed: 14th May 2024)

[35] L. Skrba, L. Reveret, F. Hétroy, M.-P. Cani, and C. O'Sullivan, "Animating Quadrupeds: Methods and Applications," *Computer Graphics Forum*, vol. 28, no. 6, pp. 1541–1560, Sep. 2009, doi: https://doi.org/10.1111/j.1467-8659.2008.01312.x.

[36] G. Xin, F. Zeng, and K. Qin, "Loco-Manipulation Control for Arm-Mounted Quadruped Robots: Dynamic and Kinematic Strategies," *Machines*, vol. 10, no. 8, p. 719, Aug. 2022, doi: https://doi.org/10.3390/machines10080719.

[37] B. Sandeep and P. Tamil Selvan, "Design and Development of an Autonomous Quadruped Robot," *IOP Conference Series: Materials Science and Engineering*, vol. 1012, no. 1, p. 012016, Jan. 2021, doi: https://doi.org/10.1088/1757-899x/1012/1/012016.

[38] Rahman, Md Hasibur & Islam, Md & Al Monir, Md & Alam, Saadia & Ruzbelt, & Rahman, Md & Shidujaman, Mohammad & Islam, Rubaiyat. (2022). Kinematics analysis of a quadruped robot: Simulation and Evaluation. 1-6. 10.1109/ICIPRob54042.2022.9798744.

[39] A. Shahriar, "A Simulation-based Approach to Kinematics Analysis of a Quadruped Robot and Prototype Leg Testing," *arXiv.org*, Feb. 19, 2024. https://arxiv.org/abs/2312.06365 (accessed May 15, 2024).

[40] P. G. de Santos, E. Garcia, and J. Estremera, *Quadrupedal Locomotion: An Introduction to the Control of Four-legged Robots*. Springer London, 2012. Accessed: May 15, 2024. [Online]. Available: https://books.google.co.uk/books/about/Quadrupedal_Locomotion.html?id=P0yrcQAACAAJ&redir_esc=y

[41] J. Z. Kolter, M. P. Rodgers and A. Y. Ng, "A control architecture for quadruped locomotion over rough terrain," 2008 IEEE International Conference on Robotics and Automation, Pasadena, CA, USA, 2008, pp. 811-818, doi: 10.1109/ROBOT.2008.4543305.

[42] N. Meng and W. Xiaodong, "Kinematics analysis and simulation of quadruped robot," Proceedings of 2011 International Conference on Fluid Power and Mechatronics, Beijing, China, 2011, pp. 816-821, doi: 10.1109/FPM.2011.6045874.

Computer vision references:

[43] K. Itakura and F. Hosoi, “Automatic tree detection from three-dimensional images reconstructed from 360 spherical camera using YOLO v2,” *Remote sensing (Basel, Switzerland)*, vol. 12, no. 6, p. 988, doi: 10.3390/rs12060988.

[44] O. Risbøl and L. Gustavsen, “LiDAR from drones employed for mapping archaeology – Potential, benefits and challenges,” *Archaeological prospection*, vol. 25, no. 4, pp. 329–338, 2018, doi: 10.1002/arp.1712.

[45] S. Rajkumar, A. Hariharan, S. Girish, and M. Arulmurugan, “An Efficient Vehicle Detection and Shadow Removal Using Gaussian Mixture Models with Blob Analysis for Machine Vision Application,” *SN computer science*, vol. 4, no. 5, 2023, doi: 10.1007/s42979-023-01832-y.

[46] K. Boudjit and N. Ramzan, “Human detection based on deep learning YOLO-v2 for real-time UAV applications,” *Journal of experimental & theoretical artificial intelligence*, vol. 34, no. 3, pp. 527–544, 2022, doi: 10.1080/0952813X.2021.1907793.

[47] Y. Xue, Z. Ju, Y. Li, and W. Zhang, “MAF-YOLO: Multi-modal attention fusion based YOLO for pedestrian detection,” *Infrared physics & technology*, vol. 118, p. 103906, 2021, doi: 10.1016/j.infrared.2021.103906.

[48] T. Diwan, G. Anirudh, and J. V. Tembhurne, “Object detection using YOLO: challenges, architectural successors, datasets and applications,” *Multimedia tools and applications*, vol. 82, no. 6, pp. 9243–9275, 2023, doi: 10.1007/s11042-022-13644-y.

[49] Z. Jiang, H. Zhang, J. Xu, G. Tian, and X. Rong, “Real-Time Target Detection and Tracking System Based on Stereo Camera for Quadruped Robots,” in *2019 Chinese Control And Decision Conference (CCDC)*, IEEE, 2019, pp. 2949–2954. doi: 10.1109/CCDC.2019.8833212.

[50] N. A. A. Norizan, M. R. Md Tomari, and W. N. Wan Zakaria, "Object Detection Using YOLO for Quadruped Robot Manipulation", *EEEE*, vol. 4, no. 1, pp. 329–336, May 2023, Accessed: May 13, 2024. [Online]. Available: https://publisher.uthm.edu.my/periodicals/index.php/eeee/article/view/10778

[51] T.Dewi, Z.Mulya, P.Risma and Y.Oktarina, "BLOB analysis of an automatic vision guided system for a fruit picking and placing robot", in *2021 International Journal of Computational Vision and Robotics,* vol. 11, no. 3, pp. 315-327, doi: 10.1504/IJCVR.2021.115161

[52] MathWorks, "Computer Vision Toolbox" MathWorks. https://uk.mathworks.com/help/vision/ (Accessed: May 15, 2024).

[53] MathWorks, "Object Detection Using YOLO v2 Deep Learning" MathWorks. https://uk.mathworks.com/help/deeplearning/ug/object-detection-using-yolo-v2.html (Accessed: May 15, 2024).

[54] MathWorks, "Get Started with the Image Labeler" MathWorks. https://uk.mathworks.com/help/vision/ug/get-started-with-the-image-labeler.html (Accessed: May 15, 2024).

[55] MathWorks, "jitterColorHSV" MathWorks. https://uk.mathworks.com/help/images/ref/jittercolorhsv.html (Accessed: May 15, 2024).

[56] MathWorks, "trainYOLOv2ObjectDetector" MathWorks. https://uk.mathworks.com/help/vision/ref/trainyolov2objectdetector.html (Accessed: May 15, 2024).

Path finding references:

[57] Unreal Engine, "Basic Navigation," *docs.Unrealengine.com*. https://docs.Unrealengine.com/4.27/en-US/InteractiveExperiences/ArtificialIntelligence/NavigationSystem/BasicNavigation/

[58] S. Erke, D. Bin, N. Yiming, Z. Qi, X. Liang, and Z. Dawei, "An improved A-Star based path finding algorithm for autonomous land vehicles," *International Journal of Advanced Robotic Systems*, vol. 17, no. 5, p. 172988142096226, Sep. 2020, doi: https://doi.org/10.1177/1729881420962263.

[59] Z. Wu, Z. Meng, W. Zhao, and Z. Wu, "Fast-RRT: A RRT-Based Optimal Path Finding Method," *Applied Sciences*, vol. 11, no. 24, p. 11777, Dec. 2021, doi: https://doi.org/10.3390/app112411777.

[60] A. Protsenko and V. Ivanov, "COMPARATIVE ANALYSIS OF RRT-BASED METHODS FOR PATHFINDING IN UNDERGROUND ENVIRONMENT," *Advanced Information Systems*, vol. 4, no. 3, pp. 109–112, Oct. 2020, doi: https://doi.org/10.20998/2522-9052.2020.3.15.

Stability analysis references

[61]M. Toussaint, “Robotics Dynamics.” Accessed: May 17, 2024. [Online]. Available: https://www.user.tu-berlin.de/mtoussai/teaching/14-Robotics/03-dynamics.pdf

[62]R. Featherstone and D. Orin, “Robot Dynamics: Equations and Algorithms.” Available: https://homes.cs.washington.edu/~todorov/courses/amath533/FeatherstoneOrin00.pdf

[63]W. Sun, X. Tian, Y. Song, B. Pang, X. Yuan, and Q. Xu, “Balance Control of a Quadruped Robot Based on Foot Fall Adjustment,” Applied Sciences, vol. 12, no. 5, p. 2521, Feb. 2022, doi: https://doi.org/10.3390/app12052521.

[H4]D. T. Polet and John, “Competing Models of Work in Quadrupedal Walking: Center of Mass Work is Insufficient to Explain Stereotypical Gait,” Frontiers in Bioengineering and Biotechnology, vol. 10, May 2022, doi: https://doi.org/10.3389/fbioe.2022.826336.

[65] J. Meng, Y. Li, and B. Li, “A Dynamic Balancing Approach for a Quadruped Robot Supported by Diagonal Legs,” International Journal of Advanced Robotic Systems, vol. 12, no. 10, p. 142, Oct. 2015, doi: https://doi.org/10.5772/61542.

[66]R. Mcghee and A. 1 ', “UNIVERSITY OF SOUTHERN CALIFORNIA OP7TMUM QUADRUPED CREEPING GAITS.” Accessed: May 17, 2024. [Online]. Available: https://apps.dtic.mil/sti/tr/pdf/AD0675256.pdf

[67]E. Papadopoulos and D. A. Rey, “The Force-Angle Measure of Tipover Stability Margin for Mobile Manipulators,” Vehicle System Dynamics, vol. 33, no. 1, pp. 29–48, Jan. 2000, doi: https://doi.org/10.1076/0042-3114(200001)33:1;1-5;ft029.

[68]M. VUKOBRATOVIĆ and B. BOROVAC, “ZERO-MOMENT POINT — THIRTY FIVE YEARS OF ITS LIFE,” International Journal of Humanoid Robotics, vol. 01, no. 01, pp. 157–173, Mar. 2004, doi: https://doi.org/10.1142/s0219843604000083.

[69] Kinova, “Kinova Link 6 Industrial Evolution.” Kinova Robotics. https://www.kinovarobotics.com/product/link-6-cobot (Accessed 16th May 2024).

[70] L. Argyrou [lucas argyrou], ACS330 – Requirements, May. 14, 2024. [Video]. Accessed: May. 14, 2024. Available: https://www.youtube.com/watch?v=crjXywnZG1Q&t=1s

[71] L. Argyrou [lucas argyrou], ACS330 – Final demonstrator, May. 14, 2024. [Video]. Accessed: May. 14, 2024. Available: https://www.youtube.com/watch?v=qs7VKDzolAo

[72] Chufan Jiang, Shveta Dhamankar, Ziping Liu, Gautham Vinod, Gregory Shaver, John Evans, Corwin Puryk, Eric Anderson, Daniel DeLaurentis, 'Co-simulation of the Unreal Engine and MATLAB/Simulink for Automated Grain Off Loading', IFAC-PapersOnLine, Volume 55, Issue 24, 2022,

[73] Andaluz, Víctor & Chicaiza, Fernando & Gallardo, Cristian & Quevedo, Washington & Varela Aldás, José & Sánchez, Jorge & Arteaga, Oscar. (2016). Unity3D-MatLab Simulator in Real Time for Robotics Applications. 246-263. 10.1007/978-3-319-40621-3_19.

[74] Akharas, Ismail & Hennessey, M. & Tornoe, Eric. (2020). Simulation and Visualization of Dynamic Systems in Virtual Reality Using SolidWorks, MATLAB/Simulink, and Unity. 10.1115/IMECE2020-23485.

APPENDIX

Table 9 – Table of Requirements

ID	Requirements	Verification method	Verification Owner
Functional Requirements			
1	The quadruped shall move forward and turn on flat terrain	Visual – Unreal	Oliver Burge
2	The quadruped shall move forward and turn on uneven terrain	Visual – Unreal	Hari Nanda
3	The quadruped shall detect unexpected objects using computer vision	Visual – Simulink	George Craft
4	The quadruped shall traverse to the end position using pathfinding.	Visual – MATLAB & Unreal	Jenna Hazard
5	The manipulator should be able to locate ball with computer vision	Visual – Simulink	Oliver Rawlings

6	The manipulator shall be able to pick up a ball at the desired location.	Visual – Unreal & Simulink	Lucas Argyrou
7	The quadruped should walk up stairs	Visual – Unreal	Oliver Burge
8	The manipulator shall be able to drop off a ball at desired location.	Visual – Unreal	Jamie Whitaker
9	The simulation shall be completely run through MATLAB	No need for unreal installed	Lucas Argyrou
Non-Functional Requirements			
1	**Environment Generation**		
1.1	Terrain shall include random uneven gradients	Visual - Unreal	Oliver Rawlings
1.2	Terrain shall include obstacles	Visual - Unreal	Oliver Rawlings
1.2.1	Obstacles shall include trees	Visual - Unreal	Oliver Rawlings
1.2.2	Obstacles shall include rocks	Visual - Unreal	Oliver Rawlings
1.3	Terrain shall include end structure	Visual - Unreal	Oliver Rawlings
1.3.1	End structure shall have platform with stairs leading to it	Visual - Unreal	Oliver Rawlings
1.3.2	End structure shall have a table to pick up ball	Visual - Unreal	Oliver Rawlings
1.3.3	End structure shall have a table to drop off ball	Visual - Unreal	Oliver Rawlings
1.4	Terrain should include randomly placed static objects	Visual - Unreal	Oliver Rawlings
1.5	Terrain should include moving objects	Visual - Unreal	Oliver Rawlings
2	**Kinematic and Simulation Modelling**		
2.1	The quadruped shall be modelled in Simscape	Visual - Simscape	Jamie Whitaker
2.2	The quadruped shall have 3 joints per leg	Visual - Kinematic diagram	Jamie Whitaker
2.2.1	The individual joints shall receive input angle data and follow them	Visual - MATLAB	Jamie Whitaker

2.3	Manipulator shall have 6 dof	Visual - Kinematic diagram	Jamie Whitaker
2.4	Manipulator should be to reach underneath itself	Visual workspace	Jamie Whitaker
2.5	Manipulator should go to inputted coordinates	Visual - MATLAB	Lucas Argyrou
2.6	The manipulator shall have trajectory smoothing	Visual MATLAB	Lucas Argyrou
2.7	The manipulator shall have a Simscape model	Visual - Simscape	Lucas Argyrou
2.8	The manipulator shall have a gripper	Visual - CAD	Lucas Argyrou
2.8.1	The gripper should be attached to the manipulator	Visual - Unreal	Lucas Argyrou
3	**Gait modelling**		
3.1	The quadruped should have all 12 joints in Unreal connected to the controller in Simulink	Visual - Simulink & Unreal	Oliver Burge
3.2	The controller shall provide a walking pattern for each cycle when set to walk forward	Visual graph - Simulink	Oliver Burge
3.3	The controller shall provide a turning pattern for each cycle when set to turn	Visual graph - Simulink	Oliver Burge
3.4	The forward walk shall be modified to allow the quadruped to climb a staircase	Visual graph - Simulink	Oliver Burge
4	**Computer Vision**		
4.1	The quadruped shall detect trees in the terrain	Visual – Simulink	George Craft
4.2	The quadruped shall detect rocks in the terrain	Visual – Simulink	George Craft
4.3	The quadruped shall choose a path depending on the objects it sees	Visual – Simulink	George Craft
4.4	The manipulator shall detect a ball	Visual – Simulink	Lucas Argyrou
5	**Path Finding**		
5.1	The path finding algorithm shall be able to detect obstacles	Visual - MATLAB	Jenna Hazard
5.1.1	The path finding algorithm shall be able to detect trees in the terrain	Visual - MATLAB	Jenna Hazard
5.1.2	The path finding algorithm shall be able to detect rocks in the terrain	Visual - MATLAB	Jenna Hazard
5.2	The path finding algorithm shall be able to produce a collision-free path	Visual - MATLAB & Unreal	Jenna Hazard

5.3	The path finding algorithm should produce an optimal path	Visual - MATLAB	Jenna Hazard
5.4	The path finding algorithm shall be able to find a path from beginning to end if a solution exists	Visual - MATLAB & Unreal	Jenna Hazard
6	**Stability Analysis**		
6.1	The analysis program shall be able to run in real time with a high sample rate.	Confirmed in MATLAB	Hari Nanda
6.2	The program must provide explainability.	Visual - MATLAB	Hari Nanda
6.3	The program shall only use data that would be realistically available.	Confirmed in MATLAB	Hari Nanda
6.4	The analysis program shall assess stability accurately the vast majority of time, with only infrequent erroneous false assessments.	Visual - MATLAB	Hari Nanda
7	**Integration of MATLAB with Unreal:**		
7.1	The main programming shall be coded within MATLAB		Jamie Whitaker
7.2	The modelling shall occur in MATLAB's Simulink.		Jamie Whitaker
7.3	The program output shall be visualised within Unreal engine 4.	Visual - Unreal	Oliver Burge
7.4	The two platforms shall be synced and the simulation times shall be aligned.	Confirmed in Simulink and Unreal sample values	Oliver Burge
7.5	Unreal Blueprint should provide all necessary environment data to the controller.	Visual - Simulink	Oliver Burge

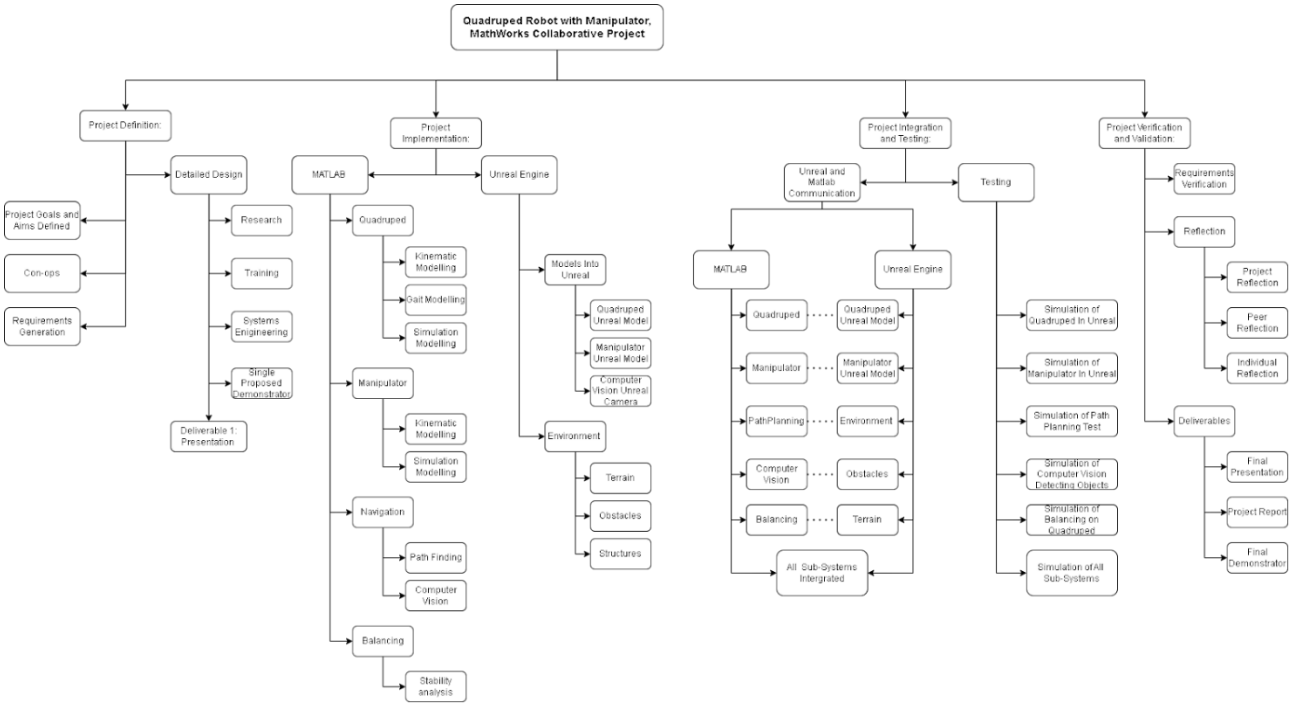

Figure 96– work breakdown structure

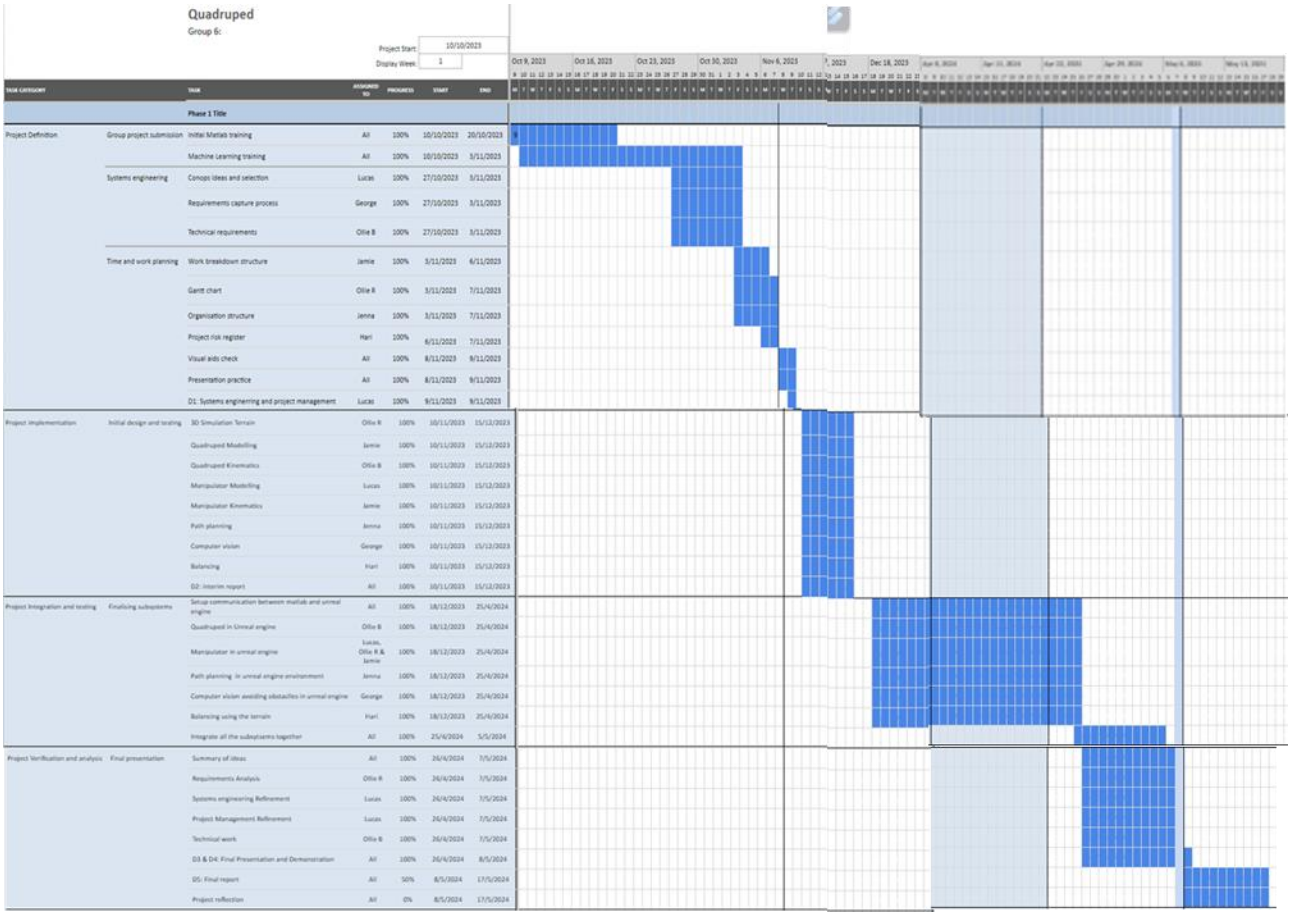

Figure 97 – Gantt Chart

Miniature Report Document link

https://docs.google.com/document/d/1j2rZ9HkYP1LEKnUlygSGsyo8L3FAWsCCsF4nGOs3w/edit?usp=sharing

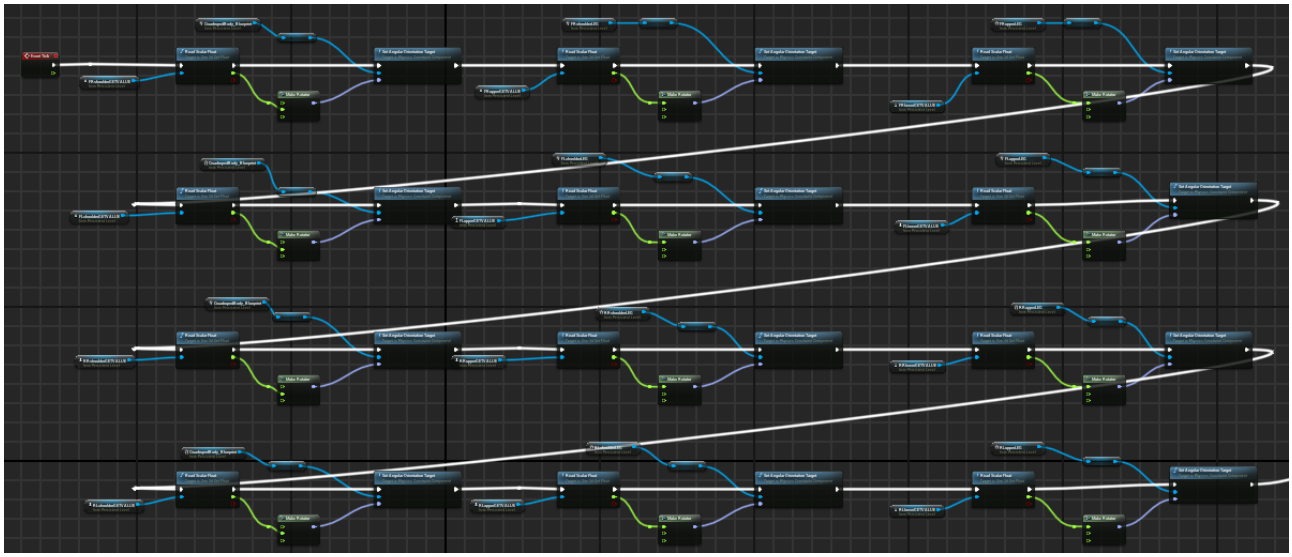

Figure 98 - Entire Level Blueprint graph for joint control

Initial Requirements

Functional Requirements:

1.1. The quadruped shall be able to walk forward at a user predetermined speed.

1.2. The quadruped shall have the ability to turn left and/or right.

1.3. The quadruped shall have animal-like joint mobility.

1.4. The quadruped shall be able to navigate over rough terrain.

1.5. The quadruped shall be able to maintain balance whilst moving.

1.6. The quadruped shall have sensors that will integrate with the Unreal engine.

1.7. The simulation shall contain a representation of a realistic physical environment with rough terrain.

1.8. The quadruped shall have an attached 3-axis manipulator.

1.9. The quadruped shall have computer vision to correctly identify objects.

1.10. The quadruped shall navigate and be able to use the manipulator at the same time.

Performance Requirements:

2.1. The quadruped shall have a user adjustable walking speed.

2.2. The quadruped's turning radius shall be predefined and no greater than the user stated value.

2.3. The sensor's performance specs, detection range and FOV shall be similar to readily available sensors.

2.4. The quadruped's manipulator shall have a predefined operating speed.

2.5. The quadruped's manipulator operating range shall be predefined.

Low-Level Requirements:

Quadrupeds Kinematics motion and Manipulator:

1.1. Define joint setup configurations, link lengths, rotation and any other parameters and measurements needed for the quadruped's legs.

1.2. Implement both forward and inverse kinematics models.

1.3. Dynamics of the robot shall take into account masses, inertias and forces acting on each leg will be modelled, the centre of mass of each link should be considered as part of the whole system.

1.4. Implement inverse kinematics models for the manipulator.

1.5. To achieve animal-like mobility the quadruped shall have 3 joints per leg.

Control Systems:

2.1. Design a successful feedback controller to maintain robots' stability and balance.

2.2. Implement a controller for the walking motion, turning operations and balancing.

2.3. Systems controlling the robot's movement and balance should be rapid and react almost instantly to the changing dynamic situation.

Sensor Modelling:

3.1. Implement a model for obstacle avoidance and detection sensors, LIDAR, Load Cell sensor, accelerometers.

3.2. Implement a model for obstacle interaction and picking up, e.g. pressure sensors or capacitive touch sensors.

3.3. Defined parameters such as sensor range, resolution, and baud rate.

3.4. Integrate simultaneous sensor readings if multiple sensors are used.

Terrain Modelling:

4.1. Include Multiple different terrain scenarios and environments and a classification system for them. Flat, rocky, inclined, incremental and stairs.

4.2. Record the interaction between the robot's feet and the contact terrain it touches, considering forces acting on it and any slippages.

Integration with Simulation Tools:

5.1. Define the method for data acquisition between MATLAB and the Unreal Engine add-on.

5.2. Ensure the two platforms are synced and the simulation times are aligned.

5.3. Provide extensive error handling and logging.

Testing and Validation:

6.1 Create test scenarios and environments to validate the robot's ability to navigate different terrains and scenarios and the ability to document progress using these test scenarios ready for final implementation.

6.2. Verify accuracy and reliability of sensors.

6.3. Make stress test scenarios to determine simulation limits.

Figure 99 - Requirements list

Table 10 – Risk Register

Risk	Risk Description	Impact Description	Impact Level	Probability Level	Mitigation Notes	Risk Owner
Nature of the risk.	Summary of the risk.	What will happen if the risk is not mitigated or eliminated.	Rate 1 (LOW) to 5 (HIGH)	Rate 1 (LOW) to 5 (HIGH)	What can be done to reduce the risks impact or probability.	Who will be responsible for the risk.

Injuries Due to Robotic Malfunction	Unexpected behaviour or failure of the quadruped that could impact or entrap the operator's limbs.	Bruising Cuts/Abrasion Fractures	2	3	Training regarding quadruped handling and operation.	All
Electrical Shocks	Risk of electric shock during maintenance, charging or handling the quadruped and associated electronics.	Muscle Spasms Paraesthesia Burns	4	1	Testing equipment regularly.	All
Tripping Over Cables or Equipment	Cables or equipment on the floor could pose tripping hazards in the work area.	Bruising Fractures Concussion	5	2	Keeping cables and equipment clear of the walkways at all times, ensuring it is properly stored.	All
Repetitive Motion Injuries	Repetitive tasks such as coding or assembly of small components can cause carpal tunnel syndrome or tendonitis.	Tendonitis CTS	3	2	Ensure regular breaks are taken,	All
Eye Strain	Extended periods of computer use without proper eye protection or breaks	Visual Disturbances Eye Fatigue	2	3	Ensure regular breaks are taken,	All

	could result in digital eye strain.	Irritated Eyes Headaches				
Slips or Spills	Spills of any liquids in the workspace could cause slipping accidents.	Bruising Fractures Concussion	5	2	Only bring liquid which are stored with a lid into the workspace.	All
Cuts or Abrasions	Working with tools or the quadrupeds' mechanical parts could lead to cuts or abrasion.	Nerve Damage Abrasions Cuts	4	1	Training on proper tool usage.	All
Strain from Improper Ergonomics	Incorrect chair height, monitor placement or keyboard placement can lead to back or neck injuries.	Neck Injuries Back Injuries	3	3	Follow expert guidance relating to the workstation setup.	All
Chemical Exposure	There is a risk from handling batteries used within the quadruped.	Chemical Burns	5	1	Use PPE when handling batteries, plus training on battery handling.	All
Thermal Injuries	Overheating components of the quadruped could possibly cause burns.	Burns	5	1	Do not touch the quadruped's components during or after the operation.	All

Computer vision does not work	The risk that the computer vision is not completed, does not work, or can't be integrated.	The robot will not be aware of its surroundings and will be unable to navigate.	5	1	George will take personal responsibility for this aspect of development.	George C
Terrain Generation does not work	The risk that the terrain generation either is not completed or does not work correctly.	We will be unable to demonstrate the capability of the robot to walk over rough terrain.	4	1	Ollie R will take personal responsibility for this aspect of development.	Ollie R
Path Finding does not work	The risk that the path finding either is not completed or does not work to find suitable paths.	The robot will not follow correct paths, potentially leading to it falling over.	4	2	Jenna will take personal responsibility for this aspect of development.	Jenna H
Leg Kinematics does not work	The risk that the leg kinematics is not completed or produced poor leg control leading to poor outcomes.	The robot will be unable to move.	5	1	Ollie B will take personal responsibility for this aspect of development.	Ollie B
Balancing does not work	The risk that the balancing algorithm is not completed or that it does not	The robot will be unable to move reliably.	5	2	Hari will take personal responsibility for	Hari N

	balance the quadruped as intended.				this aspect of development.	
Quadruped Model does not work	The risk that the quadruped model is not completed or does not accurately model the quadruped.	We will not be able to deliver the initial project goal of simulating a quadruped.	5	1	Jamie will take personal responsibility for this aspect of development.	Jamie W
Problems with integration of algorithms,	The risk that the various algorithms can not be effectively integrated together into the final simulation.	The robot may not behave as intended, the deliverables may not be met.	5	2	All the various aspects of the project will be developed with integration in mind. Good communication.	Lucas A
Deliverables are not delivered	The risk that the deliverables are either not delivered, delivered late or delivered poorly.	Everyone in the group would receive a poor mark for the module.	5	2	Each member of the group will take responsibility for deliverables.	All
Team cohesion breaks down.	The risk that the group stops working together effectively.	The various parts of development are not integrated. Deliverables are not delivered.	4	2	Ensure the group meets on a weekly basis, ensure everyone knows their roles. Maintain contact.	Lucas A

Slow progression of the project.	The risk that as a collective, the project is developed too slowly resulting in suboptimal outcomes.	Not all the initial project goals are achieved.	4	2	Set clear goals for each aspect of development with deadlines.	Lucas A