TALLINNA TEHNIKAÜLIKOOL
TALLINN UNIVERSITY OF TECHNOLOGY

Department of Electrical Power Engineering and Mechatronics

IMITATIONS OF PERCEPTIONS PROCESSES FOR AN ARTIFICIAL LIFEFORM

TAJUPROTSESSI IMITEERIMINE KUNSTLIKU ELUVORMI POOLT

MASTER THESIS

Student: Vladislav Babuškin

Student code 163216MAHM

Supervisor: Mart Tamre, Professor

Tallinn, 2018

AUTHOR'S DECLARATION

Hereby I declare, that I have written this thesis independently.
No academic degree has been applied for based on this material. All works, major viewpoints and data of the other authors used in this thesis have been referenced.

"....." ..... 201.....

Author: .....
/signature /

Thesis is in accordance with terms and requirements

"....." ..... 201....

Supervisor: .....
/signature/

Accepted for defence

"....."..... 201... .

Chairman of theses defence commission: .....
/name and signature/

TTU

School of Engineering

MSc THESIS TASK

Student: Vladislav Babuškin 163216MAHM

Study programme, main specialty: MAHM02/13 - Mechatronics

Supervisor(s): Professor Mart Tamre

Thesis topic:

(In English) Imitation of perceptions processes for an artificial life form.

(in Estonian) Tajuprotsessi imiteerimine kunstliku eluvormi poolt.

Thesis main objectives:

1. Create an artificial life form as an object of concept confirmation.
2. Design internal world space for the object.
3. Provide a feedback loop, in order to connect the external and internal worlds

Thesis tasks and time schedule:

No	Task description	Deadline
1.	Analysis of artificial replacement of the Sense organs.	20.09.2018
2.	Artificial life form mechanical design.	04.10.2018
3.	Artificial life form hardware design.	25.10.2018
4.	Artificial life form software design.	08.11.2018
5.	Environment simulation software design.	22.11.2018
6.	Communication establishment.	06.12.2018
7.	Proof of the concept.	20.12.2018
8.	Finalizing Master thesis.	03.01.2019

Language: …… **Deadline for submission of thesis:** "....."....201….a

Student: …….. ..... "....."....201….a

/signature/

Supervisor: …… ..... "....."....201….a

/signature/

CONTENTS

PREFACE

The topic for the master thesis is proposed by the author. The main aim of the project is to research a solution for perception simulation. The solution can based on the concept of a digital twin. I consider it as research about the possibilities in the robotics field. Digital twin is a dynamic virtual representation of a physical object or system across its lifecycle, using real-time data to enable understanding, learning, and reasoning. The problem of the digital twin concept is that the information goes only in one direction. However, received information from the real world should create or change the virtual environment and then send back new action plan according to the updated environment. For another hand, the technique of hardware-in-loop can be applied. HIL is a technique where real signals from a controller are connected to a test system that simulates reality, tricking the controller into thinking it is in the assembled product. Problems of usage the Hardware in loop technique in the frame of the current project that simulated reality is that these techniques do not exist, it needs to be built. In this way, two methods were combined in order to achieve the aim of the project. Shortly speaking, the system simulates in virtual reality not only the unit but also the environment. Then HIL applied and at the same time, the real unit duplicates everything that is happening in the virtual environment. The roots of the topic come from earlier author`s [1] bachelor paper, which one was dedicated to the concept of robust autonomous exploration mobile unit. However, besides that the current project presents another autonomous exploration unit, the master thesis topic has no more common features with earlier paper. In Bachelor paper there was covered the idea how to make exploration unit robust and to be able to restore basic functions by redirecting the remaining resource and disabling additional functionality, and the current topic cover option to extend interaction with the external environment and to improve perception for the unit about the current position and surrounding condition for himself. Also, the complexity of the unit was leveled up in order to match master degree level. The process of developing this type of units is covering multiple fields of engineering, including mechanical, electrical and software branches, in additional it is challenging to reach set goals. Because of the above-mentioned reasons, I found this topic challenging. I am thankful to Programme Director of Mechatronics - Mart Tamre for supervising me during the current project. Advice at the beginning of the project was helpful to me.

List of abbreviations and symbols

VR	Virtual Reality
AG	Augmented reality
DoF	Degree of Freedom
IMU	Inertial Measurement Unit
MPU	Motion Processing Unit
DMA	Direct memory access
ADC	Analog to Digital Converter
PWM	Pulse-Width modulation
IDK	Integrated Development Environments
I2C	Inter-Integrated Circuit
SDA	Serial Data Line
SCL	Serial Clock Line
MSB	Most Significant Bit
LSB	Least Significant Bit
ASCII	American Standard Code for Information Interchange
PCB	Printed Circuit Board
SLAM	Simultaneous localization and mapping
AI	Artificial intelligence
CPU	Central Processing unit
IK	Inverse Kinematics
DC	Direct Current
PCB	Printed Circuit Board
CPU	Central Processing Unit
SPU	Sub-Processing Unit
IDE	Integrated Development Environment
LDB	Local Data Base
SCS	Single-Chain Solver
RPS	Rotate-Plane Solver
UE4	Unreal Engine 4
HIL	Hardware-in-loop

INTRODUCTION

The name of the current paper formulated as, the Imitation of perceptions processes for an artificial life form. The combination of such words together sounds complicated and not self-explanatory enough for the wide public as it does; for instance, the combination of words as a self-driving car. Meanwhile, it also does not clarify what it is all about specifically enough to fit in in one paper and give the main idea of the current paper. Thereby, as a first step, it is necessary to implement more detail clarification about the concept, which stays behind this world combination and then explains why the certain way of realization for the current topic was done in a presented way, for what purpose and generally – "why".

The first point of interest is the word – perception. The internet research of the definition for this word will provide few most common definitions as the ability to see, hear, or become aware of something through the senses and the way in which something is regarded, understood or interpreted. Does something like this suit for a typical machine of the device?

For a moment, the concentration will be on the first definition of it: the ability to see, to hear or become aware of something through the senses. What exactly of the concept need to be aware of? In the frame of this paper the current definition covers the simulation, localization, and mapping (SLAM) of the environment where the control object currently located. In the presented paper the SLAM feature will be not considered as the main point, only the ability to draw the map of surrounding objects. Mainly the accent will be set on sense organs, simply speaking – the set of sensors to collect data input for SLAM feature.

The second definition stated as: the way in which something is regarded, understood, or interpreted. The "something", what needs to be understood or interpreted is own position in space or different uncertainties in the environment.

Why does the object need to percept? In known environments, for instance, production area, it is not necessary. Every movement can be hardly preprogramed and performed thousands of times with acceptable error. However, in the case of the unknown environment, every move will be done the first time and the case structure should be flexible enough to provide locomotion of the object on rough-terrain.

Above mentioned features can be applied for the exploration of autonomous units. Simply speaking, before the moment when the decision of the movement direction and the patch purpose can be made, firstly it is important to understand where we are, and from where we came from. In this way, the evolution process makes sense and makes it logical enough to consider this features in the frame of the presented paper. However, the direction of the movement more is the next step of the research and more related to artificial intelligence _algorithms (AI) and not will be covered in the frame of the current concept. Step by step.

The perception is peculiar to advance life forms like animals, humans. Peculiar to something, that has brains or developed neuron network. In any case, it is not related to the testing subject which is considered in the frame of the project. Only the attempt was made to simulate those complex processes, and this defines another word choice for a thesis topic.

The remaining part to define is - artificial life form. During the concept of shaping the author of this paper was inspired by the output of the Nature evolution process. The shape of the control object was borrowed from already existing live creatures, and because the project is dedicated to the recitation of something that already exists, the topic additionally defined with pre-fix of «artificial». It was possible to use the alternative name as autonomous exploration unit, but it would force additional accent on the purpose of the test object, but the main idea is about the perception of environment for single unit, and the purpose of this test object is a different topic.

1. THE GENERAL CONTROL OBJECT DESIGN ANALYSIS

1.1 Unit type

The shape of the control object dictated by one of the main purposes of the unit. At the endpoint, the machine should explore unknown environments and cover as much as a possible variation of obstacles. The most sufficient way to perform exploration would be from the air – the flying unit, it can cover big distances in a shorter period of time. However, it goes along with big energy consumption and eventually cannot stay in the air forever and requires landing and take-off spots. Also, it brings additional requirements to air density. Meanwhile, the ground vehicle would be more stable, robust, can carry the bigger amount of additional weight and also can be bigger by itself. In any case, each unit type has own advantages and disadvantages. The decision was made towards the ground type of the unit. In some meaning, it is easier to build a wheeled autonomous unit than stable and effective air apparatus.

1.2 Object shape

The control unit defined as a ground type. The evolution present many variation of the limb shape, which one capable to provide the object relocation by limb operation. However, all of option, presented by Nature, can be combined towhead and named as a leg (multiply hard long shape pieces connected by flexible joints in serial). In this way the artificial life form, presented in frame of current paper, should also have a leg shape limb.

Next question is the number of the limbs. Most common number can be named as a four. Potentially it is a good choice for the unit also. But this number is peculiar to well-developed systems with complete stag of different features as balance, coordination and control. By other words, the human has only two legs and is able to move, run and jump. However, the machines with same configurations require powerful CPU, agile drivers and short response time. Machine with four leg are more stable, but question of stability is still valid. In case of limb movement in groups by 2, the requirements for balancing is still strong. The requirement can be lowered in case of one by one limb locomotion. In this way the machine will be slow. Decision was made to create object with six legs. In this way the unit can perform movement by operating of two groups of limbs by three legs in each group and at the same time keep stable positioning by creating three point connection with the surface. Second positive advantage that the unit will able to grab some type of surfaces and it should increase possibility to overcome obstacles.

1.3 Perception channels

Normally, the live being have a six sensing systems. Every of which provide necessary data to operate in certain environment or survive in different situations. The most data stream is provided by the vision system (Visual data). Eyes allow to estimate distance to surrounding objects, live being position in environment. Next are sense of smell, hearing and taste. This three systems are not consider to be interesting in the frame of current project and will not be simulated by the test object. Except ears. This organ is interesting because of additional sense provided by inner ears – biological gyroscope, which one play assisting role in the process of perception of own positioning in the space. This feature is considered crucial for the test object and need to be simulated. By type define that it is ground walking unit. Last, but not least is tactile system. The tactile system also provided various data as temperature or pressure applied to surface. In case of contact with something, what potentially can harm the object, it will send signal in the shape in order to call fast reaction and avoid damage to the system.

1.4 Biological sense organs replacement

In this way would like to outline the three most important feedback systems for the test object in a frame of current research. The list contains the vision, tactile and balance segments. Those three systems need to be simulated in order to fulfill paper goals.

1.4.1 Vision segment

Human vision is a complicated feature. To design the artificial system, that can replace it, is a challenge by itself. Human vision contains the pair of two-dimensional images shifted in space between each other. Taking into account the distance between images and direction, with each eye is pointing, the final output can be upgraded with the depth of the image. Thereby, the initial design for the artificial system can repeat the same method and process data, which one is necessary to build geometry in a virtual environment. However, the end result will not be satisfying. Relying on an output of the data calculation based on two images will lead to unnecessary complicated additional algorithms and processes.

The reason is that human vision is a lie. If an average person, with two normal eyes, will close one eye, does he will continue to observe the same image with the correct depth? Yes. And the trick is that the human brain is able to complete missing data because of the trained brain. Generally, the advanced algorithms can handle this current process and fulfil the missing data, if they will assume that the lines are straight.

Even with two eyes, human vision can be a victim of visual illusion, which may lead to the wrong perception of the environment. It is happening because the amount of data, which the brain is constantly guessing is much more than just the calculation of the depth based on the difference between a pair of two-dimensional images.

So far the ideal replacement would be the system that combines visual stream from three different points in space and process along with a cloud of points, received through 3D laser scanning. For the current concept is enough to create a cloud of points around the test object, what can be considered as a base for the project evolution.

Thereby the initial set of sensors, that would be able to simulate machine vision is set as a 360-degree laser scanner and one ultrasonic sensor to define height between the center point of the test unit and top of the walking surface.

1.4.2 Tactile segment

The skin is the biggest organ of an average live creature. It is able to detect applied pressure to the specific point, temperature and a row of other specific parameters like potential difference or toxicity. On the first stage of evolution, the main parameter of interest is a pressure, and only in specific places, which supposed to be in contact with work surface according to design – The end point of every limb. Hereby, the machine is equipped with six pressure sensors and received data from the sensors can be used as an input for the point cloud. The location is calculated trough current body geometry and positioning in world coordinate system.

1.4.3 Self-balancing segment

Similar to the case with the vision, the brain use data stream from the inner ear and guessing missing data. Meanwhile, it also uses the data from the biological gyroscope to compensate for missing part of data from the other segments of fells. The artificial replacement for this organ already exists in the shape of plug and play. The data from the gyroscope is usable in case of static measurements. In the case of dynamic situations, the data stream can be adjusted with the data from the accelerometer, magnetometer, and different data filters. Shortly speaking, this segment is straight and does not require reinventing of the wheel. More details can be found in chapters, related to this sensor.

2. MECHANICAL DESIGN

2.1 Decision to make own design

Nowadays the market provides a wide variation of the chassis, which are possible to use in the frame of the current project. In order to prove the concept in a complete way, the chassis should satisfy a self-made list of requirements. However, every specific model did not meet to one or few requirements, highlighted by the author. Below are listed important requirements:

- The limbs should be flexible enough in order to operate on top of the not fully horizontal surfaces or grab the protrusions on the surface.
- The platform should be capable with different body modification in order to easily install different types of sensors.
- The chassis should be able to carry own weight, taking in to account the weight of the battery.
- Design should include an option to provide necessary feedback from the different elements.

The last point in the list is more about feedback from the actuators as the servo-motors. Normally the servo-motors have a closed loop control system. Models with the feedback to the microcontrollers also exist on the market. However, the necessity to find the chassis, which one will meet all the requirements and line up a specific model of the servo-motor will be expensive or the number of the modifications to the chassis will be big enough to lose the point of use out of the shelf solution.

The decision was made to design and build chassis for the project on my own. It will allow to create the frame, which will satisfy to all requirements. In addition, it is and challenging. The outcome is presented in Fig. 2.

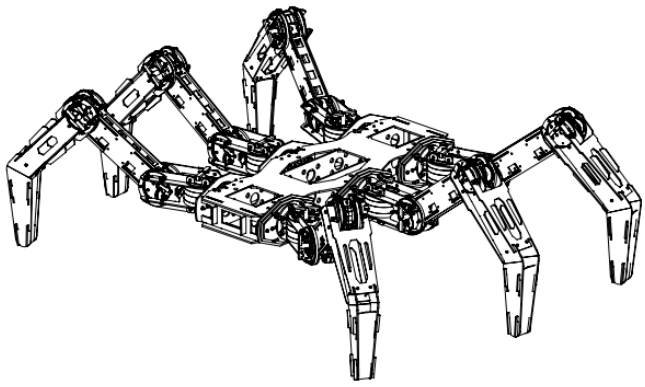

Figure 2. Omicron chassis

2.2 Common Geometry

Omicron chassis contain six limbs with four joints in each leg. In a star position, the outer diameter of the machine is two meters, main body length is 70 centimeters and height is 15 centimeters. The design is symmetrical along two perpendicular in two axes X and Y. The root point of middle limb origins (shoulders) is shifted to side in order to provide additional width to the step length. For future development iteration, it is desirable to increase shift in the middle section.

The end segment of the limb can be seen in Fig. 2.1, and it has a broken profile in order to optimize the locomotion processes and improve grabbing possibilities of the test object. Moreover, the end of the limb with an angled profile eliminates the necessity of additional joint, and also simplifies animations for the movement and complexity of the hardware.

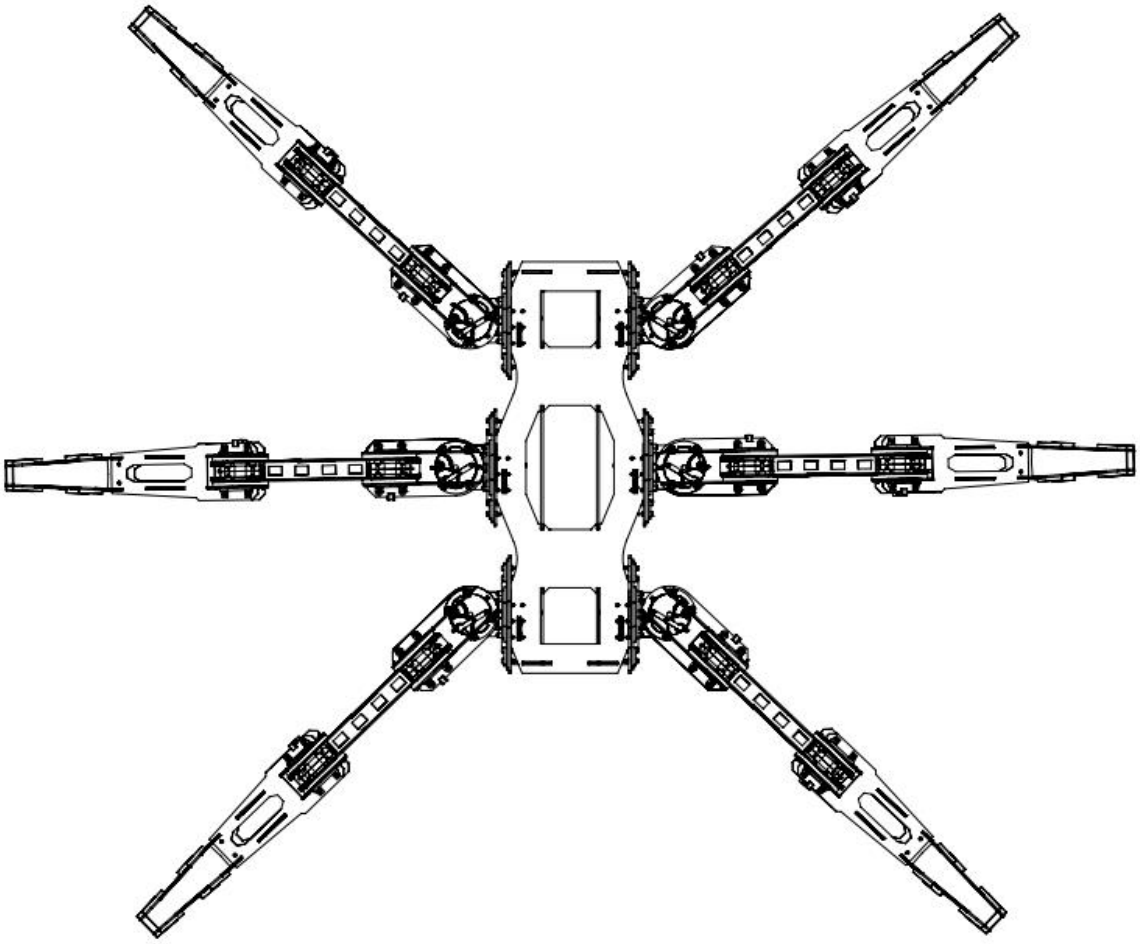

Figure 2.2 Omicron chassis geometry

2.2.1 Chassis limb DoF

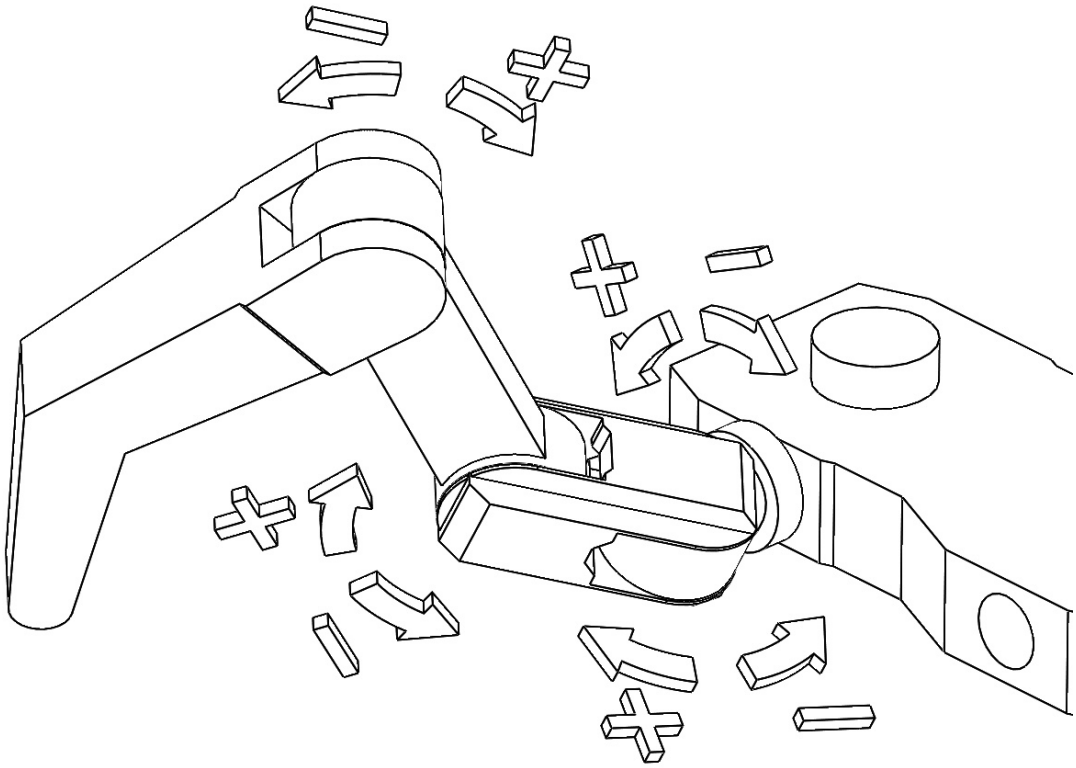

Figure 2.2.1 Object limb revolute geometry

The Omicron limb can move in various ways. The object leg use mode of the movement known as a revolute geometry. The Fig. 2.2.1 shows a mechanical arm capable of moving in three dimensions using revolute geometry. The entire assembly can rotate almost full circle (270 degrees) at the root joint (J0). The primary joint (J1) by itself is able to provide a turn for 270 degrees, same as a root joint. However, the physical rotation is limited by the machine body and lower the number down to 180 degrees. The first of the secondary joints (J2), taking into account limitations as, a maximum possible feedback element rotation and physical arm geometry of the prototype, the available rotation is 190 degree. Until this, the listed above joints can provide equal positive and negative rotation. Starting from zero position, as shown in Fig. 2.1.2, and then half of the mentioned numbers goes in one (+) and half in another (-) way. This rule is not applied to the last part of the limb, because of special profile shape. The last joint (J3) can provide 120-degree rotation in the negative direction (adduction or flexion) and 90 degrees in the positive direction (abduction or extension). Total rotation resolution for the last joint is 210 degree. Listed data of every joint limit is used later in order to create a set of animation for the test unit.

2.2.2 Limb work area

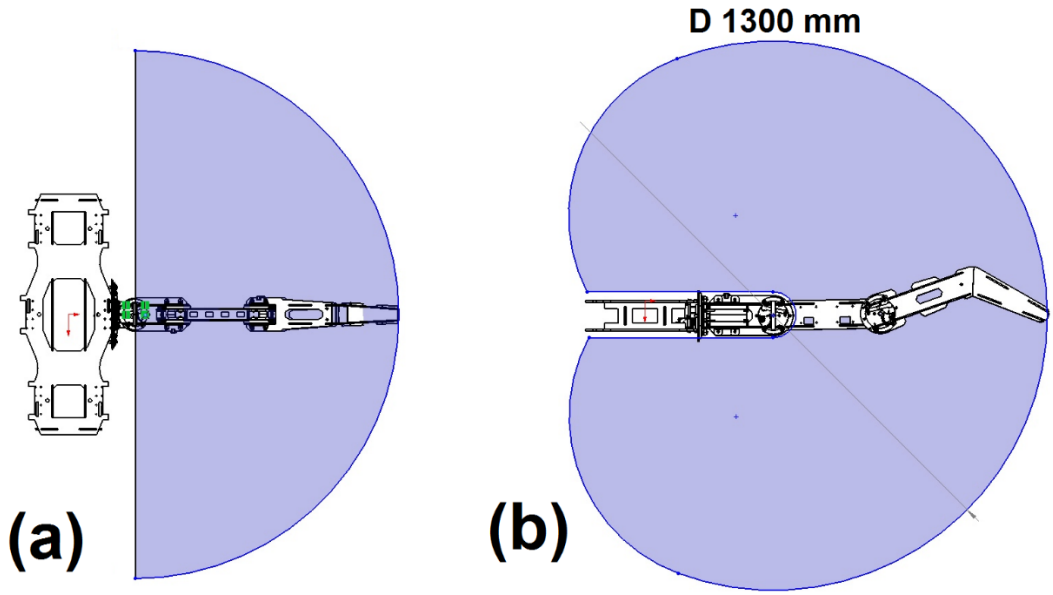

Figure 2.2.2 Limb work area

In Fig 2.2.2 the Omicron limb set in zero position. By blue area outlined the arm workspace. The top view (a) represents a horizontal cut of the work area and the front view (a) vertical cut of the work area. Both views, (a) and (b), are drawn without taking into account that the root joint (J0) can provide rotation with 270 degrees. With the rotation of the root joint the work area three dimensional shape converts into an almost spherical shaped space. Design of every limb is fully identical, what means that the work area shape is also similar, and because of the origins of the root joints the work areas are interfering between each other. Due to this fact, the general design forms areas with possible collisions. In order to avoid limb collisions, the operation of every arm is coordinated and synchronized.

2.3 Joints

The unit limb contains four joints, which are custom made servo-motors integrated in to machine frame. The custom solution is able to provide necessary features for this project. First of them is open-loop feedback, connected not only to the internal driver, but also to main machine CPU and eventually the data from position sensors going through the machine CPU, communication segment and goes to the virtual environment space to use for Inverse Kinematics (IK).The main characteristics of every joint is identical in meaning of torque, gear ratio and driver unit (DC motor).

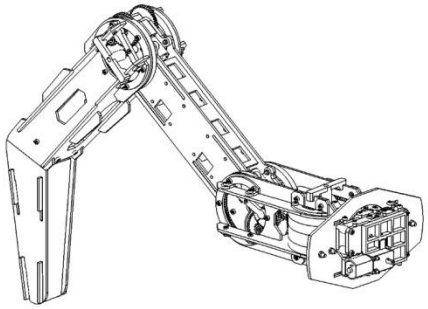

Figure 2.3 Machine Limb

2.3.1 Root Joint

The root joint (J0) is built into the main. The housing of J0 is serving multiple purposes. At the first place, it rearranges all the gears inside and forms of the secondary gearbox, but it also serves as a body structural element and provides chassis with additional reinforcement.

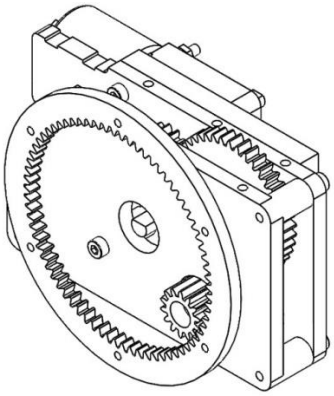

Figure 2.3.1 Root Joint (J0)

2.3.2 Primary Joint

The primary joint, unlike the root joint, have more specific shape. That not only sustain the positioning of all internal components, but it also serve as a shaft for the next segment in machine limb.

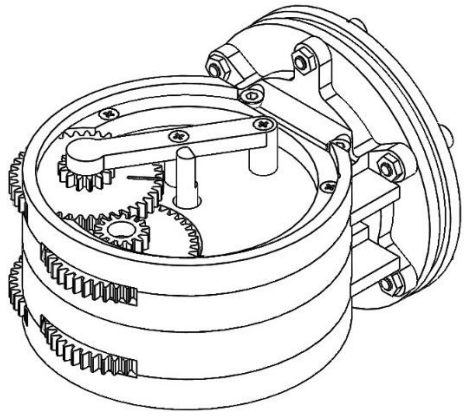

Figure 2.3.2 Primary joint (J1)

2.3.3 Secondary Joint

For the third and fourth joints in the arm (J2 and J3) are based on identical custom servo-motors. Meanwhile, this joint (device/segment) is the most light weight and complicated design. The servo-motor housing is not only built in to it, but also appears to be part of it. Additional restriction to the shape of these segments applied because of external elements of the limb, which are moving around it.

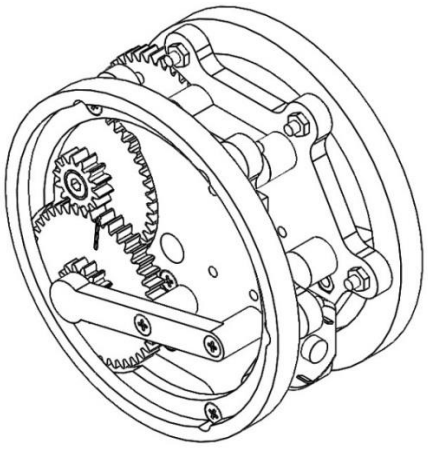

Figure 2.3.3 Secondary joint (J2 and J3)

2.3.4 Joint Design

The internal common design for all of the joints is more or less same. The difference is in the housing shape, in order to provide mounting option inside of leg frame and duplication of the secondary gearbox line. To provide an equal load for both sides, the secondary line of gears is symmetrically duplicated on both sides. Exploded view of the joint design shown at Fig. 2.3.4.

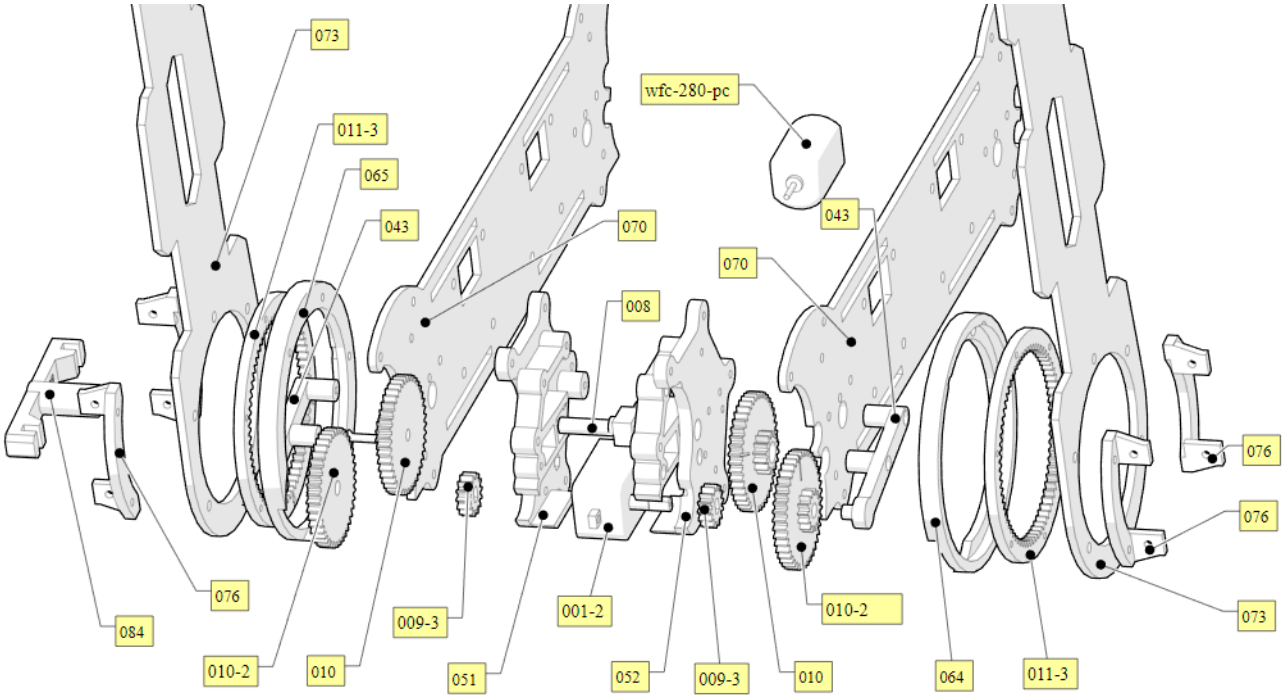

Figure 2.3.4 Internal joint design (J3 and J4)

The core of custom servo-motors is the primary gearbox (001-2), DC motor (wfc-280-pc) and the potentiometer (008), which one serves as a feedback element. Those three components are fixed together inside of the first layer of housing (051 and 052). On top of the housing are mounted lines of gears from both sides (009-3,010 and 010-2). Then goes frame part of previous leg segment (070) and on top of them is mounted stabilizations for last gear (043) with the external shaft fixator (064). All the remaining parts belong to the next segment of the limb (in particular case – end segment of the arm). The last gear (011-3) is mounted to the frame of the next segment (073) and a fixed trough along with the brackets (076). The bracket also fixing together two parts of the end leg segment. On top of the last segment (do not show on the image) is plugged the coordinator (084) and it rotates the potentiometer (008) according to the angle shift between two leg segments.

2.4 Driver unit

For the object was chosen out of the shelf module, which contains built-in gearbox and installed DC motor model 130 [5]. According to datasheet, current model have rated load 0,09 [N*m] torque and 9000 [rpm]. On a preparation phase was made a prototype of the machine limb, which had the original built-in DC motor in primary gearbox and gear ration in joint 760 to 1. As an outcome, the joint should have the angular speed to be somewhere around 71 degrees per second without load and torque equal to 0,74 [N*m]. Named number does not take into account load condition, power loss because of the friction between the components and efficiency of the gearbox transmission. The aim is to have around 40 [N*m]. Anyway, the output is not strong enough to provide movement capability for the test unit. For the first prototype, the gear ratio was changed to 2180 to 1 and DC motor was replaced by 280 [6] models. The specification of the 280 models has rated load as 1,47 [N*m] and 7500 [rpm] with the 6[V] voltage rate. The latest join design has a gear ratio 2280 to 1 from the DC motor to arm segment. Thereby, the torque output equals 33,5 [N*m], and angular speed to 19 degrees per second. However, the motor drivers are supplied with the 12 volts and the final values are bigger, that should compensate power loss in gear transmission because of friction, non-ideal manufacturing processes, and other reasons.

3. HARDWARE DESIGN.

3.1 The Omicron hardware

So far the Omicron has 12 dual H-bridge DC motor controllers, 24 potentiometers, inertial measurements unit, sonar, laser scanner, and a communication module. All together those components require a decent amount of calculation power. Few of them, like IMU, Sonar and laser sensor, are highly relying on the timing between calculation steps. This issue may be solved by a powerful on-board computer, which one can handle not only all the basic calculation but also different graphical tasks. In order to avoid decent investment for internal hardware, the biggest part of the processing was transferred to a desktop computer and the basic processes remain in the test unit. Current decision allow proving the concept with the smallest financial investments. However, in order to optimize calculation processes, even more, is necessary to make them parallel (multicore processor). Solution to the above-mentioned question is to separate handling of the end components of the Omicron between multiple controllers. The general hardware schematic is presented in Fig. 3.1.

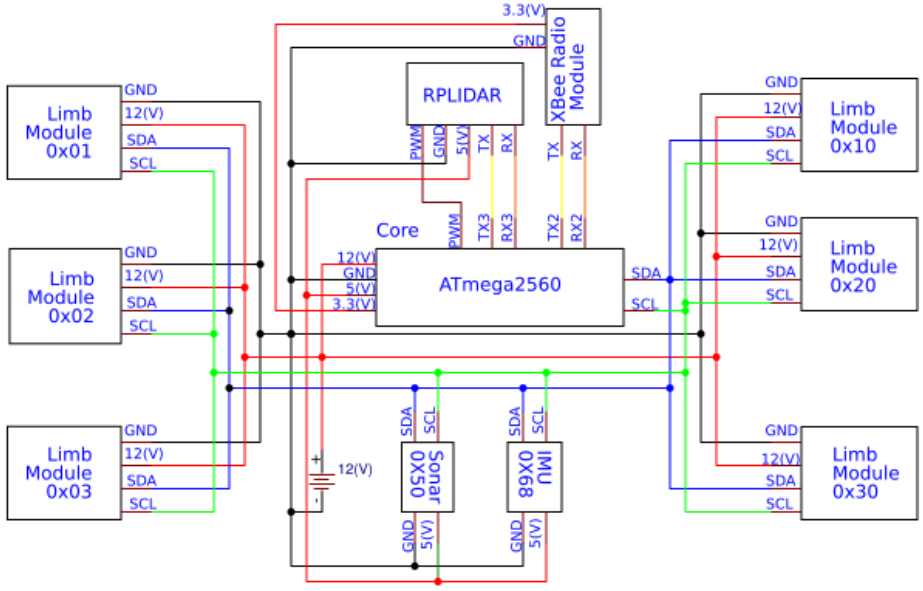

Figure 3.1 Omicron general hardware schematic

In total, the design contains seven 8-bit AVR microcontrollers ATmega328, for each the limb (SPU) and one for sonar segment. Another 8-bit AVR microcontroller as ATmega2560 is acting as a core (CPU) of the Omicron. The last two modules, the communication, and laser scanning segment, also are connected directly to the Core. By current solution the wire-loom is simplified (less wires).

3.2 Core Hardware

The Core is connected with the limb and sonar controller through the I2C bus. The communication and laser scanner is connected through the serial bus with the RS-232 protocol. Beside one of the PWM pin of the ATmega2560 (used to control the rotation speed of the PRLidar), not a single digital or analog port (GPIO) is used. The core is supplied with 12 volts and with internal step-down voltage regulator of the Core is supplying connected to its elements with five and three volts. The limb controllers are not supplied from the core.

3.3 Limb hardware

The limb hardware part represent independent segment and connected to the system only with voltage supply and communication bus. Similar, as the plug-in for desktop computers as, for instance, mouse or keyboard. Originally it was done in order to simplify maintenance and make the limbs to be easily detachable. It also would make the unit transportation simpler than the transportation of the assembled machine. However, detachment of the limb require complete disassembly of main body (gears of the root joint are locking linear movement). The complete schematic of the Limb presented in Fig. 3.2. It contains four DC motors, one in every joint, two dual H-bridge motor drivers, four potentiometers as a feedback and one pressure sensor.

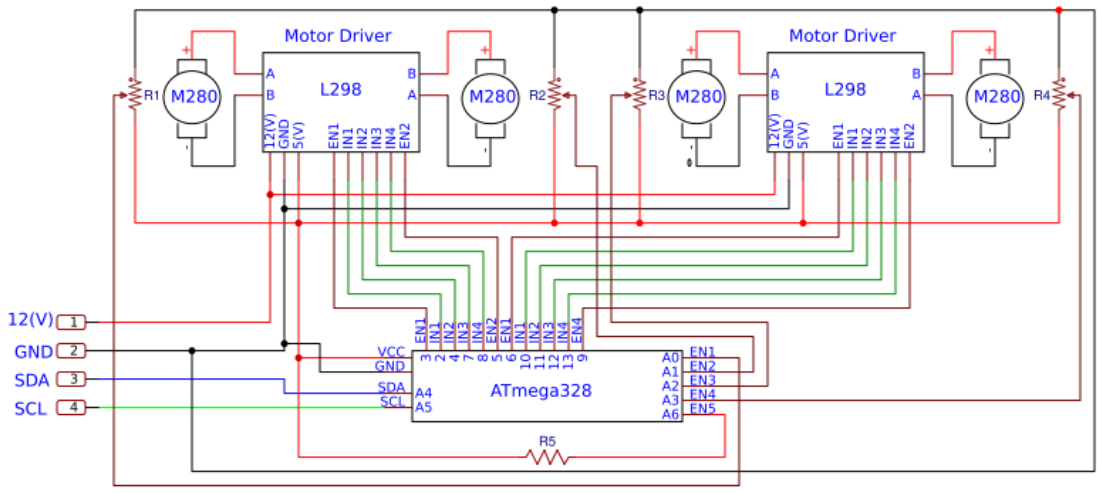

Figure 3.2 Omicron limb hardware schematic

3.3.1 Motor Driver

The requirements to the drive, inside of the joint, are that it should rotate in a different direction with controllable rotation speed. The most common solution that meets to above-mentioned requirements is the H-Bridge [1]. For current project is chosen the H-Bridge driver based on L298. The model of the motor driver is L298N 25W Dual Channel H Bridge DC Stepper Motor L298N Drive Controller Board Module 5 V 2 A. This model can handle load up to two amperes. According to DC motor datasheet, the maximum load is 3.2 Amperes. The actual maximum registered load, per one motor, in the current design, was registered as a 1,7 A during the unit walking process. The control element based on MC33886 driver can handle up to 5 Amperes. However, the decision was made in favor of the first option because of additional list of features, which PCB assembly has. The additional features are the built-in diode protection segment, massive heat conductor and step-down voltage regulator. Besides that, the PCB assemblies with MC33886 cost two times more.

3.3.2 Limb Support Processing Unit

As the Sub-Processing Unit was chosen PCB assembly based on ATmega328 microcontroller. The name of the assembly with ATmega328 is Arduino Pro Mini. The main reason for the choice is the general dimension of the development board. The idea was to fit all the necessary hardware for the limb inside of one of the arm segment. Besides the size, the amount of pins is enough to provide management of all end components of the limb without additional elements as, shift registers, multiple channels analog or PWM drivers. The assembly requires 8 digital, 4 PWM outputs and 5 analog inputs. The Pro Mini has 7 analog inputs (two of them are used for I2C interface, 5 of them in use by Omicron) and 14 digital GPIO ports (first two of them meant for serial communication), 12 of which are in use by limb hardware. In this way, can say that the Arduino Pro Mini suits well into the frame of the current project.

3.4 Sensors hardware

The machine has a row of different sensors. By the type of feedback, all those sensors can be separated into two groups. The data from both groups are used in order to build cloud points and coordinate locomotion. The first group contains the laser scanner, sonar and pressure sensors. The main aim of this group is to provide data of angle and distance to every detected point. The second group contains the IMU and potentiometers. Data from the last group mainly are angles in the joints and main body tilting. This data is used to transfer data from the first group from the local coordinate system to the world coordinate system.

3.4.1 RPLIDAR

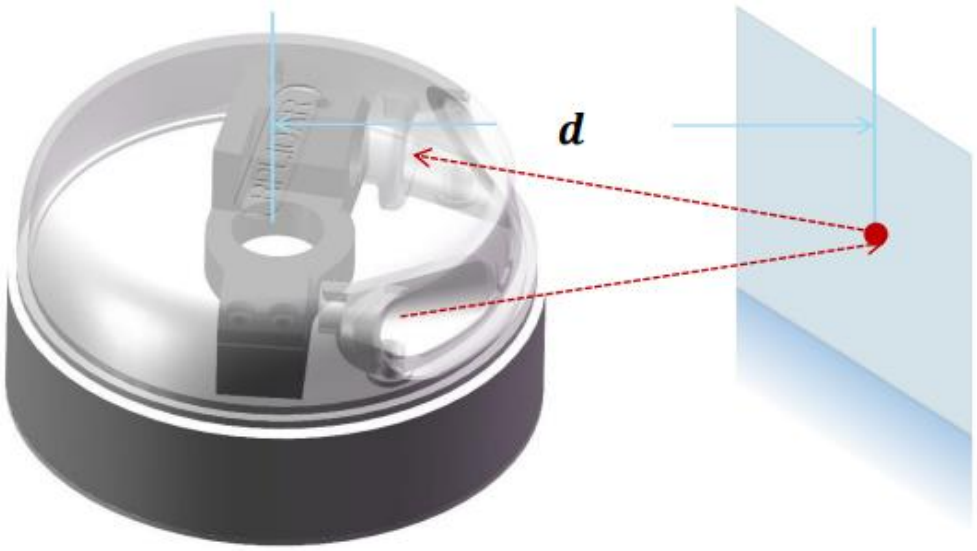

Figure 3.4.1 The RPLIDAR Working Schematic [7]

As the laser scanner was chosen RPLIDAR A2M8 [7]. It is a low cost and low power 360 degrees 2D laser scanner solution developed by SLAMTEC. It can take up to 8000 samples of laser ranging per second with high rotation speed. The data from scanner provide information as angle and distance from the sensor center point. The initial data allows creating two-dimensional map of the surrounding. The third dimension is created artificially by moving it in space. The sensor is mounted on the main body and moves along with it. Taking into account that the machine is a mobile exploration platform, all the necessary requirements are meet to build a three-dimensional cloud of points.

3.4.2 Sonar

As the sonar is used ultrasonic sensor HC-SR04. Low-cost solution to provide information related to the distance between the central point of the device and surface under it. The Omicron van move whit tilted main body is. Because of the tilt, the data from the sonar is not valid. However, the distance can be adjusted by the data from the IMU. There are two cases when the data need to be adjusted. The case scenario and decision-related to the adjusting and calculation is delegated to the virtual environment. The schematics of the sonar segment presented in Fig. 3.4.2.

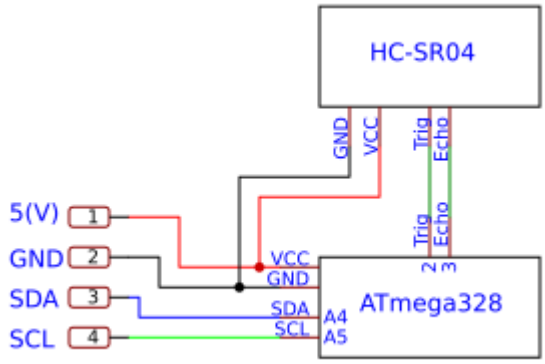

Figure 3.4.2 Omicron sonar segment schematic

The sonar is not connected directly to the Core because of the readings taking method. The sonar first sends the signal, then waits until the echo will return. The delay between those two steps can badly effect on the calculation processes of the Core. In order to make this sensor independent, all the calculations and reading taking processes transferred to separate SPU.

3.4.4 Pressure sensor

Every limb end segment equipped with a dynamic pressure sensor in order to detect contact pressure created between the leg and the work surface. The value received from the pressure sensor is used only to detect contact by the threshold and define an additional point in the point cloud. Potentially the same data can use to determine stress created in the limb and stop the motion in order to secure mechanical parts from the overload and hardware to avoid high current in driver units. It would be good to implement because the ampere rating of motors is higher than the ampere rating of the control unit. This feature will be set into the backlog and implemented by chance.

3.4.5 Inertial Measurement Unit

The purpose of the inertial measurement unit, in the frame of the current project, is to track the tilting of the unit chassis. Because of the relatively slow and smooth motion, the digital gyroscope will be enough to meet the set requirements. However, as the MPU was made a decision to use not only gyroscope but also accelerometer. It will provide an option to apply different types of data filtering as, for instance, the Magwick filter or others, and improve the quality of readings. The specific model of IMU is MPU-9150 [8]. This chip communicates through I2C bus and has built-in accelerometer and gyroscope. Moreover, it also has a built-in magnetometer (compass), but for some reason the magnetometer located under different I2C address. Theoretically, by using a combination of data from those three sensors, is possible to track shift in space taking in to account the position (accelerometer), change of the direction (gyroscope), and the travelled distance can be calculated through the acceleration and velocity.

3.4.6 Potentiometers

The feedback in every joint is the 10 000 [Ohm] potentiometer with the 270-degree rotation. Disadvantages of this solution are the accuracy of the readings. During the time the resistive contact surface inside of potentiometers is losing own characteristics and provide with invalid data. The second disadvantage is the rotation possibility. The root joints do not have physical limitations and can rotate 360 degrees. Hard to say how many of potentiometers was destroyed during the prototyping and how many of them are damaged in the current machine. The advantages are that the potentiometers are absolute sensors, they are giving direct absolute value related to actual position without necessarily to search zero position and calculate angle according to the number of pulses, as in the case of encoders. It requires fewer calculations, simplifies the general design and decreases the cost of the single joint.

3.5 Communication hardware

In an ideal condition, all of the processes should run on an internal hard drive of the unit. However, on the current moment, the heavy lifting transferred to the desktop computer (Host) and all the necessary data is transferred through the serial port. In order to make device wireless, the physical connection is interrupted by the additional element in the chain, which allows sending and receiving data without the hard connection. Among the many options was the chosen method of radio communication as an XBee S1 module [9]. According to the datasheet, the module can transfer up to 250 kbps of data. The more detail overview of the data traffic is presented in Chapter 4.2.

The desired level of communication is to send 8 000 points from the laser sensor and exchange with matrices 20 times per second. With rough calculation, it equals 110 000 bytes of data or 884736 bits. By one XBee package can be transmitted 2 bytes or 16 bites, but the package size is 21 byte.

Along with the 16 bytes will be also sent 152 bits of package information. It will take 55297 packages to transfer all messages meant for one second or 9289728 bits. By other words, the transfer rate of 9,3 Mbps of data. For the next test iteration, the communication needs to be replaced by the G4 module.

3.6 Battery

While waking the one Omicron Limb consume 2,7 A without load. In total, the Omicron has six limbs and consumption goes up to 16,2 A per hour. The Omicron battery assembled with A 3,6V Lithium Ion battery cells. In order to reach the necessary level one element should contain four cells connected in serial. In this way, the voltage rate is 14,4 V. All of the components are capable of current voltage level because of the built-in voltage step down in the Core and motor drivers.

One row of Lithium-Ion battery has 3000 mAh capacity. One hour of work will drain a bit less than 6 rows of batterers (18 Ah). In total, the number of cells reached 18. One cell weight IS 48 g. Thereby, one hour of working process for the device will require an additional 864 g.

By other words, in case if the aim for the Omicron is to walk 10 hours straight, it should be equipped with 8,6 kg battery assembly.

4. CONTROL UNIT PROGRAM

4.1 Development Environment

All the programs for the Control Unit are written on C++ language with Arduino IDE. The reason for this development environment selection is due to the following advantages:

- Well-developed software environment. Stable, robust, intuitive and easy to use.
- Open source with a large community. Has a wide selection of libraries that covers almost every commonly uses devices in robotic hobby segment.
- Wide variation out of the shelf solutions of different hardware segments.
- Capable with most of the operating systems.

Shortly speaking, in case if I have a simple idea or the concept, usage of the well-developed base will save the time, resources and nerves. Instead of fighting with complicated IDE and Software Development Environments, if it is not the point of the interest (sometimes it is), more preferably to concentrate more on the concept by itself. When the concept is proven, then it is a good moment to evaluate the weak point of the concept/project and switch to more advanced development environments.

4.2 Limb Program

The Limb SPU provides control over four custom made servo-motors and provides open loop feedback. The program presented in Fig 4.2 and Appendix 1. The control of the servo-motors realized with the PID approach [2].

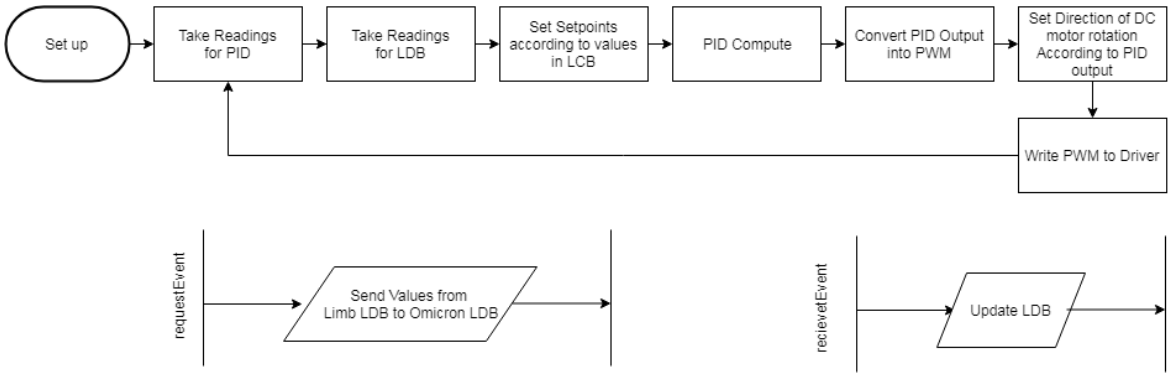

Figure 4.2 Limb programm block scheme

4.3 Sensor program

Machine equipped with three sensors. The ultrasonic sonar, laser scanner RPLIDAR M2A8 [22] and IMU-9150 [23]. The sonar deals with one part of the artificial perception concept, related to the understanding of distance between the surface, on top of which the unit is located, and the main body. This information allows adjusting walking animation. The RPLIDAR acts like eyes but does not receive information about the object, only the distance between the object and the unit. The last, but not least, is IMU sensor. IMU allows being aware of the main body tilting relative to a work surface, or the rotator in word coordinate system. The common between those sensors is that the initially the machine is not aware of the location of the sensors on the main body and data is transferred to the virtual environment without origin information. The location is hardly fixed and set in VR.

4.2.1 Sonar program

The main feature of the sonar is to measure distance. The code segment of the program, which one is taking readings, was taken from the library made by Martin Sosic [10]. The complete program for the sonar is presented in Appendix 2. The Sonar Program Flowchart is presented at Chart 4.2.1.

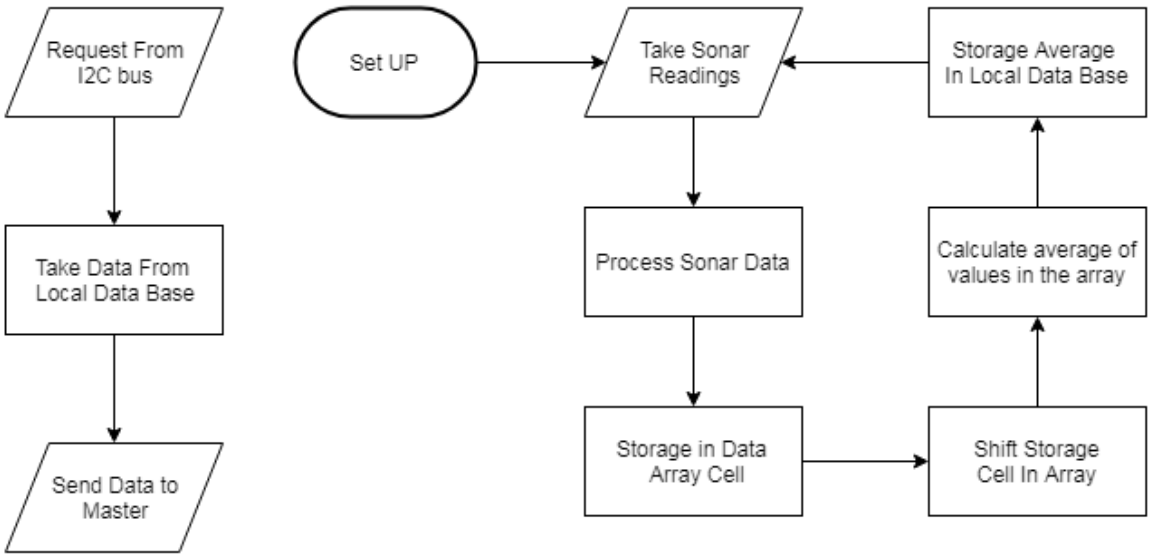

Chart 4.2.1 Sonar Program

Every new loop the program is taking one reading and placed it in one of 20 cells of the array. At the end of the loop, the algorithm calculates the average of all the values in array and storage it as a separate integer in LDB. In this way, the average contains data about the 20 last readings, not the 20 new separate readings for every loop. In this way, the value changes slowly up to latest reading and provide smooth value change, which can be applied in animation and not cause instant position change.

4.2.2 RPLIDAR program

The RPLIDAR is stand-alone sensor with own SPU with a serial communication. The communication is handled by library and example program [4] provided by RoboPeak.

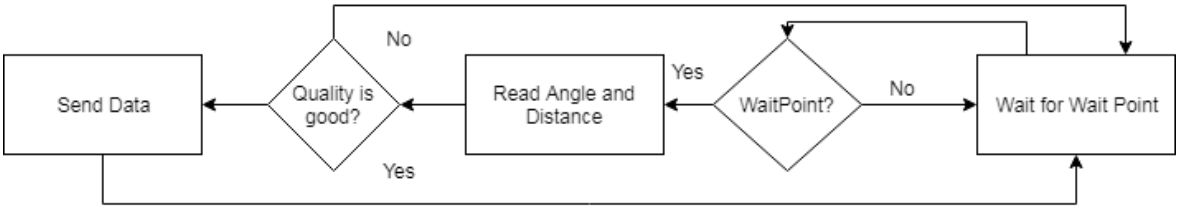

Chart 4.2.2 RPLidar Segment

The example program for Arduino IDE provided by RoboPeak was modified and adjusted in order to fit in a frame of Omicron core program. In the case of sensor malfunction, the original control program stops the rotation of the sensor vision part and then tries to restore the connection with it. In case of success, the control program restores rotation. A current feature was disabled due to the timing of the process. The Core hardware is not able to provide a stable operating process and smooth data flow. This can call connection to restore procedure few time per minute. In order to improve performance, in the frame of current hardware, the motor of the sensor never stops and at the moment when the program will restore communication the sensor is already ready to use. The Control Unit restore data stream without time delay until the sensor will reach the necessary rotation speed. The second modification is related to the data format. The distance is measured in millimeters. To transfer fewer bytes at the time the value is transferred as a centimeter in integer format. The VR representation of the word will be less accurate, but accurate enough for the first stages of the experiments. The third modification is data filtering. Some of the readings come empty. The angle is specified but the distance equals to zero. Because of the chassis geometry and sensor location, the zero distance measurements are not valid. To avoid spamming in communication channel all values, which are filtered out, does not participate in the data transmission process.

4.2.3 Inertial Measurement Unit program

The program was written by using basic example code by Kris Winer [11]. The common problem for the digital accelerometer is noise and the drifting effect for the gyroscope. In order to filter and smooth the raw IMU readings was applied the Madgwick filtering and data fusion of gyroscope and accelerometer. Also, final values are adjusted according to magnetometer readings. The program output is the quaternion. Calculation Pitch Yaw and Roll made base on quaternion. There is a trade-off in the beta parameter between accuracy and response speed. The code presented in Appendix 3, line 365 – 465. The segment of the code, related to Madgwick quaternion update, is located between 600 – 720 lines. There is the row of reasons why the Omicron IMU program uses the quaternions. The first one is that they are much more space efficient to store than rotation matrices it is four floats rather than sixteen. In addition, they are much easier to interpolate than Euler angle rotations, the spherical interpolation or normalized linear interpolation. Also, quaternions avoid gimbal lock.

4.4 Core software

The Core program mainly designs to collect perform data management between. The core connects one by one to each limb and collecting data from LDB at the limb SPU. When all data is collected the Core sends it to the Host along with sonar and IMU readings and laser scanner readings. At the host data goes to the simulation of the environment. The code flow presented at the Chart 4.3.

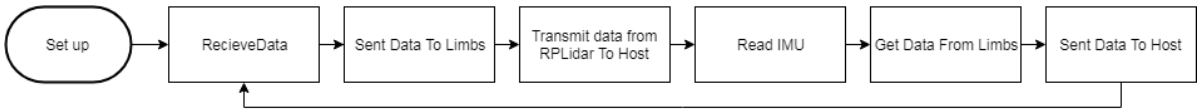

Chart 4.4 The Core Program

4.3.1 Collect data

The device has several sources of necessary data. The values of the Limb potentiometers and sonar distance is collected through I2C by getI2CData() function. The Core sending request to the limb SPU to transfer 5 specific bytes by the command Wire.requestFrom(01, 5), where 01 is the address and 5 is the number of requested bytes. After request the Core reads incoming bytes and store them at the Core LBD - for(int i=0;i<5;i++) { int c = Wire.read(); FromLimb01[i]= c;}. The same process repeats for every Limb and sonar. Request format is identical for every arm and sonar. The difference is only in the number of requested bytes and, the address and the storage cell at LDB.

The core communicates with the IMU in a similar way with the limb. However, the Limb values were lowered from 1024 to 255 in order to transfer them as one byte. The IMU values are 16 – bit values. In this way the core first request array of six bytes by command readBytes(MPU9150_ADDRESS, ACCEL_XOUT_H, 6, &rawData[0]), and then assembling three 16-bit values - destination[0] = ((int16_t)rawData[0] << 8) | rawData[1] ; . When the data is collected the Coro perform row of basic calculation, Madgwick and quaternion calculation. When the secondary calculation is done, the data storage in the LDB cell until another of data exchange session.

4.3.2 Send data

Machine needs to transfer row of numbers. Meanwhile, the Host needs to understand where those numbers belongs to. This question was solved by creating packages. Each data array, for instance, from the limb or the set of IMU and sonar data, is send out with letters at the beginning of the array and one common letter for all of them to mart the end of Array. The segment of the code is shown below:

```
Serial3.print("LF"); Serial3.print(" ");
   for(int i=0;i<5;i++) {Serial3.print(FromLimb10[i]); Serial3.print(" ");
                         }Serial3.println(" X");
```

Code Segment 4.3.2. Data package

Every limb has characters combination as LF. The first Letter stands for side. “L” for left and “R”- for right. The second letter marks the order of the arm. “F” means front, “M” for middle and “B” for the back. When all the data is send, the core is transmitting letter “X”. It serve for additional checking at the host side. More about the checking can be found in Chapter 5.2.1. The full code can be found in Appendix 3, 530-560 lines.

4.3.3 Receive data

Received data also structured in a packages. In order to simplify program at the Omicron side, instead of two letters the every array marked only by one letter at the start and at the end. In this way the program do not need to collect string out of characters and compare in a frame of the case structure. Less calculation required in a sacrifice of data transferring safety.

```
if(Serial3.available()){

char check = 0;
                        AdHstLb = Serial3.read();
switch(AdHstLb){
  case 'A':
      for(int i=0;i<4;i++) {LimbBuff[i]=Serial3.read();}
        check = Serial3.read();
        if(check=='G'){
            for(int i=0;i<4;i++) { ToLimb01[i] = LimbBuff[i];}
                      check =0;
                      } else {
                          cleanBuff();
                          check = 0;
                        }
  break;
```

Code Segment 4.3.3 detect and read package.

4.3.4 Distribute data

The only data from the Host, which goes to The Omicron, is array of set points to every limb. As soon as the Unit receives data from the host, it stores data on the Core LDB and later distribute to The Limb. The Data distribution occurs trough the I2C protocol, the segment of the program, which one is responsible for distribution, is shown at Code Segment 4.3.4.

```
Wire.beginTransmission(01); // transmit to device #01
Wire.write(ToLimb01, 4); // sends Setpoint data
Wire.endTransmission(); // stop transmitting
```

Code Segment 4.3.4 Send data from the Core to the Limb

5. PERCEPTION SIMULATION SOFTWARE

5.1 Communication

In order to bind the body and mind between each other, or the imaginary world (simulation), and the real world, it is necessary to provide an option for the data exchange. Inside of living beings, this feature provided by Nervous System. In the case of Omicron, the communication between segments implemented with I2C protocol, something as a Peripheral Nervous System. However, all collected data need to reach the simulation on the host (Brain). The communication between chassis and the host occurs wirelessly, based on the RS232 protocol, it can be considered as a Central Nervous System. In Chapter 4 were considered data transmitting and receiving algorithms on the Omicron side, and the communication on the Host side implemented in a similar way.

5.1.1 Communication set up

With additional UE4Duine plugin, created by Rodrigo Villani [12], for Unreal Engine 4, the serial ports can be reached inside of development environment as an object. The only parameters, which need to be set is the com port name and baud rate. The communication set up flow presented in Fig 5.1.1

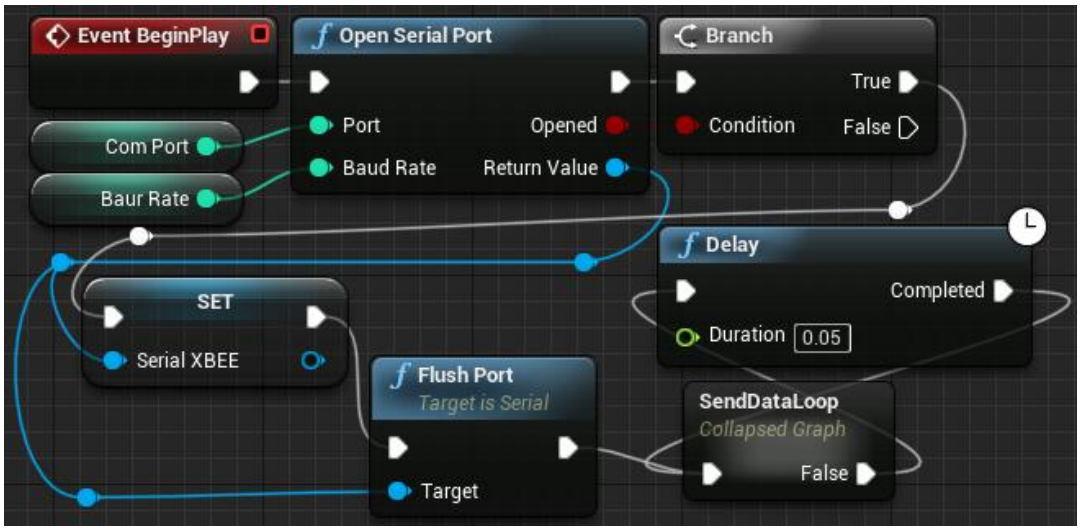

Figure 5.1.1 Communication set up.

When serial communication is established the program goes into data transferring loop. Similar to the algorithm on the Omicron side, the data is sent with custom packages. The data storage in LDB and data transferring processes occurs independently from each other.

5.1.2 Data storage

During the Simulation process all the necessary data are storage in LDB. The algorithm presented shown in Chart 5.1.2. The simulation environment does not have virtual servo-motors and not able to present specific data for servo-motor control. In order to get necessary control input for Omicron the additional data converting is needed. The algorithm presented shown in Fig 5.1.2.

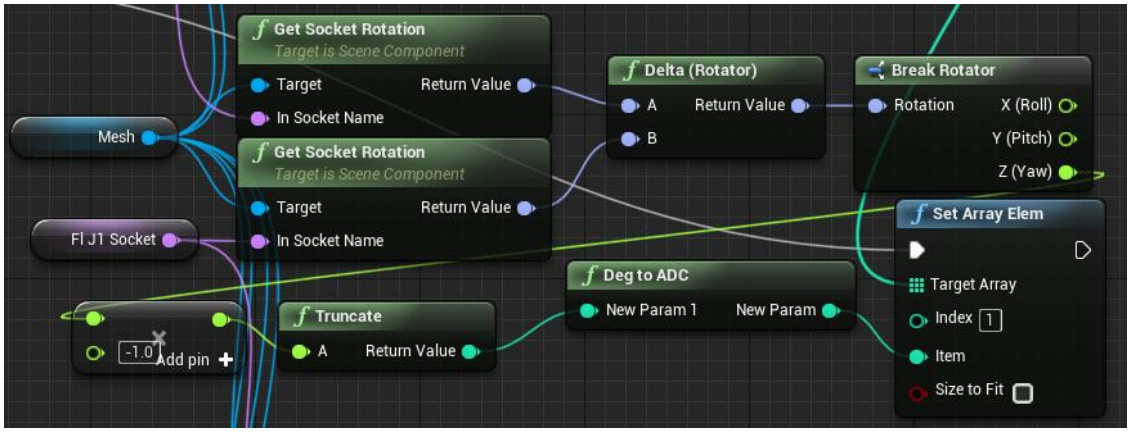

Figure 5.1.2 Data Storage in LDB.

Every joint of the skeleton is equipped with the socket. The program reads rotation difference between the previous and current socket in skeleton hierarchy and recalculating angle to PWM value. Then the PWM value is going to LDB. The data for the servo-motors are updating every calculation cycle. More details about sockets can be found in Chapter 5.2.7.

5.1.3 Data transmitting

The Data Transferring segment of the program is located outside of the main program loop and. The main reason is the frequency of the main program calculation cycle. The cycle is one time per simulation frame. By other words, in case if data transferring segment will be located in the main loop, the program will send angle matrix for the Omicron 60 - 120 times per second, what is too much for the Unit hardware. The data transmitting algorithm presented shown in Fig 5.1.3.

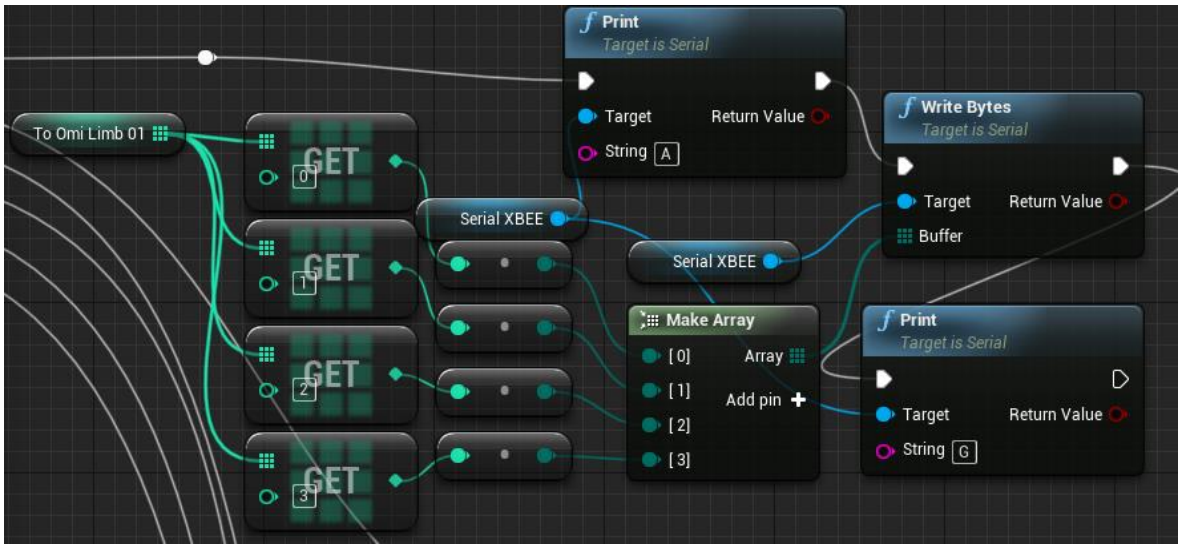

Figure 5.1.3 Data Transferring

The package from to Host has only one letter at the beginning and the end. In addition, the integer array is sent as a byte. In order to avoid value corruption while converting the integer to byte, the angle value is remapped between 0 and 255 by custom Deg_to_ADC function. It was done in order to simplify program on the Unit side and decrease the time for one calculation loop. Current solution badly effects on the limp angle positioning and angle reading resolution. However, the reading resolution is enough for primary concept testing.

5.1.4 Receive data

The Host is receiving three different packages from the Omicron. Each of the packages has a unique marking and package size. The size is only important in the meaning of reading logic, to avoid the situation when the algorithm is trying to read an empty cell from the buffer array. The first step is to define the length of the incoming package, the logic is shown in Fig 5.1.4.1. Taking into account the package length and marking of the package, the program defines a case of data storing, the logic is shown in Fig 5.1.4.2. The last step of the data receiving process is the data storage in LDB, the logic is shown in Fig 5.1.4.3. The first case has six additional subcases, each for every limb. In this way, the incoming data is separated into three groups. The first one is for the Limb matrix, the second group is dedicated to IMU and sonar readings. The third group is mentioned for the onboard laser scanner. At the end of the third case the program is calling event to build point in virtual space according to incoming data.

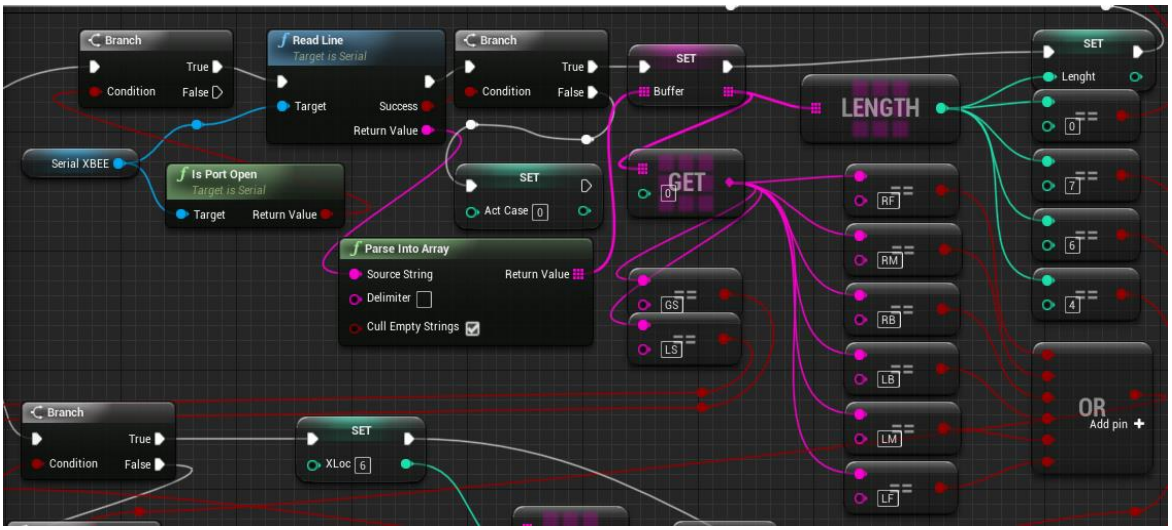

Figure 5.1.4.1 Define package lenght.

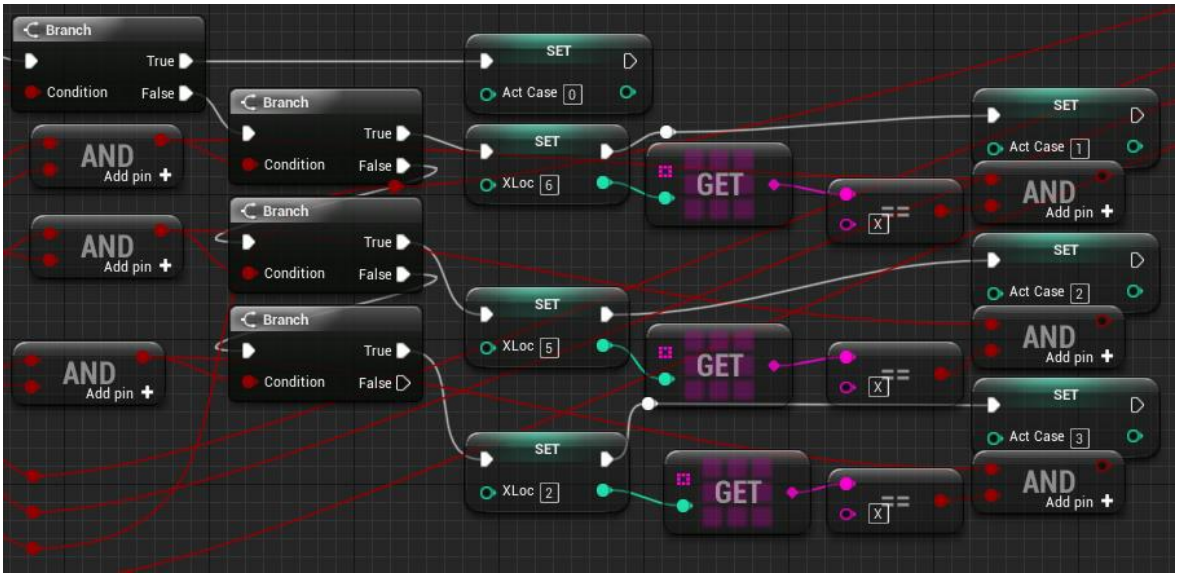

Figure 5.1.4.1 Check package marking

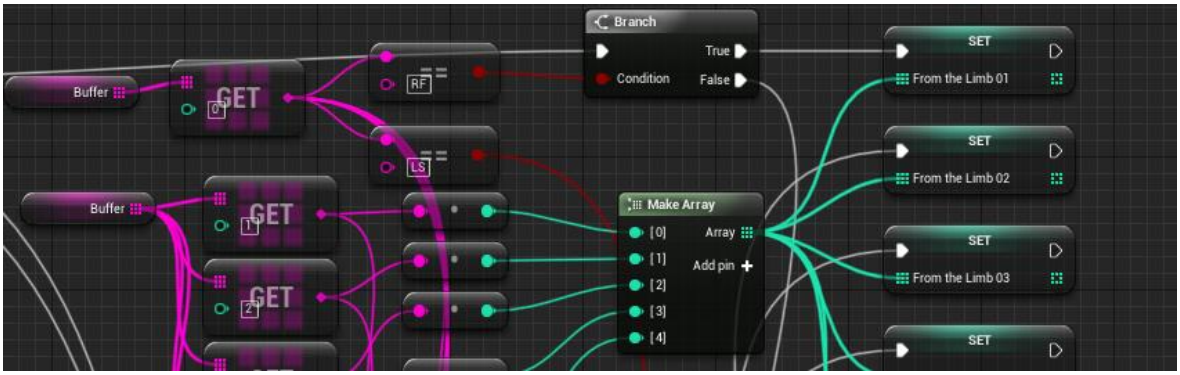

Figure 5.1.4.2 Storage data from buffer into LDB

5.2 Animation

In frame of current project the VR animations act as a reference. By other words, every life form was trained how to walk at the beginning. Later, during performance of walking process the movements are adopted to work surface. Hereby the simulation uses the set of animations for every locomotion type as walking forward, backward and turning and then the animations are adopting to environment.

5.2.1 Skeleton

The animation rig, or the Unit skeleton was created by using Maya, the computer animation and modeling software. The Unit skeleton shown in Figure 5.2.1.

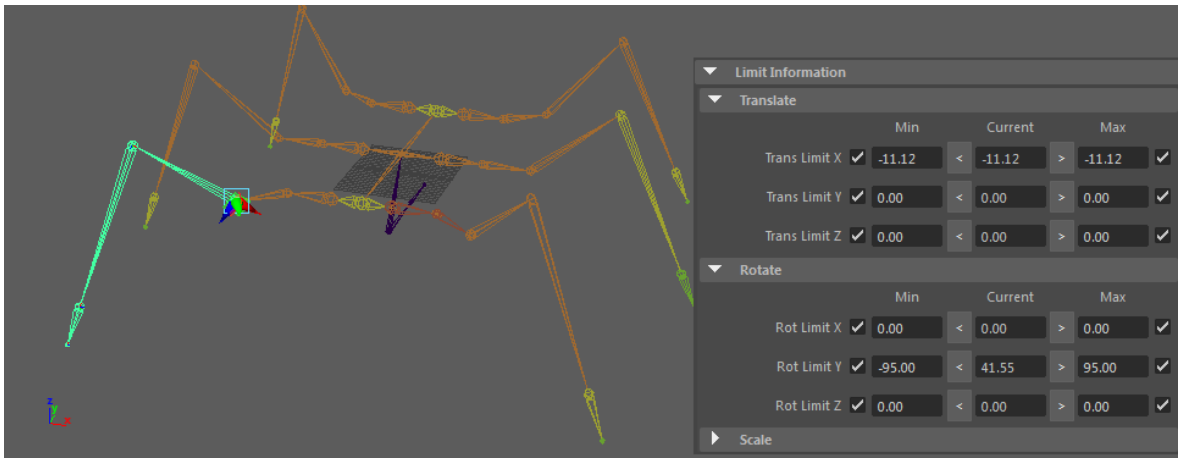

Figure 5.2.1 Unit Virtual Skeleton

In terms of size and flexibility, the skeleton replicates the real unit. Every joint has same one DoF and rotation limitations. The distance between joints also identical to the real Unit. The translation and rotation limitation will take part in the process of the animations creation as input for IK solutions.

5.2.2 Omicron simplified mesh.

On top of the skeleton was applied skin or the mesh. The mesh is shown in Fig 5.2.2. In the frame of this paper, the mesh caring only representative sense. Namely, it will act only as a visualization of internal processes in the simulation. However, after exporting files from Maya to Unreal Engine 4, this mesh will acquire collision volume and physics assets. It will help to develop the simulation environment and improve AI control algorithm testing in the future.

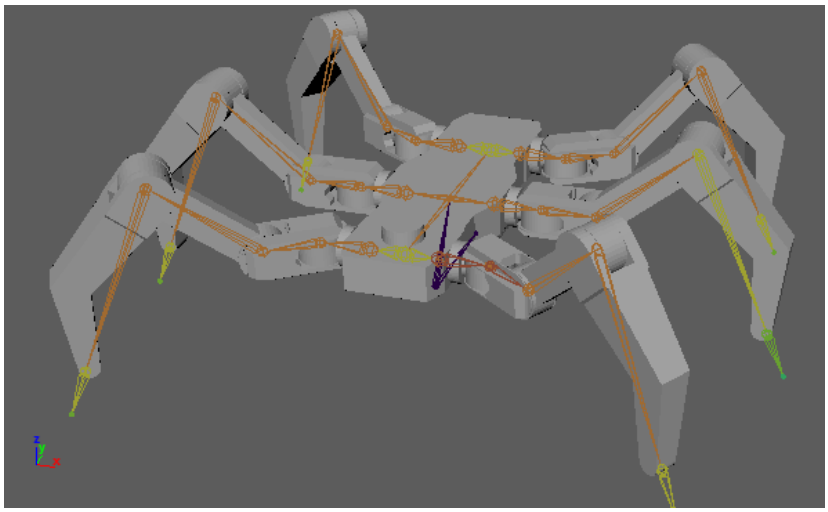

Figure 5.2.2 Simplified Omicron Mesh

5.2.3 Inverse Kinematics Handlers.

Every Omicron Limb equipped with IK Handlers. The handlers are shown in Fig 5.2.3.

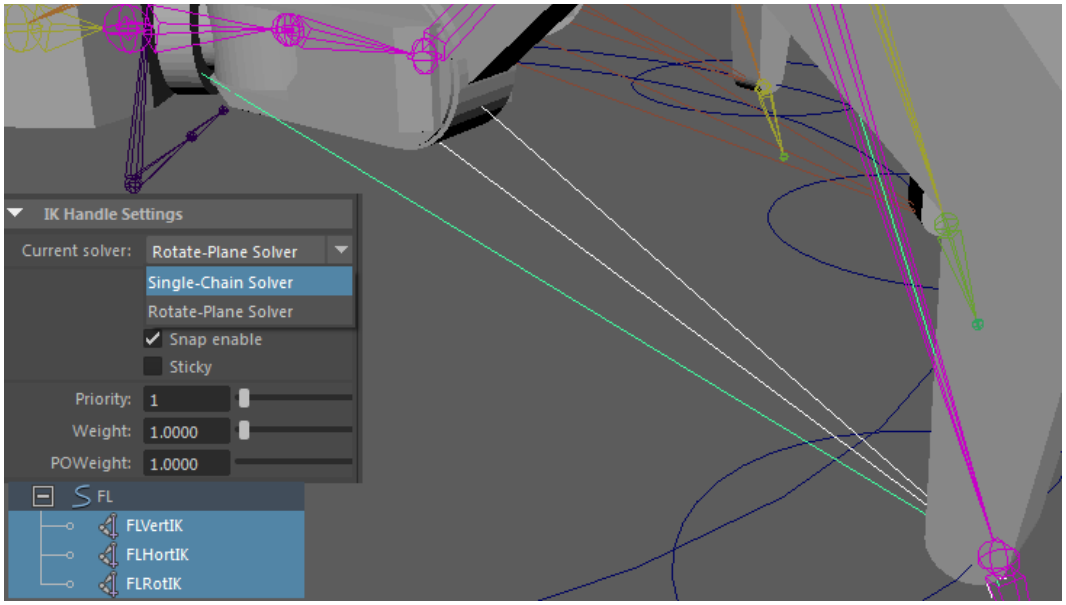

Figure 5.2.3 IK Handlers

Limb include three IK handlers. Two of them are single-chain solvers and one is rotate-plane solver. The difference single-chain solvers use the single chain solver to calculate the rotations of all the joints in the IK chain and rotate plane IK handle uses the rotate plane solver. The “VertIK” is an RPS between J2 and limb and provide vertical limb motion. The “HorIK” is an SCP between J1 and limb end and deals with horizontal movement. The “RotIK” dedicated to limb rotation at J0.

5.2.4 Animations

IK handlers are parented to the curve. When location of the curve is changing in space, all of IK Handlers are following to the curve in space. In this way, the effect of all three IK solutions are summarized and allows to match the Limb end with new position of the curve in space, taking in to account limitations of the joints. The IK Handlers is sown in Fig 5.2.4.

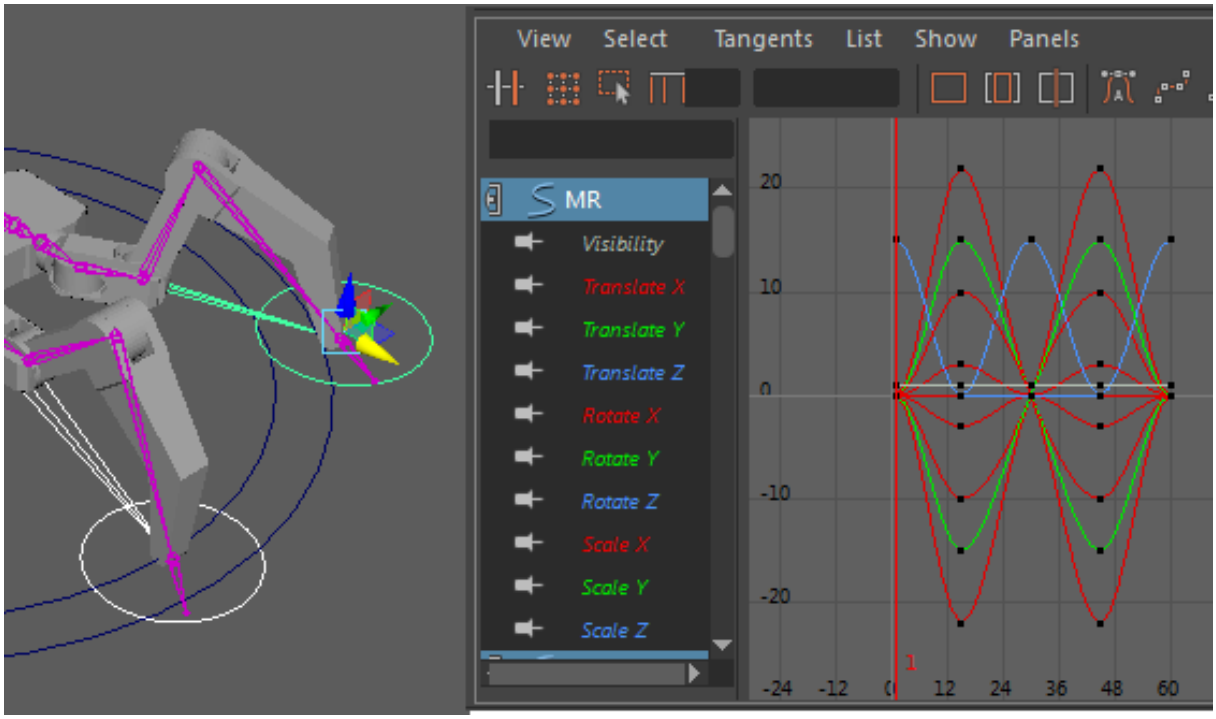

Figure 5.2.4. IK Handlers.

To create animation, every locomotion was separated into four points in time. The first and the last point are identical in order to create a smooth loop of the same animation in case of continuous action. In others two points, the legs are shifted in space according to the plot of motion. Then Maya creates s-shape curvature of motion to fulfill missing frames. In this way was created 122 base animations, which should cover different cases as walking straight, recede and turning left and right with chassis tilt into a different direction and chassis offset.

5.2.5 Animation Blend Space

Blend Spaces are special assets that allow for blending of animations based on the values of two inputs. Blend Spaces provide a means of doing more complex blending between multiple animations based on multiple values. The goal of Blend Spaces is to reduce the need for creating individual, hard-coded nodes to perform blending based on specific properties or conditions. The Blend Space shown in Fig 5.2.5.1.

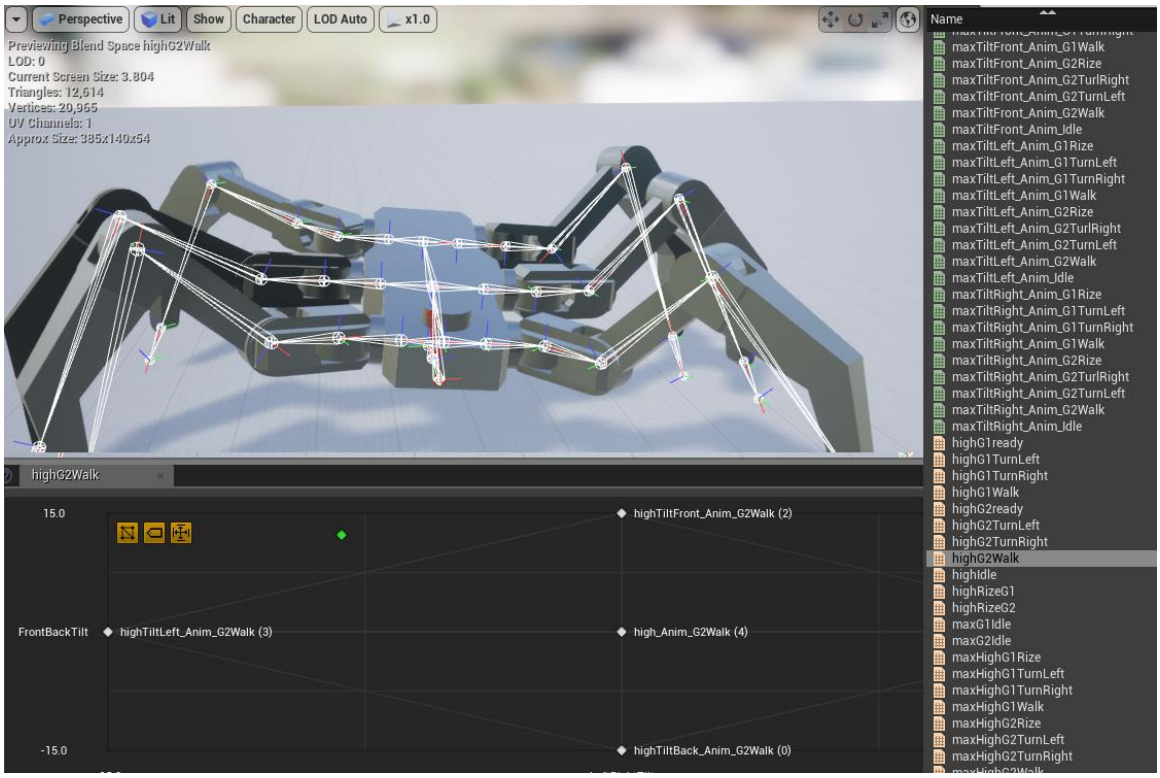

Figure 5.2.5.1 Blend Space

In this way, for every blend space were used 5 animations. The middle point of Blend Space is taking one of the 15 basic animations and two extreme values of the vertical line meant for identical base animation, but with main body front or back tilt of the main body. In this way, by changing vertical input value the end pose can be displayed with any main body tilt starting from 15 to -15 degree by using only three animations. The same principle applies to a horizontal line with two additional animations with left and right main body tilt. However, the animation blending occurs based on two values. Third value added trough end pose blending between different blend spaces based on animation weight value.

Speaking of adding third dimensions in blend spaces. The Unreal Engine development environment allows getting the similar final result in multiple ways. For Instance, the high. Before was described method of blending separate blend spaces based on their weight values. The blend weight method presented in Fig 5.2.5.2.

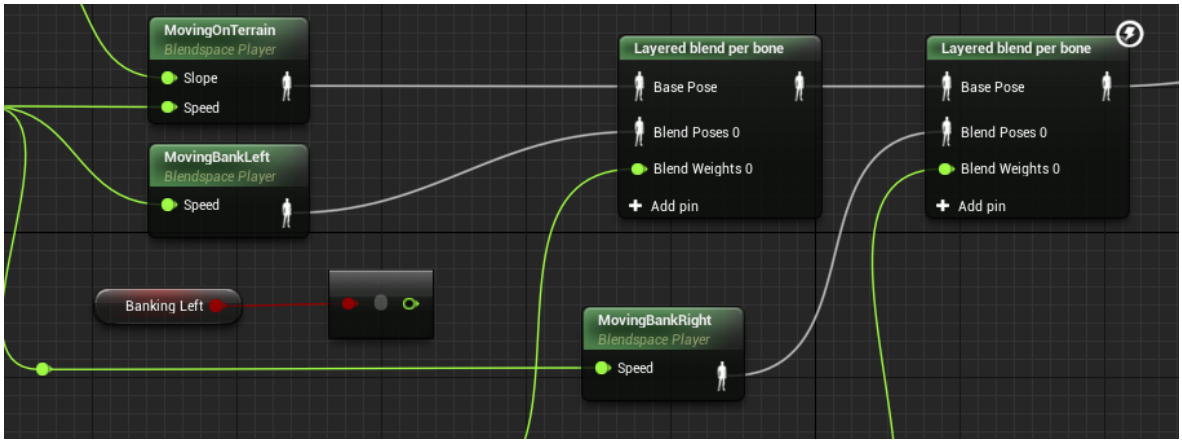

Figure 5.2.5.2 Layered Blend per bone.

By using Layering Blend Method, can be achieved the adaptation of chassis in the meaning of distance between the main body and work surface. Meanwhile, the same result can be achieved through offset of the root bone for all of the basic and blended animations. The root bone offset method presented in Fig 5.2.5.3.

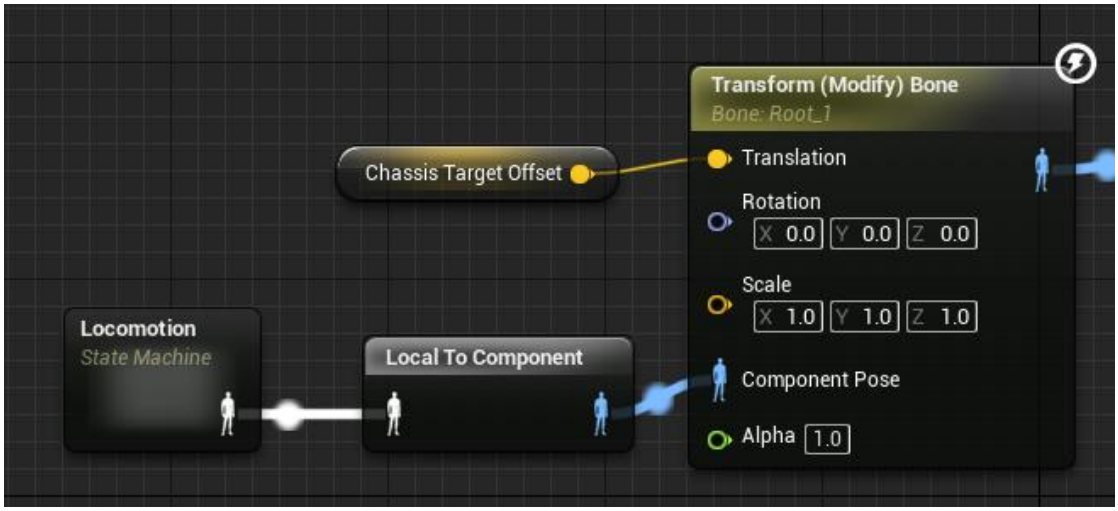

Figure 5.2.5.3 Bone transformation.

So far were chosen the second method due to the simplicity of applying. However, the first method has a more accurate outcome and more preferably to use. So far, this feature is in backlog for this project.

5.2.6 Animation State Machine

The Simulation count 15 blended animations for different cases as, walking, turning and idling. Somehow all of those animations need to be connected between each other. This task done by using Animation State Machine. State Machines provide a graphical way to break the animation of a Skeletal Mesh into a series of States, for instance, walk forward or turn left. These states are then governed by Transition Rules that control how to blend from one state to another. State machine presented in Fig 5.2.6.1

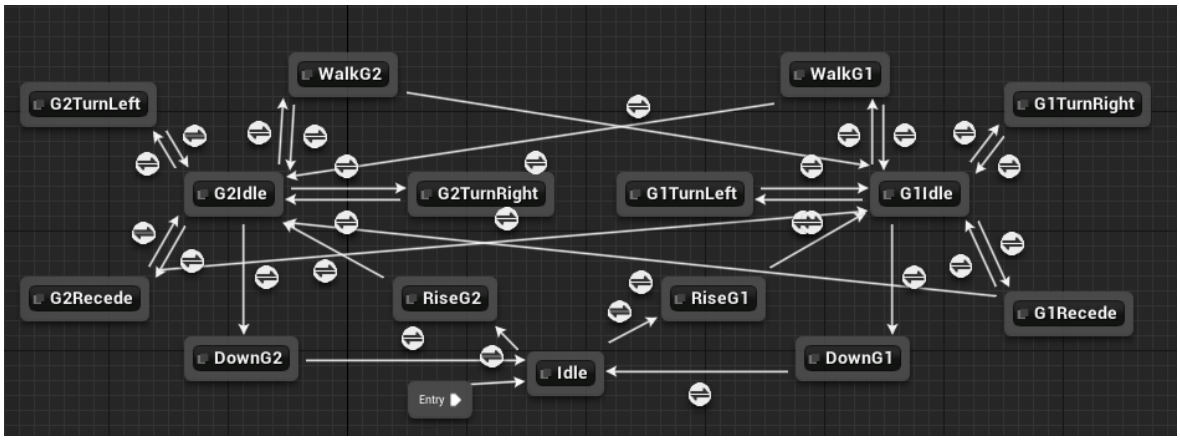

Figure 5.2.6.1 Omicron Animation State Machine.

State machine allows to change action according to incoming control command and provide a smooth transaction between states in order to avoid shard movement. Every state has rules for entering and leaving the state. The examples of entering and leaving rules presented in Fig 5.2.6.2. and 5.2.6.3 respectively.

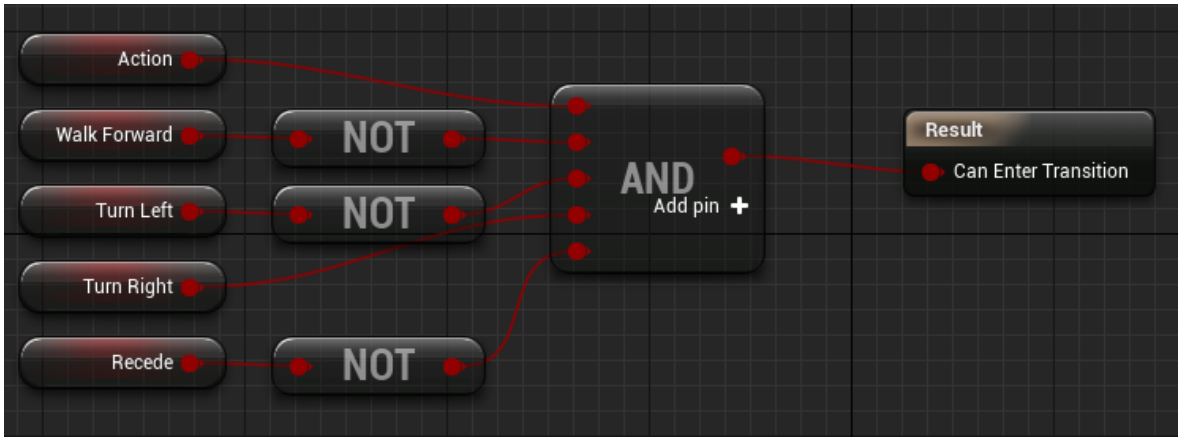

Figure 5.2.6.2 Entering Rule

In Fig 5.2.6.2 describer transaction rule between idle and turning state. In order to enter the state, it requires only an incoming action command, but only one specific. Others need to be disabled. The idle state animation has matching pose with the beginning of the walking animation, and because of this, the transaction rule does not check the actual pose.

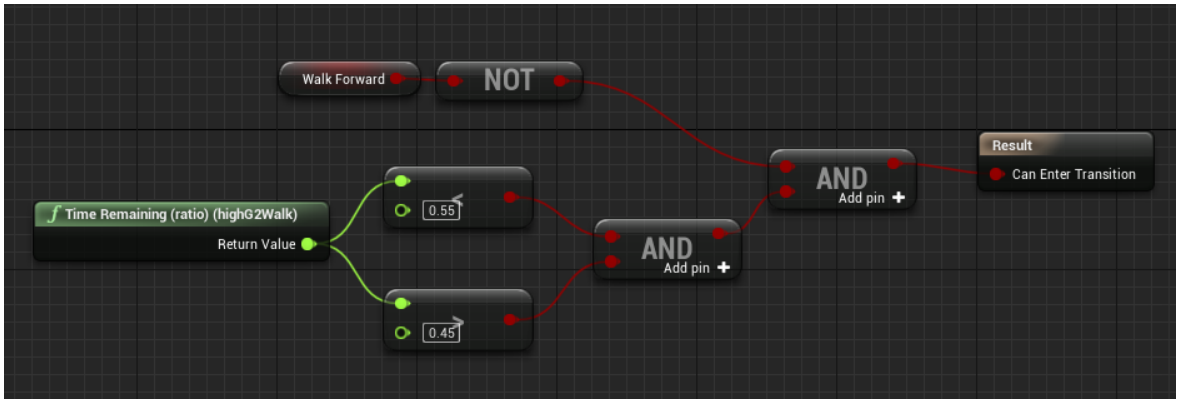

Figure 5.2.6.3 Living Rule

In Fig 5.2.6.3 describer transaction rule between walking and idle state. The walking state has two exits. First exit is entering to idle state while first group of legs are in air, and the second exit is to idle state with second group of legs in air. It was done in order to decrease reaction time for incoming command.

Generally, there is a space for the evolution of the Animation State Machine. For instance, the current state machine does not include walking by diagonal, sidewalk or the combination of the walking and turning. The advantage of the current project is that the segments of the concept are scalable and the effect will not require a redesign of the other segments. For instance, in the meaning of animations, the actual unit acts as a digital twin, so it does not matter what will happens inside of virtual environment, the Omicron will repeat it.

5.2.7 Sockets – data to send back

In order to track location and orientation of joints of mesh skeleton, every joint equipped with socket. Socket shown in Fig 5.2.7.1

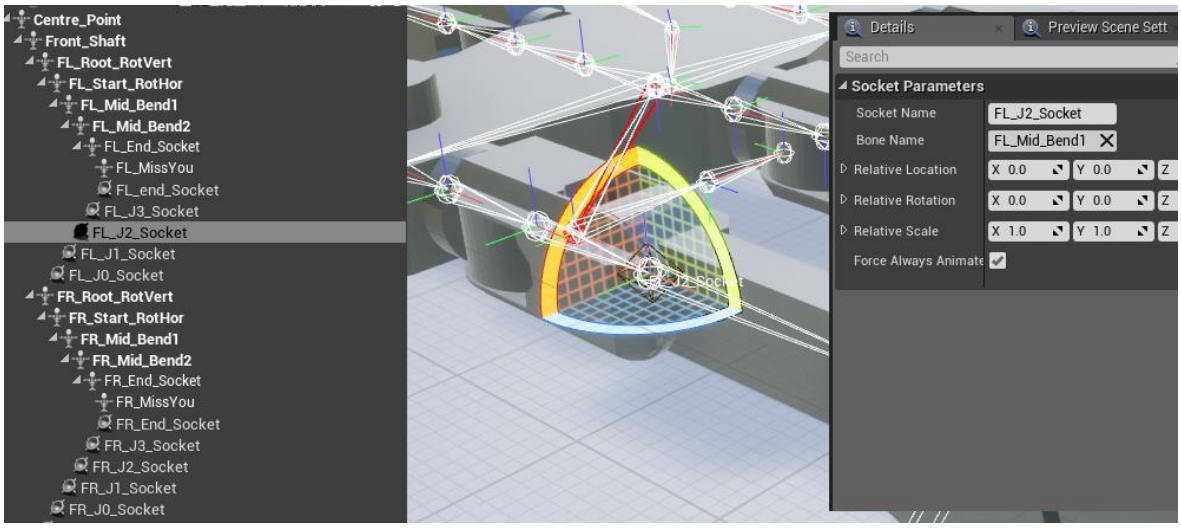

Figure 5.2.7.1 Sockets in joints of the skeleton

Every Socket is parented to the joint in the skeleton. It moves and rotates along with the joint. In the main program the Sockets are called as an object in virtual space with the option to read multiple parameters in real time. The received data from sockets can be processed and used for different purpose. For instance, to read the state of the socket. Is it in space or experience collision with some object? To evaluate control signal for servo-motors the main program reads delta rotations between sockets. Because of skeleton architecture and animations, the delta in rotations occurs only in one plane and can be used for servo-motors managing.

The socket, attached to the end of the limb, does not have an actual driver behind it. Instead of the driver, it processed along with the pressure sensor at the end of the leg. The sonar and RPLIDAR also have own socket to indicate the origin of tracers in the virtual environment.

5.2.8 Inverse Kinematics for Limbs in animations

Similar to Maya, the Unreal Engine has own tools for solutions. The IK system in Simulations made out of two parts. First part is to determine the shift in space for the Limb in order to reach the work surface. It is done by line tracker, fired from the end limb socket in the world Z direction. The algorithm presented in Figure 5.2.8.1.

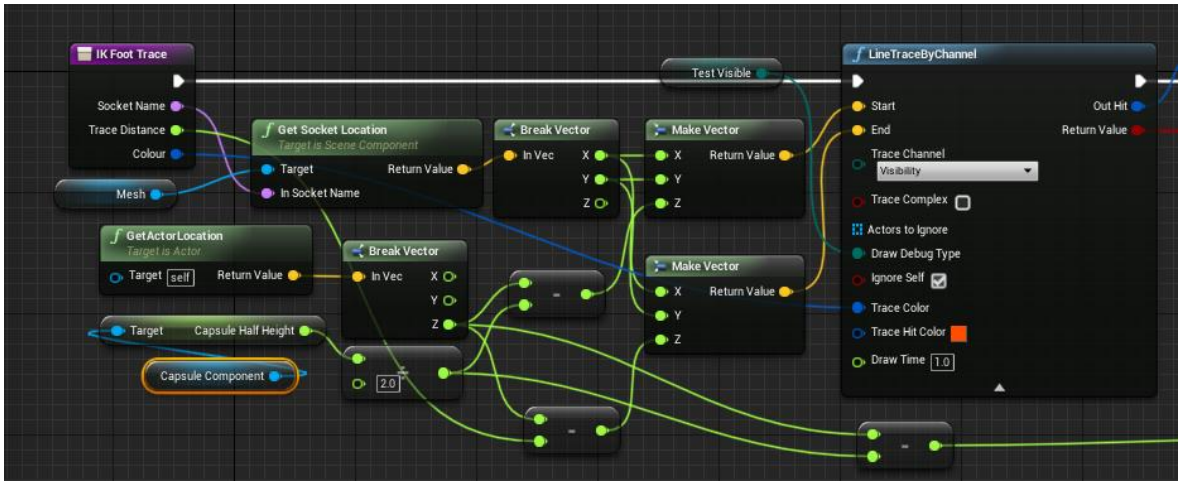

Figure 5.2.8.1 Line Tracer for IK

In case if tracer will hit surface the information of the distance between socket and work surface will go to animation blueprint and serve as an input for IK solution. UE4 tool provides two bones IK solver. The algorithm presented in Chart 5.2.8.2.

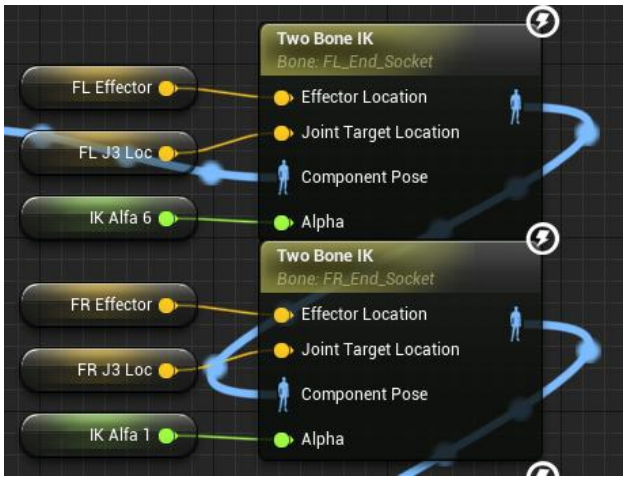

Figure 5.2.8.2 IK on Animation Side

Hereby, the position of the leg are adjusted in order to reach work surface.

5.3 The effect of the feedback to the State Machine/Simulation.

The Omicron locomotion comes from virtual environment to the real world. In case of mismatch between the simulation and the real world the simulation need to be adjusted or corrected. Thus, the process flow looks as follows. At first, the unit will assume next own actions and begin to perform them in accordance with the instructions, received from the host. In the case of inconsistencies between the instructions and the real world, the Omicron will send the corresponding data back to the host in order to reverse the simulation. After when the correction is applied to the simulation in accordance with the received data, the device will receive new instructions for action. The result of this loop is the interpretation of the surrounding space in a virtual environment. The feedback is provided by two groups of onboard sensors. First group contain the pressure sensors, sonar and IMU.

5.3.1 Pressure sensors role in Inverse Kinematic

The role of pressure sensors presented in Chart 5.3.1.

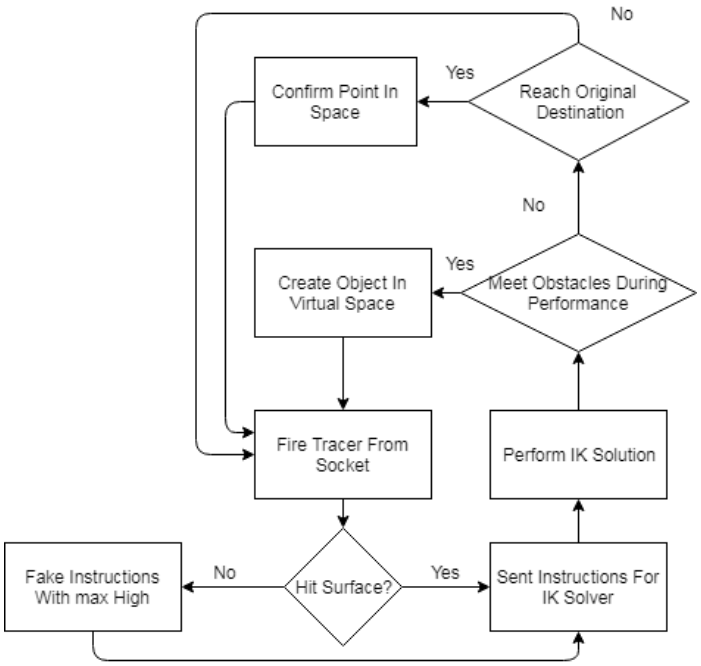

Chart 5.3.1 IK Solution Flow

The IK solver loop goes continuously during simulation process. During the process the algorithm constantly checking, does the real limb reached work surface as planned or not. The case “not” can appears if in the real world the surface does not exist anymore at old locations or the process was interrupted by hitting the new obstacle. In both cases the virtual environment should be adjusted respectively.

5.3.2 Inertial Measurement Unit role

The part of the adaptation process is the main body tilt. The tilt should provide equal load distribution to chassis and equalize the amount of motion for every limb. The source of data for mainframe tilt can various depend on the situation. The case structure presented in Chart 5.3.1.

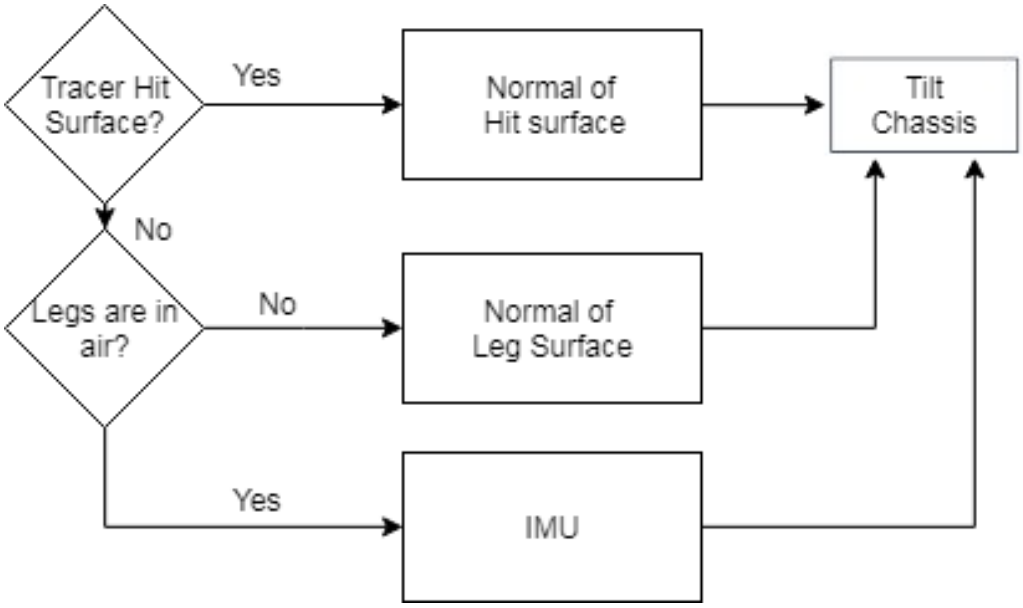

Chart 5.3.2 Source of data for chassis tilt values

The main purpose of the device is the exploration of the environment. It means that most of the time, the tracer will not hit the work surface, because it is not dawn yet, but legs still experience resistance of the surface on which the objects move. In this way, in most cases, the input source for mainframe tilt will be onboard IMU.

In addition, this specific system design created additional possibilities for environmental analysis. For instance, one of the leg groups located on the ground and calculated difference in the local coordinate system equals to zero. It means that no chassis tilt is required, but meanwhile, IMU readings show tilt. The logical conclusion can be that the device located at tilted ground and surface in the virtual environment also can be tilted in order to match with the real world.

5.3.2 Sonar role

The real world sonar functionality is similar to tracer functionality in virtual environment. In theory the data from the sonar should serve as an input for chassis offset. The case structure presented in Chart 5.3.2.

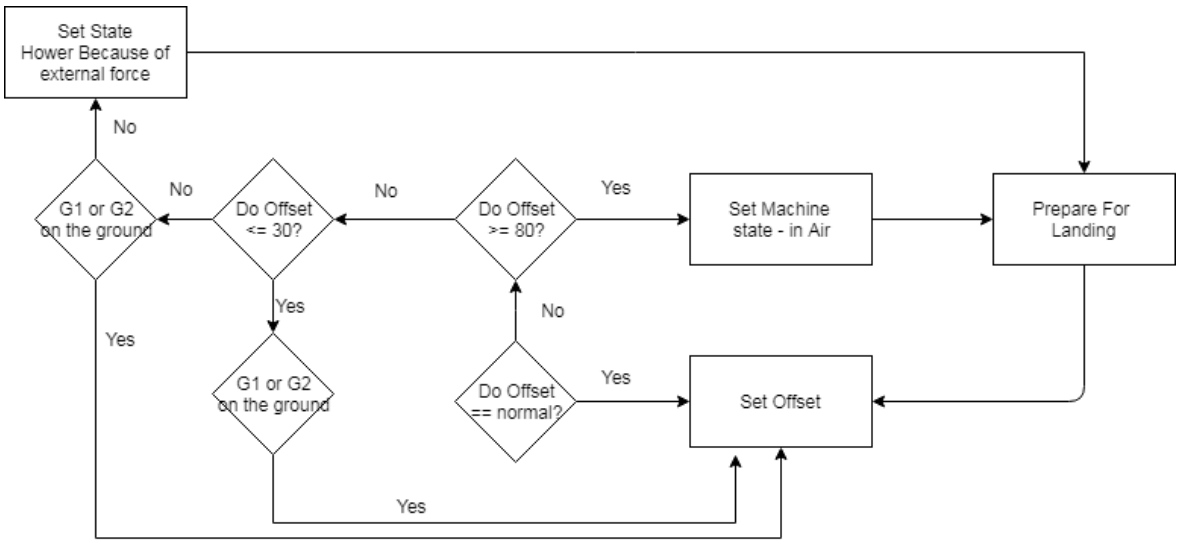

Chart 5.3.2 Sonar Case structure to set offset

However, the practical result shows that the readings from sonar and tracer go into conflict. So far, for troubleshooting, testing and simulation logic adjustment the sonar logic simplified and used only for detecting of “In Air” state.

5.4 Virtual environment map building process.

The main role in map building has the RPLIDAR. According to the datasheet, it should produce 8000 points per second. In addition, the sonar and limb pressure sensors have secondary features as producing a point for the point cloud. The location of sensors are hardly attached to the main chassis and hardcoded in a virtual environment. Hereby, by using distance data from the sonar, it is possible to define a point in the workspace. The distance from pressure sensors equals zero, but location defined by recalculation limb angles and lengths.

The Point cloud is an input for surface generation. So far, instead of the advanced algorithm to generate a surface, the simulation spawn simple shapes at the place of one point.

6. CONCEPT OUTCOME AND RESULTS ANALYSIS

6.1 MECHANICAL ISSUES

On the experimental phase of the first prototype outlined a few critical and non-critical issues in specific segments of the mechanical design of the machine. Moreover, was done an analysis of the efficiency of the hardware elements selection and general structure. The efficiency of used communication methods and protocols, in the frame of the current paper, was evaluated along with the program architecture and software realization both side of the project, the Host and The Omicron side. The current chapter is dedicated to various shortcomings in mechanical design.

6.1.1 Machine Geometry

The origin point of middle group arms slightly shifted outside, in comparing with the front and back groups. While the unit is walking the limb creates the specific volume in the work area. This area can be named as the most visited area, the normal mode walking work area or high-density volume. However, each limb has this specific volume. The purpose of the shift is to move high-density volume and decrease the interference zone in order to increase step length. The practical result shows that the shift is not big enough. This issue is bypassed by the software solution. The walking animation was made with taking in to account this specific parameter of the chassis.

6.1.2 The Plastic gears in primary gearbox.

The pre-prototype of the limb was built around hobby gearbox with integrated low voltage DC motor. The defects of the design were taken into account and according to experience was build the second version of the limb, with which the Omicron is equipped. The more powerful engine and bigger gear ratio in the secondary gearbox should increase load on the gears in primary gearbox. The new values were close to limits of the hobby product. The alternatives, which were made out of the metals, cost multiple times more. However, the actual unit is not the main point of the paper and serves only as an object to prove concept. The safety feature inside of primary gearbox should prevent damage in case of overload.

Unfortunately, the new batch of the primary gearboxes with the same product name and were bought at the same shop, as at the first place. Did not have safety feature. It was detected when the prototype was almost finished. The metal gearbox can fix the situation, but on this stage, it requires enormous investment to change primary gearbox and adapt the general design to new gearboxes. A cheaper and faster solution is to find and change all the plastic gears to metal gears inside of the primary gearbox. Anyway, this will be outside of the thesis because of timing. New gears will arrive at summer 2019.

6.2 Hardware issue

The hardware design of the Omicron can be considered more or less successful, expect performance in several segments. The first segment can be outlined as the operation frequency of the Omicron core is 16 MHz. It is not enough to provide maximum data output and update rate for the proper operation. Second issue is the data transmitting capacity, the calculation, related to data transmitting, was made in Chapter 3.4 and current set up for wireless data communication is not able to transmit data on the desired frequency. The hardware selection is a poor choice and should be replaced with a better communication solution as a Wi-Fi module. The last one is the IMU. Because of the noise level in raw readings of the IMU, the sensor is not able to track slow motion.

6.3 Omicron Program issue

The question of low-frequency operation brings a question of current algorithm optimization.

6.3.1 Place of the PRLIDAR segment in the program of the Omicron core.

During the analysis of the algorithm architecture of the Omicron, Core program was evaluated the potential issue. Does not matter how would be realized algorithm sequence of the Omicron core program, the laser scanning sensor cannot provide the maximum output without direct connection to the host or until the moment when the test unit can be equipped with more advanced hardware. The alternative with the frequency level of GigaHertz or has multiple core processor in order to separate calculation processes.

As an outcome, the amount of the data points goes down to 30 points per second. This number is too low for the effective map building process. However, the issue was known from the beginning of the process. And the decision, to continue with the current configuration, was made because of the higher efficiency and accuracy then other methods based, for instance, on ultrasonic radar mapping and less investment requirement than in the case of more advanced machine vision solutions.

6.3.2 Data read process and ASCII interference

Problem – The characters can be read as an integer. The ASCII letters have decimal value in the range of 65-90. In case if one segment of the array will be lost and at the moment when the program will read the check letter, the joint value may be identical. It will cause the wrong package interpretation. The chance is low, but still possible.

6.4 Virtual Environment Issues

In this chapter are outlined problems, related to virtual environment segment of the study.

6.4.1 Inverse Kinematics Solution

The visualization of the Omicron in the virtual environment may seem smooth and correct. In practice, the IK solver is working in a way, that and it pulls the end of the limb to the ground. Depends on the situation the IK is activated with 60 or 80 percent and with walking animation blend the final outcome looks acceptable and the actual unit is also able to walk. This is the incorrect solution and needs to be redone. The IK Alpha, the coefficient of IK solver, need to be more dynamic. It can be done by binding the IK Alpha with walking animation. The closer is the animation to the moment in time when the limb should land on the work surface, the bigger IK Alpha should be. And vice versus, as soon as the Unit wand to start the detachment process, the IK Alpha should slowly go to zero.

6.4.2 Character Pawn and Movement

In the simulation, the shift in space, or the result of the locomotion, is hardcoded and bind to the Character Pawn. Shortly speaking, the pawn moves when it should move, but the result of the shifting in space should be the result of physics calculation in a known environment and IMU input in an unknown environment. The UE4 is capable with this task, but the realization of it is more complicated. It requires more resources and other aspects of the simulation should be tested outside of this feature. Hereby, the current setup suits for primary testing, but it should be redone in perspective.

6.5 Future work

For the next iteration of the testing and project development, it is necessary to improve the IK solution system in UE4. Virtual Environment Map building algorithm needs to be implemented and replace the current solution. Implement advanced physics for the simulation and upgrade hardware for the Cote and communication segment. Most of the issues were outlined in Chapter 6. As future work is planned to solve this list of problems.

SUMMARY

Current paper was aimed to research possibilities for the simulation of perception processes for an artificial life form. In the frame of the project was made a primary analysis of the human concept of the perception. In order to outline goals clearly the general definition of the perception needed to be placed at the first step. The definition is stated as a - the ability to see, hear or become aware of something through the senses. This sentence is the core of the current project. The second step is to rephrase it into the question. What exactly the test subject need to see, hear or become aware of something and how. Before even paper writing the answer was formulated as a – ability to see surrounding, understand the general picture of the landscape and to be aware of own position in the world, be aware of changes in the environment. The representation of the world should be dynamic, flexible and the machine should adapt to changes in real time. Be aware of own locomotion. Not just be hardly programmed to perform a set of movement in case sequence, but be aware of the position of every limb and able to correct position according to needs. Be also aware of the own position in the meaning of tilting main body. Does the tilt happen because of the internal decision or the external effect? This moment is important in order to continue collecting data with taking in to account town position.

During the project, as a second step, was designed and built the test subject, control unit, machine, in the paper it was named by differently, but in general, it is a walking robot, which one imitates a six leg insect. The complicated mechanical design was a good exercise to outline different aspects of mechanical engineering as a Finite Element Method analysis, driver unit development as the custom servo-motors with calculated rotation speed and torque, and the segment of knowledge base related to industrial manipulators. The hardware design was a good option to apply to practice accrued knowledge in the electrical circuit design. The project covers the use of multiple sensors with different complexity level.

The internal communication of the Omicron realized based on two different communication protocols, as RS-232 and I2C, thanks to the Applied Data Communication subject at the second semester. The external communication was established by RS-232 protocol and radio communication between Omicron and the Host.

The development environment as a game engine Unreal Engine 4 is relatively something new for the Author. Originally, this environment is designed for the entertainment products as video games. The idea to use it in the framework of a similar project was in the mind of the Author was born a long time ago, but only in the frame of this study this idea was realized on was fully implemented at a certain level. The concept of usage the game engine, as a space for mental processes of the artificial life form, and connect the real world with the virtual reality truly excite the Author and he sees significant possibilities in it. Skeptically the host perception simulation program can be named just as a set of few nonflexible functions. Does the real brain a much different? It is just more advanced and complicated, but I will be there one day and as a start, it shows significant results.

In Virtual environment was created the rig of the actor, mesh for the body and then sets of animation, which covers a few cases, was created. By the state machine, the flexibility of the locomotion was expanded and adjusted according to data from internal sensors from the test unit.

In every covered segment were design mistakes, but for the first prototype, the general outcome is satisfying. For future recommendations, it is needed to note that there is a big opportunity for modifying the prototype. First, upgrade hardware for more advanced in order to improve data exchange and calculation performance and also transfer the Host segment to the machine. The host should act as a coordinator between units and use collected data in order to build the bigger picture. However, this is for far future. For the next iteration is needed to solve the issues mentioned in Chapter 6.

The few questions still remain outside of the project frame. First one is the algorithm to convert point cloud in to surface, which one can describe environment. And in addition, the algorithm of flexible dynamic environment modification also should take a place. The second missing point is the purpose for the artificial life. Or simply speaking, the artificial Intellect is missing. SO far this question does not bother since the Author also does not have Purpose in life. The loop goes like: Set the goal, achieve it, take in to account the acquired experience and then set the new goal. How the AI can be how the AI can be taught this.

As a conclusion can say that the main point of the paper was to create a system to study the options of the perception and to introduce perception later into more complex and more effective AI control algorithms of future machines. The unit is aware of the environment and able to adapt to the dynamic environment.

LIST OF REFERENCES

[1] Vladislav Babuskin, “A Robust Control System For A Mobile Robot” [Online]. https://digi.lib.ttu.ee/i/?7504. [Accessed 03 01 2019].

[2] “H-Bridges – the Basics” [Online]. http://www.modularcircuits.com/blog/articles/h-bridge-secrets/h-bridges-the-basics/. [Accessed 03 01 2019].

[3] ”Introduction to Servo Control & PID Tuning ” [Online]. http://support1.motioneng.com/pdf_notes/powerpoint/Introduction%20to%20PID%20and%20Tuning4.pdf. [Accessed 03 01 2019].

[4] “RPLIDAR Application Note Arduino Driver Support & Demo” [Online]. http://www.robopeak.net/data/doc/rplidar/appnote/RPLDAPPN01-rplidar_appnote_arduinolib-enUS.pdf [Accessed 03 01 2019].

[5] “DC Toy / Hobby Motor - 130 Size” [Online]. http://www.farnell.com/datasheets/1863913.pdf. [Accessed 03 01 2019].

[6] “TFK-280SA-22125 MOTOR” [Online]. https://www.arduino.cc/documents/datasheets/DCmotor.PDF. [Accessed 03 01 2019].

[7] “RPLIDAR A2” [Online]. https://www.slamtec.com/en/Lidar/A2. [Accessed 03 01 2019].

[8] “MPU-9150 Product Specification Revision 4.3” [Online]. https://www.invensense.com/wp-content/uploads/2015/02/MPU-9150-Datasheet.pdf. [Accessed 03 01 2019].

[9] “DIGI XBEE® S1 802.15.4 RF MODULES” [Online]. https://www.digi.com/pdf/ds_xbeemultipointmodules.pdf. [Accessed 03 01 2019].

[10] GitHub Profile Page of Martin Šošić [Online]. https://github.com/Martinsos?tab=repositories. [Accessed 03 01 2019].

[11] GitHub Profile Page of Kris Winer [Online]. https://github.com/kriswiner. [Accessed 03 01 2019].

[12] GitHub Profile Page of Rodrigo Villani [Online]. https://github.com/RVillani. [Accessed 03 01 2019].

APPENDICES

Appendix 1 Omicron Limb program

```
71 void loop() {
72
73 // ---------- Collect data from joint potentiometers ----------
74 Input0 = map(analogRead(J0A),0,1023,-255,255);
75 Input1 = map(analogRead(J1A),0,1023,-255,255);
76 Input2 = map(analogRead(J2A),0,1023,-255,255);
77 Input3 = map(analogRead(J3A),0,1023,-255,255);
78
79 OtData[0]= map(analogRead(J0A),0,1023,0,255);
80 OtData[1]= map(analogRead(J1A),0,1023,0,255);
81 OtData[2]= map(analogRead(J2A),0,1023,0,255);
82 OtData[3]= map(analogRead(J3A),0,1023,0,255);
83 OtData[4]= map(analogRead(LPS),0,1023,0,255);
84 // ---------- Collect data from joint potentiometers END-------
85
86 // ---------- Use incoming data as Set Point  -----------------
87 Setpoint0 = map(InData[0],0,255,-255,255);
88  //TODO add limits related to each joint
89 Setpoint1 = map(InData[1],0,255,-255,255);
90 Setpoint2 = map(InData[2],0,255,-255,255);
91 Setpoint3 = map(InData[3],0,255,-255,255);
92 // ---------- Use incoming data as Set Point  END--------------
93
94 // ---------- PID Computing the output (PWM)-------------------
95 J0PID.Compute();
96 J1PID.Compute();
97 J2PID.Compute();
98 J3PID.Compute();
99 // ---------- PID Computing the output (PWM)END----------------
100
101 // ---------- Convert Output Value to PWM Signal --------------
102 Output0a = constrain((Gain*(abs(Output0))), 0 , 255);
103 Output1a = constrain((Gain*(abs(Output1))), 0 , 255);
104 Output2a = constrain((Gain*(abs(Output2))), 0 , 255);
105 Output3a = constrain((Gain*(abs(Output3))), 0 , 255);
106 // ---------- Convert Output Value to PWM Signal END-----------
```

```
// ---------- Set Driver direction control pins ---------------

             if (Output0 > 0) { // 1-st Joint (Buildin to main Body)
        digitalWrite(M0A, HIGH);  dM0A = 1;
        digitalWrite(M0B, LOW);   dM0B = 0;
      } else if (Output0 < 0) {
        digitalWrite(M0A, LOW);   dM0A = 0;
        digitalWrite(M0B, HIGH);  dM0B = 1;
      } else if (Output0 == 0){
        digitalWrite(M0A, LOW);   dM0A = 0;
        digitalWrite(M0B, LOW);   dM0B = 0;
      }

                  if (Output1 > 0) { // 2-nd Joint (Buildin to main Body)
        digitalWrite(M1A, HIGH);  dM1A = 1;
        digitalWrite(M1B, LOW);   dM1B = 0;
      } else if (Output1 < 0) {
        digitalWrite(M1A, LOW);   dM1A = 0;
        digitalWrite(M1B, HIGH);  dM1B = 1;
      } else if (Output1 == 0){
        digitalWrite(M1A, LOW);   dM1A = 0;
        digitalWrite(M1B, LOW);   dM1B = 0;
      }

             if (Output2 > 0) { // 3-rd Joint (Buildin to main Body)
        digitalWrite(M2A, HIGH);  dM2A = 1;
        digitalWrite(M2B, LOW);   dM2B = 0;
      } else if (Output2 < 0) {
        digitalWrite(M2A, LOW);   dM2A = 0;
        digitalWrite(M2B, HIGH);  dM2B = 1;
      } else if (Output2 == 0){
        digitalWrite(M2A, LOW);   dM2A = 0;
        digitalWrite(M2B, LOW);   dM2B = 0;
      }
                    if (Output3 > 0) { // 4-th Joint (Buildin to main Body)
        digitalWrite(M3A, HIGH);  dM3A = 1;
        digitalWrite(M3B, LOW);   dM3B = 0;
      } else if (Output3 < 0) {
        digitalWrite(M3A, LOW);   dM3A = 0;
        digitalWrite(M3B, HIGH);  dM3B = 1;
      } else if (Output3 == 0){
        digitalWrite(M3A, LOW);   dM3A = 0;
        digitalWrite(M3B, LOW);   dM3B = 0;
      }

// ---------- Set Driver direction control pins END------------
```

```
155 // ---------- Set Driver speed control (PWM) -----------------
156         analogWrite(M0S,Output0a);
157         analogWrite(M1S,Output1a);
158         analogWrite(M2S,Output2a);
159         analogWrite(M3S,Output3a);
160 // ---------- Set Driver speed control (PWM) END---------------
161 
162 }
163 
164 
165 // Function that executes whenever data is requested by master
166 void requestEvent()
167 {
168   uint8_t OtBuffer[5];
169     for(int i=0;i<5;i++)
170     { OtBuffer[i]=OtData[i];}
171   Wire.write(OtBuffer,5);
172 }
173 
174 // Function that executes whenever data is received from master
175 void receiveEvent(int howMany){
176   for(howMany; howMany > 0; howMany--){
177     int c = Wire.read();
178    InData[4 - howMany ] = c;}
179 }
```

Appendix 2 Omicron Sonar program

```
void setup() {
  Wire.begin(SonarAdress);
  Wire.onRequest(requestEvent); // register event
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input

for (int j = 0; j <= 19; j++){
                                digitalWrite(trigPin, LOW);
                                delayMicroseconds(2);
                                digitalWrite(trigPin, HIGH);
                                delayMicroseconds(10);
                                digitalWrite(trigPin, LOW);
                                duration = pulseIn(echoPin, HIGH);
                                setdist[j]= duration*0.034/2;
                                }
}

void loop() {
  if (i < 0){i = 0;}
//--------------------------------
      digitalWrite(trigPin, LOW);
      delayMicroseconds(2);
      digitalWrite(trigPin, HIGH);
      delayMicroseconds(10);
      digitalWrite(trigPin, LOW);
      duration = pulseIn(echoPin, HIGH);
      setdist[i] = duration*0.034/2;
//--------------------------------
    for (int k = 0; k <= 19; k++){
      avgdist=avgdist+setdist[k];
      delay(1);}

        avgdist=avgdist/20;
        distance = avgdist;
        avgdist=0;

        outdata = distance;
        if (outdata <  0 ){outdata = 0;}

//--------------------------------
    i++;
    if (i >= 20){i = 0;}
  }

// Function that executes whenever data is requested by master
void requestEvent()
{   Wire.write(outdata); }
```

Appendix 3 Omicron Core program

```
//---------------------Set Up ----------------------
void setup()
{
  Wire.begin();        // join i2c bus (address optional for master)
  Serial.begin(115200);
  lidar.begin(Serial2);
  Serial3.begin(57600);

  pinMode(intPin, INPUT);
  digitalWrite(intPin, LOW);
  pinMode(blinkPin, OUTPUT);
  digitalWrite(blinkPin, HIGH);
  pinMode(LaserSpeed, OUTPUT);


  uint8_t c = readByte(MPU9150_ADDRESS, WHO_AM_I_MPU9150);  // Read WHO_A

  if (c == 0x68) // WHO_AM_I should always be 0x68
  {
  MPU6050SelfTest(SelfTest); // Start by performing self test and reporti
  if(SelfTest[0] < 1.0f && SelfTest[1] < 1.0f && SelfTest[2] < 1.0f && Se
  delay(1000);
  }

  calibrateMPU9150(gyroBias, accelBias); // Calibrate gyro and accelerome

  delay(1000);

  initMPU9150(); // Inititalize and configure accelerometer and gyroscope

  // Read the WHO_AM_I register of the magnetometer, this is a good test
  uint8_t c = readByte(AK8975A_ADDRESS, WHO_AM_I_AK8975A);  // Read WHO_A
  delay(1000);

  // Get magnetometer calibration from AK8975A ROM
  initAK8975A(magCalibration);

  MagRate = 10; // set magnetometer read rate in Hz; 10 to 100 (max) Hz a
  }
  else
  {
  //  while(1) ; // Loop forever if communication doesn't happen
  }

}
//---------------------Set Up END------------------
```

```
284 //---------------------Collect  data ---------------
285
286 void getI2CData(){          // collect data from segments
287 Wire.requestFrom(01, 5);    // request data from segment 01
288     for(int i=0;i<5;i++) { int c = Wire.read(); FromLimb01[i]=  c;}
289 Wire.requestFrom(02, 5);    // request data from segment 02
290     for(int i=0;i<5;i++) { int c = Wire.read(); FromLimb02[i]=  c;}
291 Wire.requestFrom(03, 5);    // request data from segment 03
292     for(int i=0;i<5;i++) { int c = Wire.read(); FromLimb03[i]=  c;}
293 Wire.requestFrom(10, 5);    // request data from segment 10
294     for(int i=0;i<5;i++) { int c = Wire.read(); FromLimb10[i]=  c;}
295 Wire.requestFrom(20, 5);    // request data from segment 20
296     for(int i=0;i<5;i++) { int c = Wire.read(); FromLimb20[i]=  c;}
297 Wire.requestFrom(30, 5);    // request data from segment 30
298     for(int i=0;i<5;i++) { int c = Wire.read(); FromLimb30[i]=  c;}
299 Wire.requestFrom(50, 1);    // request data from segment 50
300                            int c = Wire.read(); Sonar =  c;}
301 //---------------------Collect  data END------------
302
303 //---------------------Distribute data -------------
304 void setData(){ // deliver data to the segments
305
306   Wire.beginTransmission(01); // transmit to device #01
307   Wire.write(ToLimb01, 4); // sends Setpoint data
308   Wire.endTransmission(); // stop transmitting
309
310   Wire.beginTransmission(02); // transmit to device #01
311   Wire.write(ToLimb02, 4); // sends Setpoint data
312   Wire.endTransmission(); // stop transmitting
313
314   Wire.beginTransmission(03); // transmit to device #01
315   Wire.write(ToLimb03, 4); // sends Setpoint data
316   Wire.endTransmission(); // stop transmitting
317
318   Wire.beginTransmission(10); // transmit to device #01
319   Wire.write(ToLimb10, 4); // sends Setpoint data
320   Wire.endTransmission(); // stop transmitting
321
322   Wire.beginTransmission(20); // transmit to device #01
323   Wire.write(ToLimb20, 4); // sends Setpoint data
324   Wire.endTransmission(); // stop transmitting
325
326   Wire.beginTransmission(30); // transmit to device #01
327   Wire.write(ToLimb30, 4); // sends Setpoint data
328   Wire.endTransmission(); // stop transmitting}
329 //---------------------Distribute data END----------
```

```
//---------------------RPLIDAR Segment -------------
void RPLID(){
  if (IS_OK(lidar.waitPoint())) {

    float distance = lidar.getCurrentPoint().distance;
    //distance value in mm unit
    float angle    = lidar.getCurrentPoint().angle;
    //anglue value in degree
    bool  startBit = lidar.getCurrentPoint().startBit;
    //whether this point is belong to a new scan
    byte  quality  = lidar.getCurrentPoint().quality;
    //quality of the current measurement

    dist =  (distance / 10);

    if(dist>10){
     Serial3.print("LS");  Serial3.print(" ");
     Serial3.print(angle); Serial3.print(" ");
     Serial3.print(dist);  Serial3.print(" ");
     Serial3.println(" X");
    }

  }
  else {
    rplidar_response_device_info_t info;
    if (IS_OK(lidar.getDeviceInfo(info, 100))) {
    lidar.startScan();
    }
  }

}
//---------------------RPLIDAR Segment End----------
```

```
//---------------------IMU-9150 Segment -------------
void IMU()
{
    // If intPin goes high or data ready status is TRUE,
    // all data registers have new data
    if (readByte(MPU9150_ADDRESS, INT_STATUS) & 0x01) {
      // On interrupt, check if data ready interrupt
    }
    readAccelData(accelCount);  // Read the x/y/z adc values
    getAres();

    // calculate the accleration value into actual g's
    ax = (float)accelCount[0]*aRes;
    // get actual g value, this depends on scale being set
    ay = (float)accelCount[1]*aRes;
    az = (float)accelCount[2]*aRes;

    readGyroData(gyroCount);  // Read the x/y/z adc values
    getGres();

    // Calculate the gyro value into actual degrees per second
    gx = (float)gyroCount[0]*gRes;
    // get actual gyro value, this depends on scale being set
    gy = (float)gyroCount[1]*gRes;
    gz = (float)gyroCount[2]*gRes;

    mcount++;
    if (mcount > 200/MagRate) {
    // Setting the magnetometer read rate (see below)
    readMagData(magCount);  // Read the x/y/z adc values
    mRes = 10.*1229./4096.;
    // Conversion from 1229 microTesla full scale (4096) to 12.29 Gauss full scale
    // So far, magnetometer bias is calculated and subtracted here manually

    magbias[0] = -5.;   // User environmental x-axis correction in milliGauss
    magbias[1] = -95.;  // User environmental y-axis correction in milliGauss
    magbias[2] = -260.; // User environmental z-axis correction in milliGauss

    // Calculate the magnetometer values in milliGauss
    // Include factory calibration per data sheet and user environmental corrections
    mx = (float)magCount[0]*mRes*magCalibration[0] - magbias[0];
    my = (float)magCount[1]*mRes*magCalibration[1] - magbias[1];
    mz = (float)magCount[2]*mRes*magCalibration[2] - magbias[2];
    mcount = 0;}
```

```
409     if(!AHRS) {
410     tempCount = readTempData();  // Read the x/y/z adc values
411     temperature = ((float) tempCount) / 340. + 36.53; // Temperature in degrees Centigrade
412     }
413  }
414
415   Now = micros();
416   deltat = ((Now - lastUpdate)/1000000.0f);
417   // set integration time by time elapsed since last filter update
418   lastUpdate = Now;
419    MadgwickQuaternionUpdate(ax,ay,az,gx*PI/180.0f, gy*PI/180.0f,gz*PI/180.0f,my,mx,mz);
420
421     // Serial print and/or display at 0.5 s rate independent of data rates
422     delt_t = millis() - count;
423
424     yaw   = atan2(2.0f*(q[1]*q[2]+q[0]*q[3]),q[0]*q[0]+q[1]*q[1]-q[2]*q[2]-q[3]*q[3]);
425     pitch = -asin(2.0f*(q[1]*q[3]-q[0]*q[2]));
426     roll  = atan2(2.0f*(q[0]*q[1]+q[2]*q[3]),q[0]*q[0]-q[1]*q[1]-q[2]*q[2]+q[3]*q[3]);
427     pitch *= 180.0f / PI;
428     yaw   *= 180.0f / PI;
429     yaw   += 8.46;
430     // Declination at Tallinn, Estonia is  positive. 8 degrees 46 minutes  on 2018-12-05
431     roll  *= 180.0f / PI;
432     count = millis();
433
434 }
435 //---------------------IMU-9150 Segment END----------
436
437 //-----------CLean Incoming serial buffer -----------
438 void cleanBuff(){
439 for(int i=0;i<4;i++) {LimbBuff[i] = 0;}
440 }
441 //-----------CLean Incoming serial buffer END--------
```

```
//------------Recieve Data from the HOST ------------
void RecieveData(){
if(Serial3.available()){

char check = 0;
                        AdHstLb = Serial3.read();
switch(AdHstLb){
  case 'A':
      for(int i=0;i<4;i++) {LimbBuff[i]=Serial3.read();}
        check = Serial3.read();
        if(check=='G'){
            for(int i=0;i<4;i++) { ToLimb01[i] = LimbBuff[i];}
                      check =0;
                      } else {
                           cleanBuff();
                           check = 0;
                        }
  break;

  case 'B':
      for(int i=0;i<4;i++) {LimbBuff[i]=Serial3.read();}
        check = Serial3.read();
        if(check=='H'){
            for(int i=0;i<4;i++) { ToLimb02[i] = LimbBuff[i];}
                      check =0;
                      } else {
                           cleanBuff();
                           check = 0;
                        }
  break;

  case 'C':
      for(int i=0;i<4;i++) {LimbBuff[i]=Serial3.read();}
        check = Serial3.read();
        if(check=='I'){
            for(int i=0;i<4;i++) { ToLimb03[i] = LimbBuff[i];}
                      check =0;
                      } else {
                           cleanBuff();
                           check = 0;
                        }
  break;
```

```
  case 'D':
      for(int i=0;i<4;i++) {LimbBuff[i]=Serial3.read();}
        check = Serial3.read();
        if(check=='J'){
            for(int i=0;i<4;i++) { ToLimb10[i] = LimbBuff[i];}
                      check =0;
                      } else {
                           cleanBuff();
                           check = 0;
                        }
  break;

  case 'E':
      for(int i=0;i<4;i++) {LimbBuff[i]=Serial3.read();}
        check = Serial3.read();
        if(check=='K'){
            for(int i=0;i<4;i++) { ToLimb20[i] = LimbBuff[i];}
                      check =0;
                      } else {
                           cleanBuff();
                           check = 0;
                        }
  break;

  case 'F':
      for(int i=0;i<4;i++) {LimbBuff[i]=Serial3.read();}
        check = Serial3.read();
        if(check=='L'){
            for(int i=0;i<4;i++) { ToLimb30[i] = LimbBuff[i];}
                      check =0;
                      } else {
                           cleanBuff();
                           check = 0;
                        }
  break;

  default:
  break;
}
                      }
}
//------------Recieve Data from the HOST END---------
```

```
//-------------Send Data to the HOST ----------------
void sendData(){

  Serial3.print("RF"); Serial3.print(" ");
     for(int i=0;i<5;i++) {Serial3.print(FromLimb01[i]);
                           Serial3.print(" ");
                            }Serial3.println(" X");

  Serial3.print("RM"); Serial3.print(" ");
     for(int i=0;i<5;i++) {Serial3.print(FromLimb02[i]);
                           Serial3.print(" ");
                            }Serial3.println(" X");

  Serial3.print("RB"); Serial3.print(" ");
     for(int i=0;i<5;i++) {Serial3.print(FromLimb03[i]);
                           Serial3.print(" ");
                            }Serial3.println(" X");

  Serial3.print("LB"); Serial3.print(" ");
     for(int i=0;i<5;i++) {Serial3.print(FromLimb30[i]);
                           Serial3.print(" ");
                            }Serial3.println(" X");

  Serial3.print("LM"); Serial3.print(" ");
     for(int i=0;i<5;i++) {Serial3.print(FromLimb20[i]);
                            Serial3.print(" ");
                            }Serial3.println(" X");

  Serial3.print("LF"); Serial3.print(" ");
     for(int i=0;i<5;i++) {Serial3.print(FromLimb10[i]);
                           Serial3.print(" ");
                            }Serial3.println(" X");

     Serial3.print("GS"); Serial3.print(" ");
     Serial3.print(yaw);    Serial3.print(" ");
     Serial3.print(pitch);  Serial3.print(" ");
     Serial3.print(roll);   Serial3.print(" ");
     Serial3.print(Sonar);  Serial3.print(" ");

     Serial3.println(" X");
}
//-------------Send Data to the HOST ---------------
```

```
572 //=============================================================================
573 //============================================ MAIN LOOP ================
574 //=============================================================================
575 void loop()
576 {
577
578   RPLID();
579   IMU();
580   getI2CData();
581   setData();
582
583   int dekt_t = millis() - freq;
584   if( freq > 50) { sendData(); }
585   freq = millis ();
586
587   sendData();
588   RecieveData();
589
590
591  }
592
593 //=============================================================================
594 //============================================ MAIN LOOP END =============
595 //=============================================================================
596
```

```
598 //=====================================================================================
599 //====== Quanterion Filters
600 //=====================================================================================
601
602         void MadgwickQuaternionUpdate(float ax, float ay, float az,
603         float gx, float gy, float gz, float mx, float my, float mz)
604         {
605             // short name local variable for readability
606             float q1 = q[0], q2 = q[1], q3 = q[2], q4 = q[3];
607             float norm;
608             float hx, hy, _2bx, _2bz;
609             float s1, s2, s3, s4;
610             float qDot1, qDot2, qDot3, qDot4;
611
612             // Auxiliary variables to avoid repeated arithmetic
613             float _2q1mx;
614             float _2q1my;
615             float _2q1mz;
616             float _2q2mx;
617             float _4bx;
618             float _4bz;
619             float _2q1 = 2.0f * q1;
620             float _2q2 = 2.0f * q2;
621             float _2q3 = 2.0f * q3;
622             float _2q4 = 2.0f * q4;
623             float _2q1q3 = 2.0f * q1 * q3;
624             float _2q3q4 = 2.0f * q3 * q4;
625             float q1q1 = q1 * q1;
626             float q1q2 = q1 * q2;
627             float q1q3 = q1 * q3;
628             float q1q4 = q1 * q4;
629             float q2q2 = q2 * q2;
630             float q2q3 = q2 * q3;
631             float q2q4 = q2 * q4;
632             float q3q3 = q3 * q3;
633             float q3q4 = q3 * q4;
634             float q4q4 = q4 * q4;
635
636             // Normalise accelerometer measurement
637             norm = sqrt(ax * ax + ay * ay + az * az);
638             if (norm == 0.0f) return; // handle NaN
639             norm = 1.0f/norm;
640             ax *= norm;
641             ay *= norm;
642             az *= norm;
```

```
644             // Normalise magnetometer measurement
645             norm = sqrt(mx * mx + my * my + mz * mz);
646             if (norm == 0.0f) return; // handle NaN
647             norm = 1.0f/norm;
648             mx *= norm; my *= norm; mz *= norm;
649
650             // Reference direction of Earth's magnetic field
651             _2qlmx = 2.0f * ql * mx;
652             _2qlmy = 2.0f * ql * my;
653             _2qlmz = 2.0f * ql * mz;
654             _2q2mx = 2.0f * q2 * mx;
655             hx = mx * qlql - _2qlmy * q4 + _2qlmz * q3 + mx * q2q2 +
656             _2q2 * my * q3 + _2q2 * mz * q4 - mx * q3q3 - mx * q4q4;
657             hy = _2qlmx * q4 + my * qlql - _2qlmz * q2 + _2q2mx * q3
658             - my * q2q2 + my * q3q3 + _2q3 * mz * q4 - my * q4q4;
659             _2bx = sqrt(hx * hx + hy * hy);
660             _2bz = -_2qlmx * q3 + _2qlmy * q2 + mz * qlql + _2q2mx *
661             q4 - mz * q2q2 + _2q3 * my * q4 - mz * q3q3 + mz * q4q4;
662             _4bx = 2.0f * _2bx;
663             _4bz = 2.0f * _2bz;
664
665             // Gradient decent algorithm corrective step
666             s1 = -_2q3 * (2.0f * q2q4 - _2qlq3 - ax) + _2q2 * (2.0f *
667             qlq2 + _2q3q4 - ay) - _2bz * q3 * (_2bx * (0.5f - q3q3 -
668             q4q4) + _2bz * (q2q4 - qlq3) - mx) + (-_2bx * q4 + _2bz *
669             q2) * (_2bx * (q2q3 - qlq4) + _2bz * (qlq2 + q3q4) - my) +
670             _2bx * q3 * (_2bx * (qlq3 + q2q4) + _2bz * (0.5f - q2q2 - q3q3) - mz);
671             s2 = _2q4 * (2.0f * q2q4 - _2qlq3 - ax) + _2ql * (2.0f *
672             qlq2 + _2q3q4 - ay) - 4.0f * q2 * (1.0f - 2.0f * q2q2 -
673             2.0f * q3q3 - az) + _2bz * q4 * (_2bx * (0.5f - q3q3 -
674             q4q4) + _2bz * (q2q4 - qlq3) - mx) + (_2bx * q3 + _2bz * ql)
675             * (_2bx * (q2q3 - qlq4) + _2bz * (qlq2 + q3q4) - my) +
676             (_2bx * q4 - _4bz * q2) * (_2bx * (qlq3 + q2q4) + _2bz *
677             (0.5f - q2q2 - q3q3) - mz);
678             s3 = -_2ql * (2.0f * q2q4 - _2qlq3 - ax) + _2q4 * (2.0f *
679             qlq2 + _2q3q4 - ay) - 4.0f * q3 * (1.0f - 2.0f * q2q2 -
680             2.0f * q3q3 - az) + (-_4bx * q3 - _2bz * ql) * (_2bx *
681             (0.5f - q3q3 - q4q4) + _2bz * (q2q4 - qlq3) - mx) + (_2bx
682             * q2 + _2bz * q4) * (_2bx * (q2q3 - qlq4) + _2bz * (qlq2 + q3q4)
683             - my) + (_2bx * ql - _4bz * q3) * (_2bx * (qlq3 + q2q4) + _2bz
684             * (0.5f - q2q2 - q3q3) - mz);
685             s4 = _2q2 * (2.0f * q2q4 - _2qlq3 - ax) + _2q3 *
686             (2.0f * qlq2 + _2q3q4 - ay) + (-_4bx * q4 + _2bz * q2) *
687             (_2bx * (0.5f - q3q3 - q4q4) + _2bz * (q2q4 - qlq3) - mx)
688             + (-_2bx * ql + _2bz * q3) * (_2bx * (q2q3 - qlq4) + _2bz *
689             (qlq2 + q3q4) - my) + _2bx * q2 * (_2bx * (qlq3 + q2q4) +
690             _2bz * (0.5f - q2q2 - q3q3) - mz);
```

```
692             // normalise step magnitude
693             norm = sqrt(s1 * s1 + s2 * s2 + s3 * s3 + s4 * s4);
694             norm = 1.0f/norm;
695             s1 *= norm;
696             s2 *= norm;
697             s3 *= norm;
698             s4 *= norm;
699 
700             // Compute rate of change of quaternion
701             qDot1 = 0.5f * (-q2 * gx - q3 * gy - q4 * gz) - beta * s1;
702             qDot2 = 0.5f * ( q1 * gx + q3 * gz - q4 * gy) - beta * s2;
703             qDot3 = 0.5f * ( q1 * gy - q2 * gz + q4 * gx) - beta * s3;
704             qDot4 = 0.5f * ( q1 * gz + q2 * gy - q3 * gx) - beta * s4;
705 
706             // Integrate to yield quaternion
707             q1 += qDot1 * deltat;
708             q2 += qDot2 * deltat;
709             q3 += qDot3 * deltat;
710             q4 += qDot4 * deltat;
711             // normalise quaternion
712             norm = sqrt(q1 * q1 + q2 * q2 + q3 * q3 + q4 * q4);
713             norm = 1.0f/norm;
714             q[0] = q1 * norm;
715             q[1] = q2 * norm;
716             q[2] = q3 * norm;
717             q[3] = q4 * norm;
```

```
725 void getGres() {
726   switch (Gscale)
727   {
728   // Possible gyro scales (and their register bit settings) are:
729   // 250 DPS (00), 500 DPS (01), 1000 DPS (10), and 2000 DPS  (11).
730     case GFS_250DPS:
731           gRes = 250.0/32768.0;
732           break;
733     case GFS_500DPS:
734           gRes = 500.0/32768.0;
735           break;
736     case GFS_1000DPS:
737           gRes = 1000.0/32768.0;
738           break;
739     case GFS_2000DPS:
740           gRes = 2000.0/32768.0;
741           break;
742   }
743 }
744
745 void getAres() {
746   switch (Ascale)
747   {
748   // Possible accelerometer scales (and their register bit settings) are:
749   // 2 Gs (00), 4 Gs (01), 8 Gs (10), and 16 Gs  (11).
750     case AFS_2G:
751           aRes = 2.0/32768.0;
752           break;
753     case AFS_4G:
754           aRes = 4.0/32768.0;
755           break;
756     case AFS_8G:
757           aRes = 8.0/32768.0;
758           break;
759     case AFS_16G:
760           aRes = 16.0/32768.0;
761           break;
762   }
763 }
```

```
766 void readAccelData(int16_t * destination)
767 {
768   uint8_t rawData[6];  // x/y/z accel register data stored here
769   // Read the six raw data registers into data array
770   readBytes(MPU9150_ADDRESS, ACCEL_XOUT_H, 6, &rawData[0]);
771   // Turn the MSB and LSB into a signed 16-bit value
772   destination[0] = ((int16_t)rawData[0] << 8) | rawData[1] ;
773   destination[1] = ((int16_t)rawData[2] << 8) | rawData[3] ;
774   destination[2] = ((int16_t)rawData[4] << 8) | rawData[5] ;
775 }
776
777
778 void readGyroData(int16_t * destination)
779 {
780   uint8_t rawData[6];  // x/y/z gyro register data stored here
781   // Read the six raw data registers sequentially into data array
782   readBytes(MPU9150_ADDRESS, GYRO_XOUT_H, 6, &rawData[0]);
783    // Turn the MSB and LSB into a signed 16-bit value
784   destination[0] = ((int16_t)rawData[0] << 8) | rawData[1] ;
785   destination[1] = ((int16_t)rawData[2] << 8) | rawData[3] ;
786   destination[2] = ((int16_t)rawData[4] << 8) | rawData[5] ;
787 }
788
789 void readMagData(int16_t * destination)
790 {
791   uint8_t rawData[6];  // x/y/z gyro register data stored here
792   // toggle enable data read from magnetometer, no continuous read mode!
793   writeByte(AK8975A_ADDRESS, AK8975A_CNTL, 0x01);
794   delay(10);
795   // Only accept a new magnetometer data read if the data ready bit is set and
796   // if there are no sensor overflow or data read errors
797   if(readByte(AK8975A_ADDRESS, AK8975A_ST1) & 0x01) {
798     // wait for magnetometer data ready bit to be set
799   readBytes(AK8975A_ADDRESS, AK8975A_XOUT_L, 6, &rawData[0]);
800   destination[0] = ((int16_t)rawData[1] << 8) | rawData[0] ;
801   destination[1] = ((int16_t)rawData[3] << 8) | rawData[2] ;
802   destination[2] = ((int16_t)rawData[5] << 8) | rawData[4] ;
803   }
804 }
```

```
1102        // Wire.h read and write protocols
1103        void writeByte(uint8_t address, uint8_t subAddress, uint8_t data)
1104 {
1105   Wire.beginTransmission(address);
1106   // Initialize the Tx buffer
1107   Wire.write(subAddress);
1108   // Put slave register address in Tx buffer
1109   Wire.write(data);
1110   // Put data in Tx buffer
1111   Wire.endTransmission();
1112   // Send the Tx buffer
1113 }
1114
1115        uint8_t readByte(uint8_t address, uint8_t subAddress)
1116 {
1117   uint8_t data; // `data` will store the register data
1118   Wire.beginTransmission(address);
1119   // Initialize the Tx buffer
1120   Wire.write(subAddress);
1121   // Put slave register address in Tx buffer
1122   Wire.endTransmission(false);
1123   // Send the Tx buffer, but send a restart to keep connection alive
1124   Wire.requestFrom(address, (uint8_t) 1);
1125   // Read one byte from slave register address
1126   data = Wire.read();
1127   // Fill Rx buffer with result
1128   return data;
1129   // Return data read from slave register
1130 }
1131
1132        void readBytes(uint8_t address, uint8_t subAddress,
1133        uint8_t count, uint8_t * dest)
1134 {
1135   Wire.beginTransmission(address);
1136   Wire.write(subAddress);
1137   Wire.endTransmission(false);
1138   uint8_t i = 0;
1139        Wire.requestFrom(address, count);
1140   while (Wire.available()) {
1141        dest[i++] = Wire.read(); }
1142 }
```