# ROS-based self-teaching robot arm

Bachelor Thesis

Submitted by Valentine Ezekiev

In fulfillment of the requirements of the degree

Bachelor of Science and Informatics

To be awarded by the

Fontys Hogeschool Techniek en Logistiek

Venlo, 16-06-2021

## Information Page

Fontys University of Applied Sciences.

Postbus 141, 5900 AC Venlo

Graduation Report

| | |
|---|---|
| Student Name: | Valentine Ezekiev |
| Student number: | 2971429 |
| Study: | Informatics – Software Engineering |
| Period: | February 2021 – June 2021 |

| | |
|---|---|
| Company Name: | GreenTech Labs (GTL) |
| Address: | Tegelseweg 255 |
| Postal Code + City: | 5912 BG Venlo |
| Country: | Netherlands |
| Telephone: | 08850 78422 |

| | |
|---|---|
| Company Supervisor: | Jan Jacobs |
| Supervising Lecturer: | Pieter van den Hombergh |

| | |
|---|---|
| Non-disclosure agreement: | No |
| Amount of words used: | 9,791 |

## Abstract

In this graduation project, GreenTech Labs want to create a crop-weed-removal robot-arm. However, they want to achieve this using modern tools, with a system that can help them automate and simplify the process of training custom robot arms like this. The student who was assigned to this project had to research into the hows and whys of this situation, and propose and build a design that lets GTL train custom robot arms using Robot Operating System 2 and various simulation tools.

After much research, trial and error, a system was designed that works in a contained way – it requires the setting of a small amount of hyper parameters and 1 robot-arm model, simulated in a program called Gazebo – and uses these things to create neural networks from the simulation that the real-life robot would use to perform its tasks, do path finding, etc.

The project is only halfway finished, with the system being able to map out all the areas of movement that a robot can do without bumping into itself. There is more work to be done after handing off is complete, but the project  itself serves as a solid proof of concept. Further real-life testing will still be carried out after this report, as it is still not fully done.

## STATEMENT OF AUTHENTICITY

Issued by the FHTenL Examination Board, September 2017

I, the undersigned, hereby certify that I have compiled and written this document and the underlying work / pieces of work without assistance from anyone except the specifically assigned academic supervisor. This work is solely my own, and I am solely responsible for the content, organization, and making of this document and the underlying work / pieces of work.

I hereby acknowledge that I have read the instructions for preparation and submission of documents / pieces of work provided by my course / my academic institution, and I understand that this document and the underlying pieces of work will not be accepted for evaluation or for the award of academic credits if it is determined that they have not been prepared in compliance with those instructions and this statement of authenticity.

I further certify that I did not commit plagiarism, did neither take over nor paraphrase (digital or printed, translated or original) material (e.g. ideas, data, pieces of text, figures, diagrams, tables, recordings, videos, code, ...) produced by others without correct and complete citation and correct and complete reference of the source(s). I understand that this document and the underlying work / pieces of work will not be accepted for evaluation or for the award of academic credits if it is determined that they embody plagiarism.

Name (in capital letters) VALENTINE EZEKIEV

student number: 2971429

Place / Date: Venlo, Netherlands - 14-06-202I

Signature: 

## Glossary

Gazebo - a 3-D simulation program that allows for the definition of joint objects and segment objects. It has settings for time-speed manipulation, physics simulation and so on. More on the topic here - http://gazebosim.org/

ROS2 Foxy - the newest distribution of the Robot Operating System 2. Used for robot programming and causing of headaches. More information here -https://docs.ros.org/en/foxy/#

Arudino board - a micro-controller board used for sending electrical signals via pins to hardware – like an electrical motor or lights.

SOM – Self Organizing Map. A neural network structure that organizes high-dimensional data to a low-dimensional field (generally 2d). Has many different uses.

Solidworks – a program used for modeling/engineering in a 3d environment. Most often used with mechatronis engineering.

## Table of Contents

## Table of Figures

## Index of Tables

## 1. Introduction

This document is meant to be the final culmination documentation that describes all of the work on this project. It will go over the context and setting, after which straight into all of the relevant work and conclusions done thus far. There will be 3 primary body chapters -

- Research and Analysis will go over the beginning of the project, the establishment of the requirements and the research that needed to be done into specific areas for the student responsible for this project to be able to competently make design and implementation decisions down the line.
- The Design chapter will go over the design choices and plans that were born in regards to what was learned during research and what is demanded by the project itself as requirements. As well as a justification for why plausible alternatives were shot down during this phase in place of the final choices.
- Lastly, the implementation phase will detail all of the work put into facilitating the previous research and designs, as well as all of the newly gained information and conclusions from this period.

These chapters are followed by a conclusion chapter that gives and much more condensed once-over on the whole project.

## 2. Context

The organizer of this graduation project is an organization called GTL(GreenTech Lab) and it does so in partnership with Fontys HS. GTL is an organization that generally focuses on (but is not limited to) research into technology for the agricultural and sustainability sectors - things like devices that help with tracking of crop health, animal populations, drones that simplify agricultural work, advanced storage methods for agricultural resources and etc. It has a firm partnership with Fontys, which is where it sources students for internships and graduation projects. One of the primary areas of interest for GTL is the field of robotics. The field of robotics faces a significant set of hurdles, such as - complexity in programming an actuating robot, machine learning, accessible programming and development tools and extendable systems and operations. A new and ever-growing in popularity tool that combats all of these complexities is something called ROS(Robot Operating System). ROS provides a universal "environment" in which it simplifies and allows for all manner of complex and robust robotics work. ROS allows for simulation of robots, programmatic setup of control schemes, using machine learning to teach robots complex things such as image recognition and precise actuation, linking up complex networks of separate robotic systems and the list goes on. ROS also comes in primarily 2 primary distributions - ROS1 and ROS2. For the sake of brevity, ROS2 is a more advanced and robust version of ROS1 that has recently become as accessible, usable and supported as ROS1, so we will disregard further mentioning ROS1 from now on. GTL has a vested interest in looking into ROS to see if it has practical application in its research and general-work, which is where this project comes in.

### 2.1 Project Problem Description

GTL wants to create a custom robot arm that is capable of removing weeds from crops. This robot arm will be attached to a roving-type vehicle platform, which is expected to provide power, transportation and sensory input for the arm itself. The arm is yet to be fully defined, but it's expected to allow for the removal of weeds in one of 2 ways - either via cutting the weeds or by electrocuting them. It's expected to be comprised of single-axis rotating motor joints (1 or more), inter-joint connective segments (1 or more) and a simple appendage/tip. More information about what these things specifically mean can be found in the Software Specification Document (SRS).

Added below are 2 images of what the final product could look like, as well as a proposed design for testing:

Figure 1: Example image of weed-removing robot application

Figure 2: Proposed Robot Arm Design For System Testing

The goal of this project is to train the robot arm to function in a desired way. More specifically, the student responsible has to create a dynamic system that can allow the robot arm to teach itself how to work, based on hyperparameters/rules/data set by a human agent. The system has to be robust enough to work with a variety of custom robot arms (that follow a defined generic structure, outlined in the SRS) that GTL may wish to make down the line. The goal here is to create an infrastructure that can save GTL many man hours of manual control-scheme setup and programming, that can also be extended for more complex tasks past the point of the graduation project.

# 3. Phase I - Research and Analysis

Phase I started out well – firstly with several meetings between the student, Jan Jacobs and Marcel Roosen. The scope and goal of the project were defined, requirements discussed and guidelines laid out.

## 3.1 Planning

As with any project, a plan going forward was needed. The project was separated into 3 main phases that would last set amounts of time.

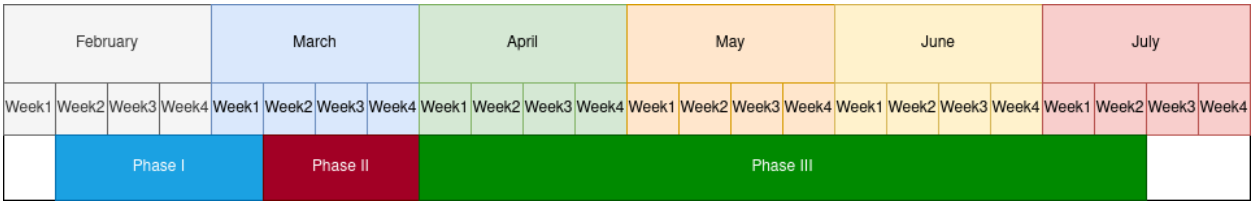

Figure 3: Project Phase Layout Planning

### *3.1.1 – Phase I*

The first phase would be the beginning of the project. This was where requirements are determined, goals are set, planning is made and any necessary research is performed. This is where the “what” and “why” are defined.

### *3.1.2 – Phase II*

In this phase, all of the requirements and goals set from the previous phase, as well as any research done, would be applied towards a design that fulfills the goals of the project. This is where the “how” of the project is planned out.

### *3.1.3 – Phase III*

This phase would constitute the application of the plans and designs from Phase II, as well as their modification should any changes arise due to the context.

### 3.2 Project requirements/specifications

The project was defined to work within the following scope and under the following conditions:

- This project is only concerned with the robot arm itself, it does not take into account power requirements, moving the entire arm itself around or any kind of visual information being provided for the robot arm by itself. It treats the robot arm as being in a vacuum where all necessary things for functionality are provided.
- A robot arm, in the confines of this project, is defined as:
  - Comprised only of single-axis rotating joints (without limitation to how many) – this means that ball joints, pistons and others are excluded from the scope.
  - Comprised of rigid, inter-joint segments.
  - Comprised of 1 last segment of the robot arm (its appendage) which would be counted as just a black-box type structure with a standard given area of effect relative to itself. This means an appendage that is simply turned off or on. This discounts any multi-joint appendages or complex mechanisms that require additional machine learning to operate.
  - Comprised of the following structure:

    Base Joint→Segment→Joint→Segment→Last Joint→Appendage.

    Any number of additional joints and segments can be added in, so long as any given joint that is NOT the Base Joint or Last Joint remains surrounded by segments.
- This project is required to use ROS2 for developing and testing itself. The project should prove the viability of ROS2 as a development tool for GTL.
- All code must be documented, as well as there must be documentation for all relevant hand-off procedures at the end of this project.
- Regular meetings should be kept with Marcel Roosen and Jan Jacobs to notify them of progress and relevant decisions that need to be made.

### 3.3 End-Goals and Deliverables

This is what was expected by the end, for this to be a fully complete project:

- A system that can train any type of robot-arm that fits within the defined structure standards of this project.
- A system that can train robots in a simulated environment to save on time, using ROS2 as the main training environment.
- A proof of concept that such a system can work.
- A robot-arm that has been trained to move and perform a task with the training system.
- A robot arm used for cutting/electrocuting weeds that has been trained with the developed system.
- Full documentation of code.
- Final report of the project.
- Full project documentation that allows for easy replication of final results.

### 3.4 Research

It was concluded, after collecting requirements and creating a complete SRS document(listed in the appendix), that the student's knowledge in machine learning was lacking. A research document was carried out and written to help the student better understand semi and unsupervised machine learning in the given context. This information was necessary so that it could be better understood how to design a system that would allow for the training of robot arms of varying types(which still maintain a specified, general mechanical structure).

The following primary question needed answering:

"How to create an environment that can teach a variety of similar robot arms to teach themselves how to actuate and perform tasks ,with set parameters, from scratch?"

In accordance with this primary question, there would be sub-questions, such as:

1. What is motor babbling?
2. What alternatives are there to it? Why should they be considered?
3. What variety/combinations of machine learning would give us the best results for such a system.
    1. Can this method be applied to a variety of robot arms?
    2. Does this method allow for motor babbling?
    3. Can this method allow for one robot to learn and copy from another robot's neural network if they share similar features in design?
    4. How long would such a method take a singular arm in a virtual environment? How would time consumption scale with degrees of freedom?
    5. How would the chosen method of machine learning be applied to the ROS2 environment? Does it even need to interact with ROS2?
    6. What parameters would be needed to set learning goals? What would be the most modular approach?
    7. Are there preferable alternatives?

Here are the important conclusions from that research:

- Baby-babbling is an excellent natural human process of semi-randomized establishment of the range of motion that a human actuation system can preform. Essentially, how the brain maps out a model of the body it is in for control purposes. This translates directly into motor-babbling - the same thing, just for robots.[3]
- Motor-babbling, by very definition, would not grant the desired amount of automation in this project. It can’t be used for skill learning, only for model-definition.
- There are multiple approaches to semi-supervised machine learning. And there are many methods of implementation. At their core, however, they can be seen as different flavors of optimization algorithms. A good example of this is Evolutionary Algorithms [11]. Evolutionary Algorithms is comprised of the general notion of having

a given "main" algorithm, mutating it into several variants, and testing out how each one performs. Then the best performing one becomes the "main" algorithm. However, this can be implemented to different effects - where one approach would yield the best singular and most efficient approach, another would yield the top few most optimal approaches, never pointing to one given final result.

The most considered approach from EAs was evolutionary strategies[5], [6], [10], [13]. On account of several convincing applications in a robotics environment, it really did seem like a plausible candidate machine learning in this project. However, after discussion with the stakeholders, it was deemed unnecessary compared to its competitors, simply due to the fact that at current time, GTL has no interest in setting up the infrastructure for this approach to be profitable.

- While there are many approaches to algorithmically teaching a system to perform given tasks, not too many are easily implementable within a ROS environment. Given the limited knowledge of the student, as well as practical and time constraints, it was determined that only accepted and practised in ROS machine learning algorithms would be used.
- Reinforcement Learning was chosen as the most preferred means of training a robot arm to teach itself how to perform a given task in ROS. It has a lot of history in training robots and fulfills the requirements of this project. [8] [12]
- The concept of applying the neural network of one robot to another robot poses too many complex problems. While 2 absolutely identical robots(from a structural perspective) can share neural networks, robots with differing structure could not adapt the neural networks of other robots that have similar structures, but not identical ones. Unfortunately, this means that the idea of robots sharing information and learning from each other, in the scope of this project at least, is deemed impractical.
- During the research process, it was concluded that one method of machine training would be not enough to achieve the desired level of self-teaching that the system needed to do. Reinforcement Learning alone would require too much pre-work. Motor-babbling doesn't work for training complex behaviours, by very definition. Using concepts from biomimicry, the student devised the idea for a system that replicates how humans teach themselves how to use their arms, first

in a general sense ( via baby-babbling and the Motor Cortex [17] ) and then in a specific, skill-based sense (manual learning of specific skills via the Cerebellum [18] ). Using motor-babbling, the robot would calculate all the movements it can do and generate a basic SOM neural network for these movements(as shown in reference [7], specifically chapter 3.4 of the book). This would require only basic hyper-parameters, such as the physical dimensions of the robot and its joint positions and angle restrictions. After that, Reinforcement Learning would pick up from there, and with a few rules and hyper-parameters set again, it can teach the robot to teach itself how to perform simple or even very complex actuation tasks.

The image below is meant to provide an overview of the theorized design/process for training robot arms, that resulted from Research. For the full information, reference the research document itself.

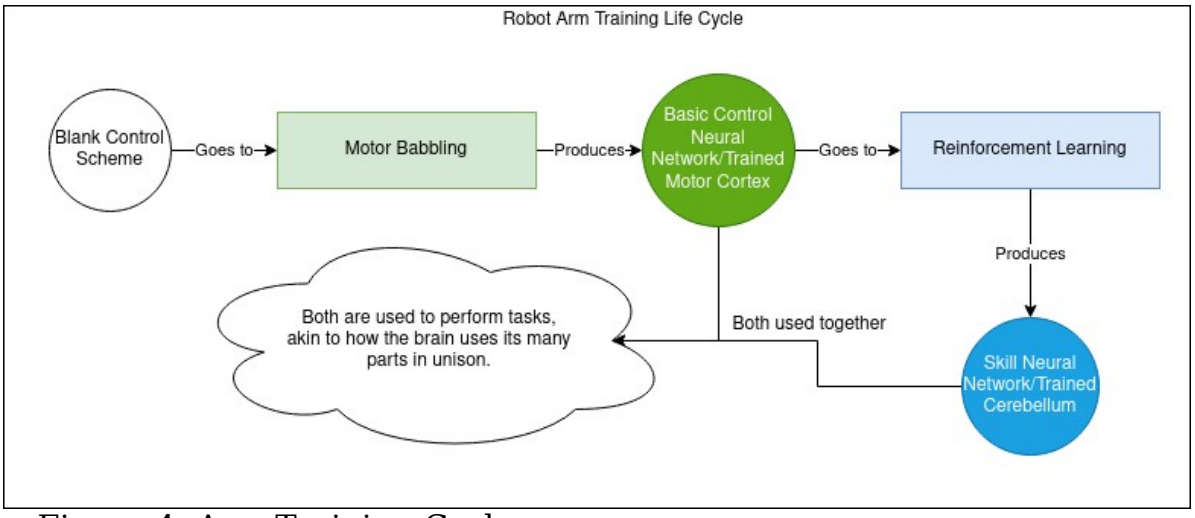


Figure 4: Arm Training Cycle

With the research document completed, as well as all other documents from Phase I, the student had a good, justified understanding of the theory behind what he needed to build and had a clear layout of what the system is supposed to be and what it requires. The information gleamed during this phase was confirmed and peer-checked during multiple meetings with the company coach, so as to ensure maximum quality of information.

# 4. Phase II – Design

This phase entails the designing of the self-teaching system using what was learned during Phase I. First, there was some experimentation.

## 4.1 Experimentation

When this Phase of work began, a lot of experimentation was done with various Self-Organizing Map packages and Reinforcement Learning Exercises to help the student more practically confirm the theory concluded from research. There were plenty of problems found during this, such as existing ROS2 Reinforcement Learning packages not being supported by the newest, used by the project, release of ROS2.

This was the source of some difficulty, as the student had virtually no reinforcement learning experience up to this point, and without the dedicated ROS packages, was left directionless. Fortunately the task was simpler than initially conceived, and after going through -> https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/

it became much clearer how to approach the reinforcement learning process from scratch and apply it in various environments.

As far as the motor babbling was concerned, the student used a package called MiniSOM to organize and visualize some basic data. The concept seemed to work just as desired, the data was organized in the way that research theorized it can be used for navigation and thus work began on drawing up a system.

## 4.2 Calculation over Simulation

### *4.2.1 Calculation Approach*

Version 1 of the system went with a “calculation over simulation” approach. What this means is that, there was a choice to be made – either the student could make the entire motor-babbling part work with just mathematical variables and very robust mathematical algorithms OR use a 3d model and physics simulation program with more minimal algorithmic requirements.

Initial attempts with using only the mathematical approach looked promising – they boasted absolute direct control over every “mathematical simulation” variable. However, the head of unnecessary complexity began to rear itself.

The image below was the student's attempt annotating all of the necessary information that just one joint would need to keep track of in the "mathematical simulation". Relative rotation values, XYZ positioning, collision space – all of these needed to be separately accounted for.

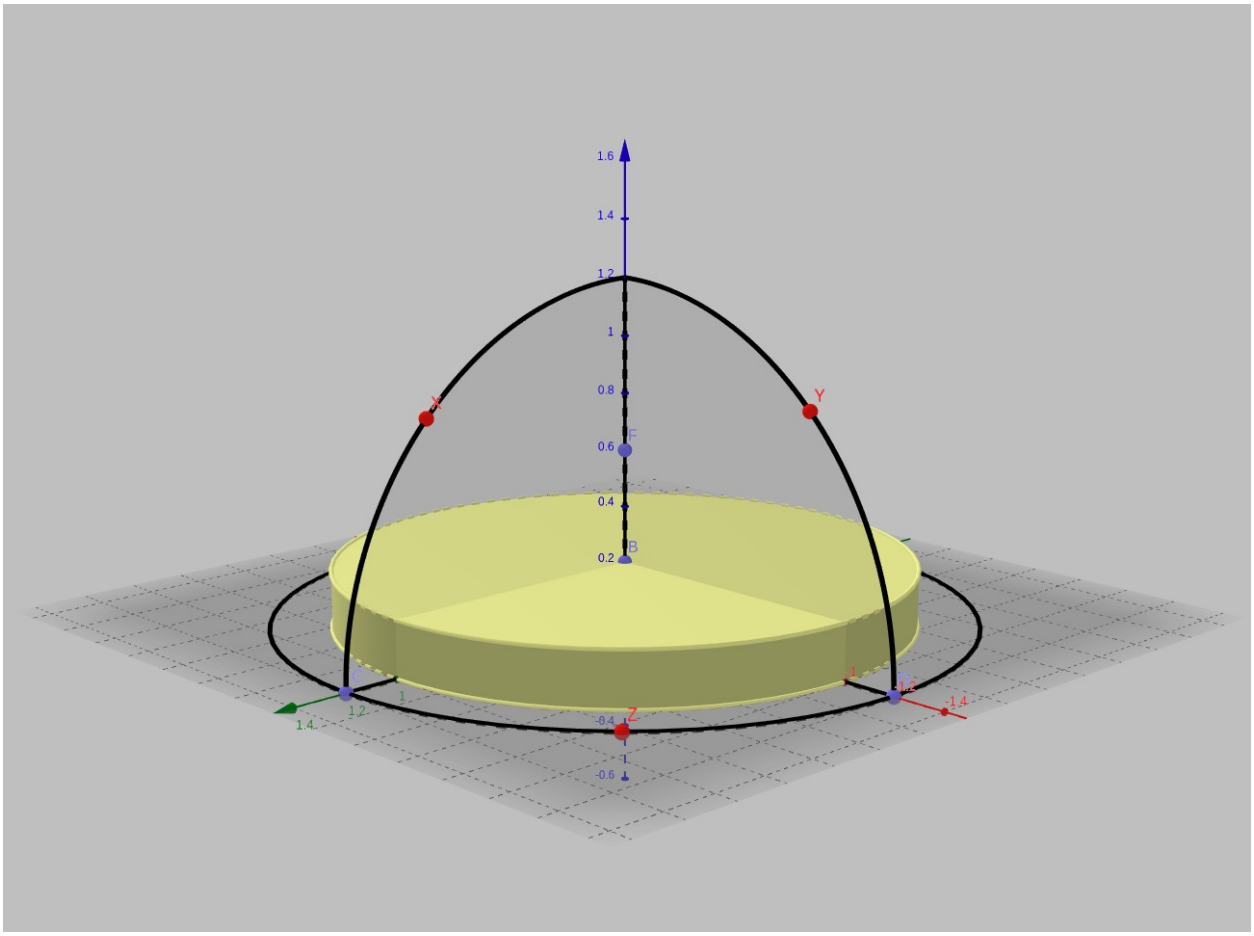


Figure 5: Attempt at mathematically annotating a 3-dimensional joint in a purely mathematical environment.

The same went for static inter-joint segments – they had to have their XYZ, collisions and physics accounted for as well. One of the most glaring problems for such a design would be how inhospitable it is for the end user. Most mechatronics engineers use programs such as Solidworks to do their modeling work. These are widely used and known programs that are designed to allow people to smoothly and efficiently engineer models. To ask them to learn an entirely custom new layout just to mathematically model a robot arm, with at least a few dozen variables would be a very jarring experience. There isn't much doubt that they could, but it would simply be an inefficient usage of time and an introduction of unnecessary

steps that have already been automated with the previously mentioned Solidworks example.

Another flaw to this approach would be that, while it certainly can be done, the math is there – it would be, in a sense, re-inventing the wheel. There already exists simulation software that solves all the problems that this approach poses. And on top of that, the complexity would cost the student a lot of time, as he does not have the necessary mathematical knowledge to quickly tackle such a complex problem.

### *4.2.2 Simulation*

On the other side of this argument, exists an application called Gazebo. Gazebo is a simulation software, that essentially allows for the simulation of collision, gravity, motor turning resistances and many, many more variables. It automates and already does most of the work for us. The only real downsides to it are that it can be unstable at times and can be complex to learn initially. However, it does already have ROS2 integration anyway and it allows for the importing of Solidworks models, so by comparison, it doesn't really have a significant enough downside to itself versus the alternative.

### *4.2.3 The Conclusion*

To summarize, between the choice of manually setting up the entire simulation with math in (for instance, a jupyter notebook environment) or just using the pre-existing tools for such a task, while in theory the purely mathematical approach would boast ultimate efficiency, the practical hurdles to achieving this efficiency would be very considerable. And on top of that, such a mathematical infrastructure wouldn't lend itself well to integrating with pre-existing engineering methodologies.

*Table 1: Comparison table for Simulation versus Calculation*

| Comparison Criteria | Gazebo | Mathematical Simulation |
|---|---|---|
| Already Integrated with ROS2? | YES | NO, would have to be built from ground up |
| Is complex to use? | Only initially | Always, on account on having very many variables |
| Integrated to work with Solidworks? | YES | NO, would require remodeling of entire robot arm from scratch |
| Time Required to implement? | Approx 2 weeks | Upwards of 1 month |
| Performance Speed | With time acceleration, only limited by hardware | Theoretically superior, and certainly more efficient |

In accordance with these criteria, the simulation approach with Gazebo was chosen.

### 4.3 Finalized Design

As mentioned above, the proposed design for the overall system architecture would be based on biomimicry. The system would be comprised of 2 main parts:

1. Part one would take a robot model, one made to be rendered in Gazebo*(Simulation Software), and using a combination of ROS2 and Gazebo packages, run a loop that has the robot perform every possible combination of movements that it can. Every single one of these position combinations would then get saved as a state object. This state object would have the X, Y and Z coordinates of the tip of the robot arm, as well as the turn value of every turning joint of the robot arm. This data would then be fed into a SOM, which after being properly trained, could be used for logical and basic path planning(for more information on why, check research document).

   This would result in the Motor-Babbling-generated neural network for the robot that would be used for part 2 of the system.

2. Next would be the reinforcement learning - using the data from before, and defining a rule set, the simulation would use its basic path planning to teach itself how to achieve given goals. Once again - just Gazebo and ROS2 packages, with perhaps a python package to help with the reinforcement learning process. With the iterative training, a neural network would be produced that the robot could use to in conjunction with it's motor-babbling SOM for full mobility and task completion.

The whole system, once complete, would be used to take any custom robot arm(that still sticks to specifications, as mentioned before) and train it to do basic movements first, and then complex sets of basic movements in specific succession. Below are 2 sub-chapters to go into the farther design details of the 2 parts. Also is given the domain model of the overarching system, for viewing purposes, as well as a general overview/flow diagram to help visualize how the whole thing comes together.

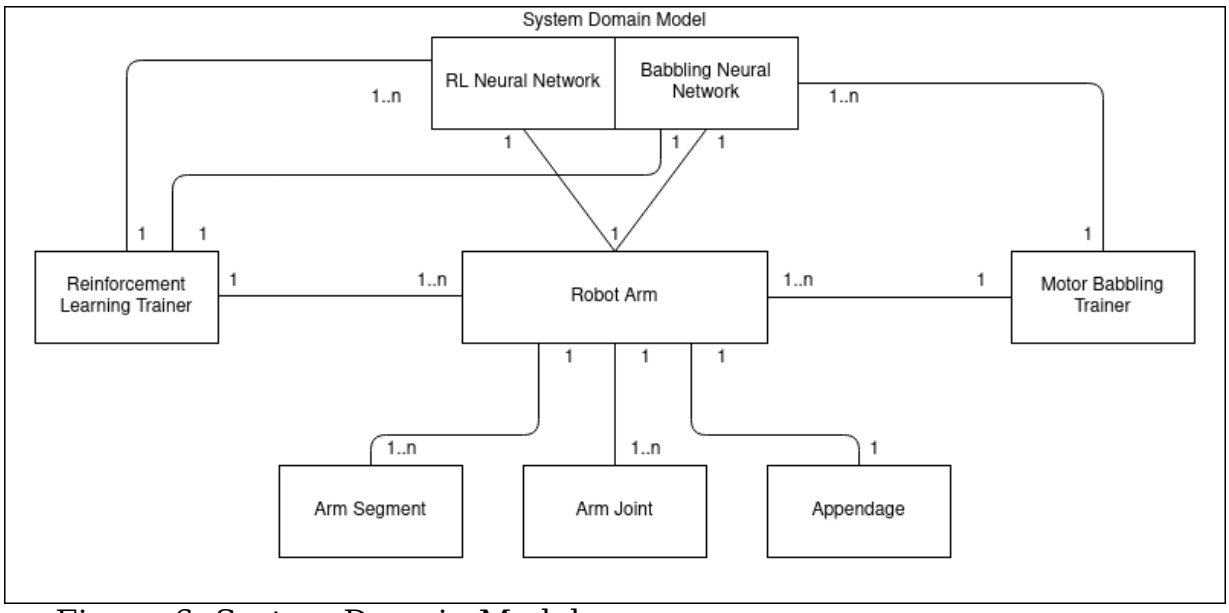


Figure 6: System Domain Model

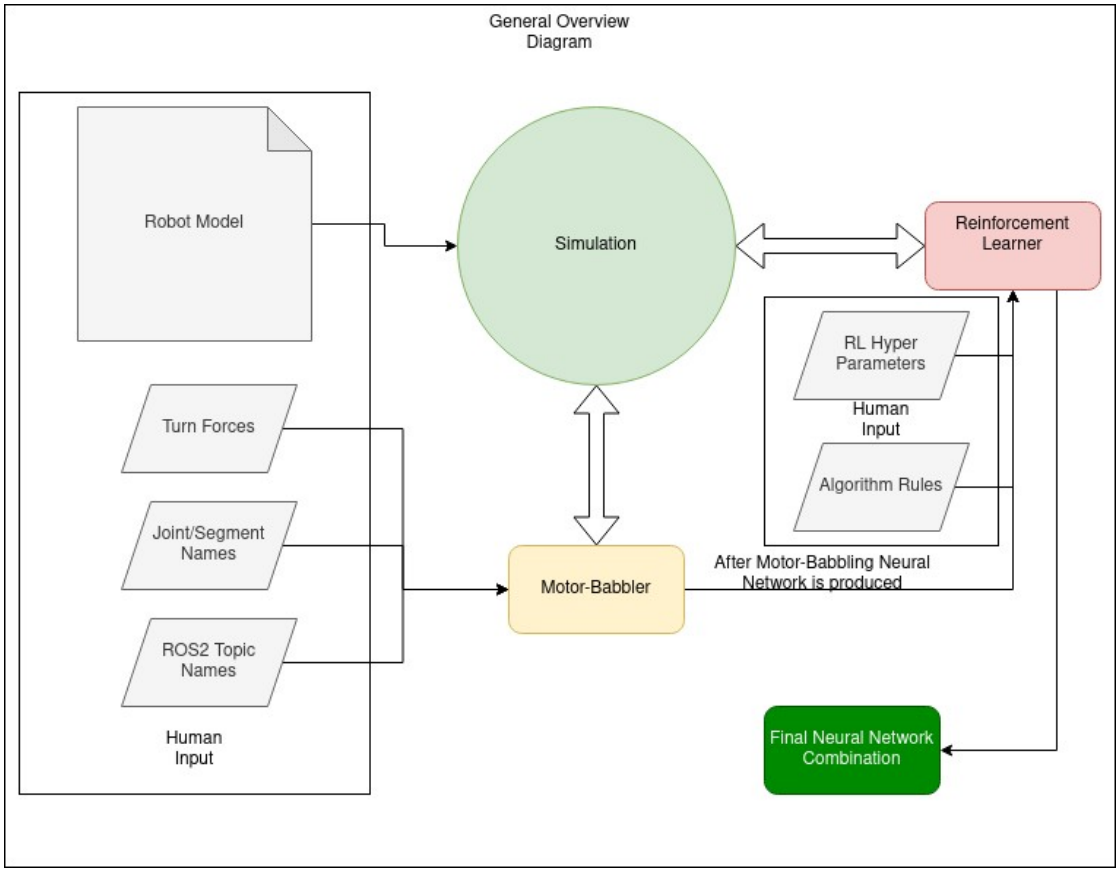


Figure 7: General Overview Diagram - just to aid with understanding how everything comes together.

### 4.3.1 Motor Babbling

Initially, the motor-babbling aspect of this project was meant to be done without simulation software and entirely done using mathematics. However, with the fact that there would be robot arms that often likely work in 3 dimensions and not 2 or 1, the complexity of this approach quickly sky-rocketed. Due to practical constraints and a streamlining of the approach, it was replaced with the model below:

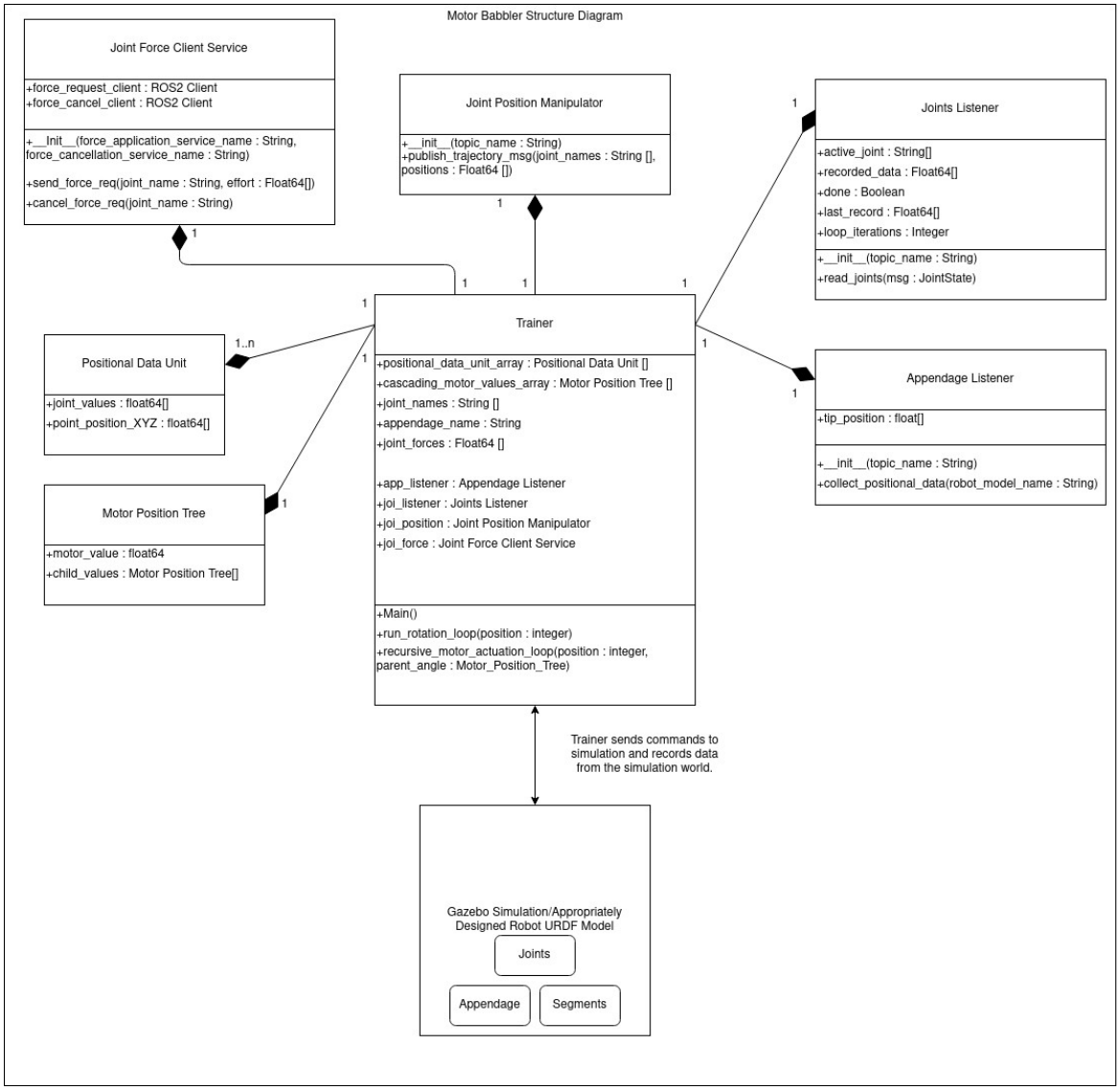

Figure 8: Motor Babbler Structure Diagram

This entire program would essentially be a ROS2 package. It would be written in python and consist of the following key features:

- There would be 2 "Listeners" - in ROS2 terms, listeners and publishers are sort of the main function of everything - sending ROS2 messages all around. This will be explained in greater detail during the Implementation chapter. The point of these listeners would be to collect joint turn data and the robot appendage tip's XYZ positional data from the simulation.

- There would be 2 joint manipulator classes - the "Joint Position Manipulator" would be responsible for spawning joints in specific positions within the simulation – what is meant by spawning, is that a joint in a gazebo environment can be told to, instantaneously, have itself turned to a specific degree. So for example, if we have a wheel that turns only 90 degrees along one axis away from a defined central point, this wheel can suddenly turn from degree 40 to degree 90 instantly. This is gone into in more detail during implementation.

  The other class, "Joint Force Client Service", would move joints around via the application of simulated force upon them. This may seem redundant, however due to problems with lacking functionality in ROS2 and Gazebo, as of the time of this project, this was the most elegant solution. This will be delved into in greater detail, again, during the Implementation chapter, for now - the important thing is that these 2 packages would be used together to achieve the best possible simulation result.

- There would be 2 "Central" data structures, although there would be many more in-between. Motor Position Tree would be a nested type of class, where 1 motor value from motor 0 could have 200 positions of motor 1 and so on. Positional Data Unit is simply the final data holder of all of the combined motor positions and resulting tip XYZ data - the object that would be used for the neural network training. More detail in chapter III.

- There would be the Gazebo simulation itself, which would have a robot model with joints and segments in between, and a bunch of ROS2 compatible packages that would allow it to communicate with the Gazebo environment. To be used for collision, physics and movement simulation.

- Lastly ,but not least, all of these elements combine in the Trainer class, which would use them to talk back and forth with the simulation. It would be intended to, with just a few set

hyperparameters, take on any simulated robot, that fits general specification, and make it do all of its movements. After that, it would take all the generated data, train a Self-Organizing Map and briefly test it to see if path finding makes sense.

All of this would come together to complete the motor-babbling aspect of the program and allow for the next big thing - the Reinforcement Learning.

### *4.3.2 Reinforcement Learning*

As of writing this, the Reinforcement Learning part of this system would in many ways be simpler to implement than it's predecessor. The RL loop needs to deal with no uncertainties about the shape of the arm, where it can move and where it can't, and since all of this information is saved in the SOM path finding - a lot less functionality is necessary. It is essentially a stripped down version of the Motor Babbler, with a different algorithm attached. So there is no need to re-explain what was already gone over before, to summarize it's work loop - it would take the SOM and, following a set of rules for its reinforcement learning loop, it would keep moving the arm around and around in 3d space and recording the results, while constantly checking if the patterns established are favorable for the RL loop or not. In simpler terms - it'll just be RL with a 3d model moving around.

The simplest example work for this would be to define two 3d points in space, in a given succession, and telling the algorithm to find the shortest path to move between these 2. Or maybe giving an additional data point which highlights certain areas of 3d space that shouldn't be passed through under specific circumstances - this could simulate objects in the way, terrain elevations - really there is a lot that can be done from here.

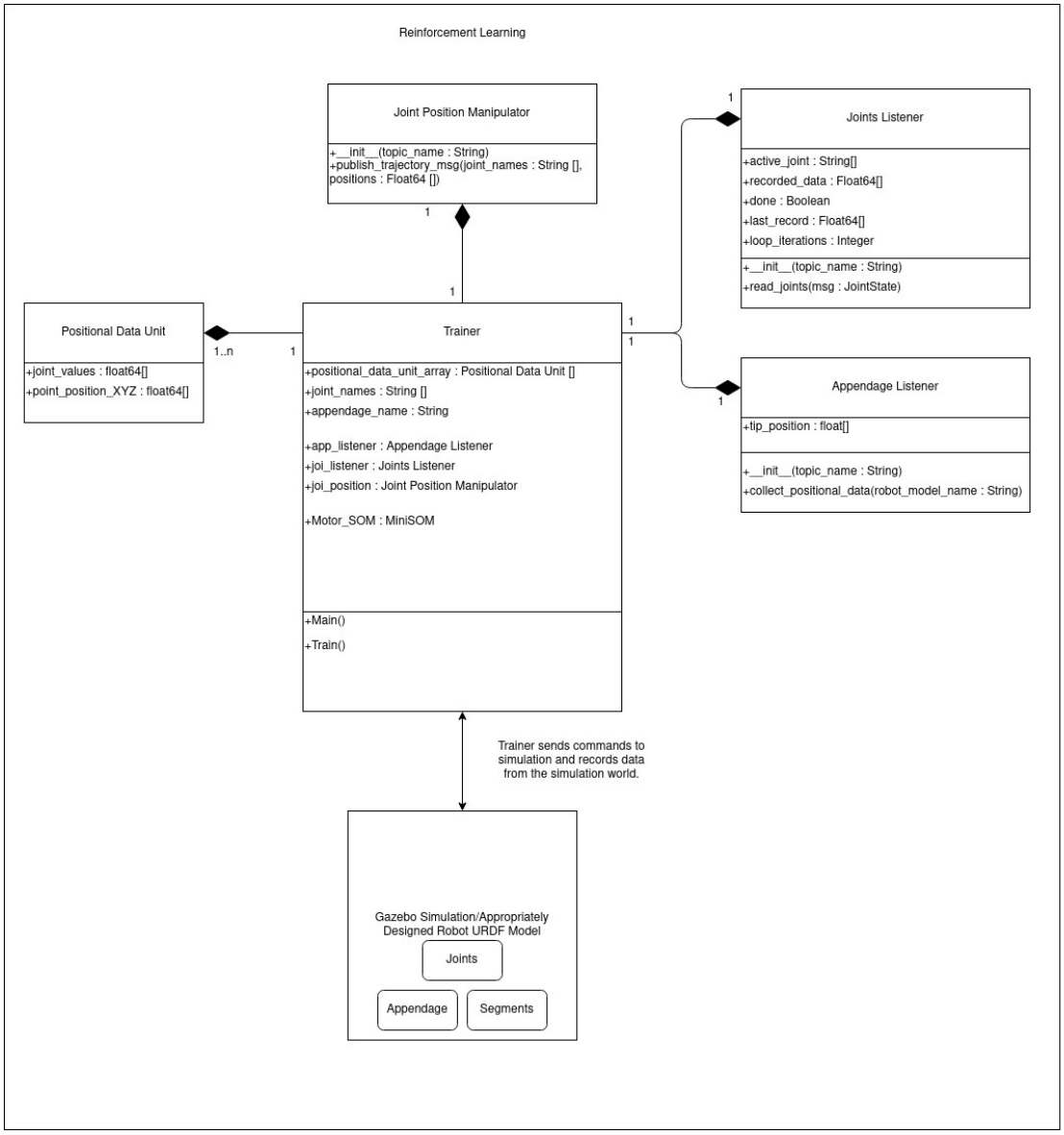

Figure 9: Reinforcement Learnier Structure Diagram

From this point onward, the system would have everything needed to have a basic robot arm perform rudimentary or even some complex maneuvers. The neural networks would likely be put on a micro-processor board of some sort, or perhaps just a normal portable computer, from where there would need to be made a simple control scheme from the ROS2 environment that matches the movements, ratios and trajectories of the simulation with their real-life robot counterpart. This would vary from case

by case, but it can be safely said that any experienced professional can knock out this kind of work in an afternoon.

In addition to the graphs and system plans, a Design Justification document was written - the purpose of which was to give commentary on all major design choices for the finalized system, so that anyone working on this project after the student is long gone can find easy answers on the most major topics as to the "Why?" of things.

# 5. Phase III - Implementation

This part took up the largest portion of the project due to 2 primary categories of factors:

- Lack of experience with ROS2 Foxy, Gazebo, Python and robotics in general.
- A plethora of issues rooted entirely in the fact that ROS2 Foxy and its Gazebo counterpart packages are severely lacking, due to how new the distribution is.

As of writing this, the project has a nearly fully completed and functioning Motor-Babbling trainer and a yet to be live-tested robot. This chapter will go over all of the work and hurdles encountered during the Motor-Babbler implementation and the results and successes thus far.

## 5.1 Motor Babbling

After planning was squared away, practical work began. A ROS2 package would need to be made to work alongside the Gazebo simulation, first we will discuss the Gazebo side of things and then go on to how the ROS2 package took shape.

### *5.1.1 Gazebo*

As mentioned in the Terms chapter, Gazebo is a program used for the physics-based simulation of basic 3d objects. It primarily supports 2 kinds of object:

- A raw joint that can be one of several defined gazebo joint types(static joint that can’t turn, joint that turns along one or more axes, etc.) or just a custom one altogether. These joints can account for collision, they can simulate friction, weight, turn limitation, force dampening, total force limitation - the list goes on, but these are the important essentials. These joints are used to simulate our robot's joints
- Segments/Shapes/Static Physical Objects that are just that - a solid geometrical object that can have collisions, colours, physics, gravity, weight, frictions and so on.

In a Gazebo simulation, a robot would be designed or imported, that fits the structural specifications of the SRS. The robot would be an unrestricted amount of joints (small sidenote, 10 000 joints would actually be a limiter, because joints are explored via recursion in the training program, and Python can only support 10 000 layers of recursion before giving out – it’s

very unlikely to ever make a robot like that, but still worth mentioning), starting with a grounded joint 0, or Base Joint, at XYZ position 000(in the simulation). This joint would then follow a pattern of being connected to a segment, which in turn would connect to joint 1, and so on and so forth until the final joint is defined. The segments in between can be of varying shapes and sizes, given that they don't defy the laws of physics. At the very final segment, at its tip, would be attached the appendage of the robot arm. This appendage, from a programmatic point of view, would consist of one XYZ point in 3d space, relative to the starting ground 0 of joint 0. It is important to note that every joint should only be able to turn along 1 axis, as the system was not designed for pistons or ball joints. Not to say that it's impossible to do this, but this was not the intended design for this system.

The image below is a simple example of a 2-jointed simple arm model, meant to simulate the design shown at the top of this document. It remains functionally identical to the one shown above - this is one of the advantages to Gazebo, there is no need to perfectly replicate a model, since we have so much control over the physics of everything. If this model perfectly replicates all of the movements of the real-life counter-part, then that's all the trainer needs to make a SOM.

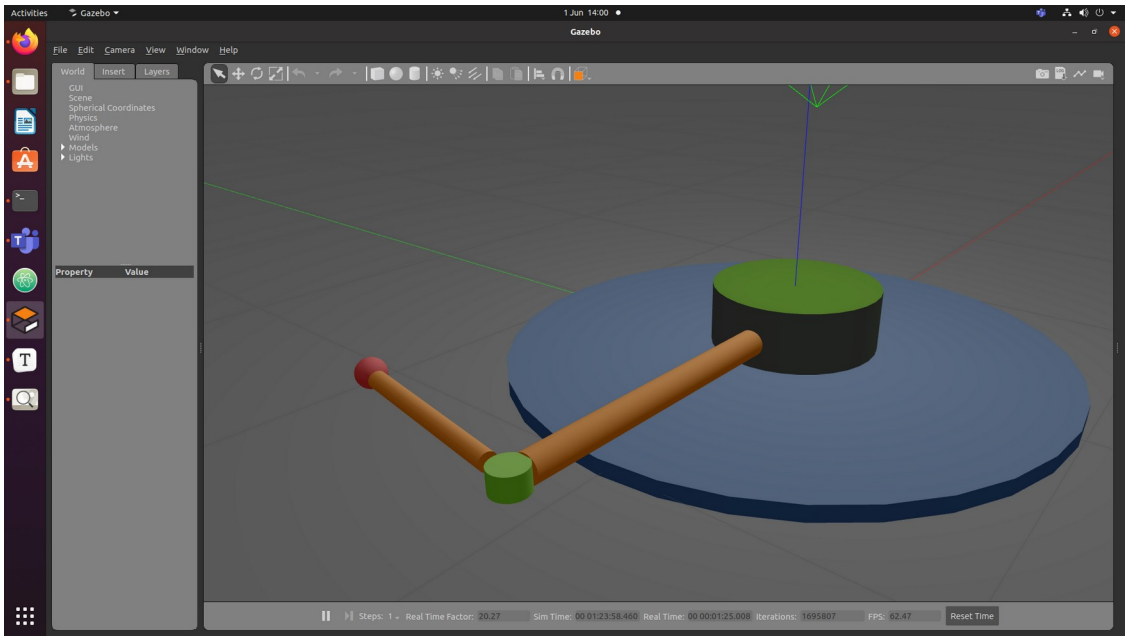

Figure 10: Gazebo Example Model

With a robot arm in place in the simulation world, generally it would be set in the Gazebo world file(a large file that accounts for all the objects and settings of the simulation, in XML format) that time flows 20 or 30 times

normal speed. One of the great features of Gazebo are that it allows for the speeding up of world time without any detrimental physics side-effects on the simulation. The only limiting factor on how far this can scale, based on the student's personal experiences with the program, is how powerful the working PC is.

### *5.1.2 Gazebo Hurdles*

The described information above should make for a simple approach to simulating, however this is where problems started popping up. Aside from an ample lack of almost any up to date documentation, we have the following:

- Gazebo joints have a tendency to clip badly outside of their physical limitations, if they are all not set to a specific weight.
- Gazebo doesn't currently allow for a way to programatically lock a joint, meaning they will always succumb to gravity and inertia.

There are more issues to describe, but the rest are more-so engaged to the ROS2 Foxy side of things than Gazebo itself. The workaround for these and the other issues is the following - set all joints to at least 1kg of weight, disable gravity and make it so that every single joint and segment are about half as heavy as their predecessors. This last part has to be done to negate the effects of inertia. Since we have no way of locking a joint, much like a joint would naturally do in real life when not turning, we go around this by making each following joint to weak to be able to affect it's predecessor. Again, to emphasize the point – this is a workaround and by no means a perfect solution. Simply the best way to get around the problems created from working with under-developed software.

Next we discuss ROS2, how it interacts with Gazebo, the problems this spawns and the workarounds.

### *5.1.3 ROS2*

The ROS2 program would be the "meat and potatoes" of the motor-babbler as it were - the package was meant to communicate back and forth with the simulation to gather joint and XYZ data, to put into a SOM that can be used for path planning either in the simulation or real life. To do anything, first we must discuss how ROS2 would communicate with the sim - gazebo_ros_api_plugin. This describes a family of packages, in this case designed to work with the newest release Foxy, that allow ROS to be able to send messages from its environment to the simulation to achieve different goals:

- gazebo_ros_state - this plugin would create a ROS2 service, which the ROS2 environment could use to get the pose and twist data of links and models.
- gazebo_ros_joint_state_publisher - a separate plugin for getting the pose and twist data of joints.
- gazebo_ros_joint_pose_trajectory - a package that allows ROS2 to immediately spawn a joint to be turned in a specific angle (as long as the angle isn't beyond the defined limit for that joint). This spawning ignores collisions and can cause errors.
- libgazebo_ros_force_system - this package is used to apply a variable force over a period of time on a ROS2 joint. It is not especially useful, because it is more akin to spinning a tire that is off the ground than telling a motor to turn a specific amount. You can define an amount of force to be applied, and the time for which it is applied and after that time is up, the joint won't stop turning, it'll simply stop applying that force. However, a useful feature of this plugin is that it accounts for collisions. So if a joint is swinging around with a link attached, it will stop and collide if an object is in the way(provided the link has collision enabled.)

#### 5.1.3.1 Listeners

These 4 (poorly documented) plugins make up the core of the feature work for the Motor-Babbler, since they define how everything has to be structured. For example, initially it was planned to use one general purpose listener that gets all positional data from the simulation, however since that hasn't been made yet - 2 separate ones had to be made for links and joints respectively. The details of the listeners aren't very useful to describe, the short version is - you can have them run just for one "tick" or continuous line of "ticks" where the listener would collect ROS2 messages and extract useful data from them. For example :

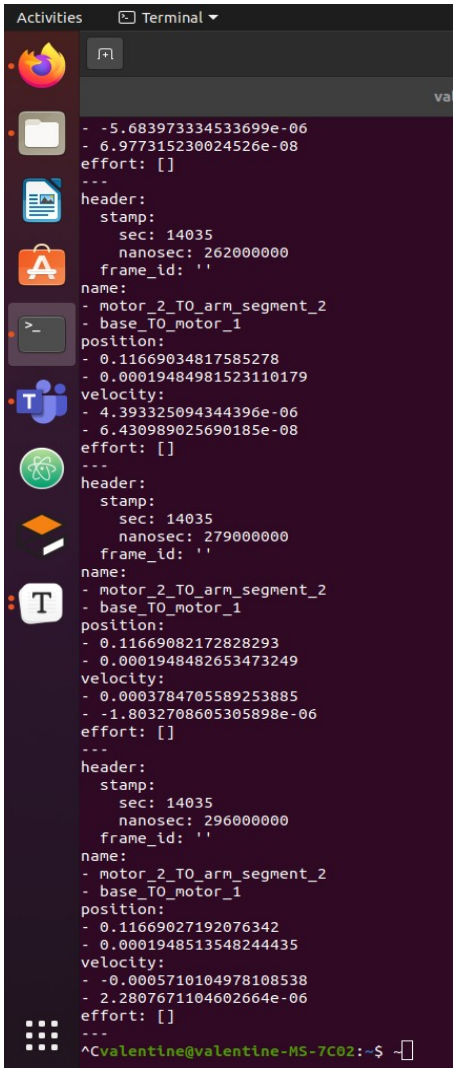


Figure 11: ROS2 Joint Message Structure Example

This is an example of a stream of joint messages - you can see that there are only 2 important fields for this project "effort" and "position". When the joint listener would be running, it would collect the data from the "position" field. So if we have a double jointed robot in a certain position, we would take the position of both joints, the XYZ of the appendage with our appendage listener, and we would have a complete state of the robot.

#### 5.1.3.2 Joint Manipulation

To move on to joints - they are incapable of stopping or locking on command, primarily due to the fact that gazebo_ros_pkgs still is in development and lacks a lot of features. As mentioned above, we only have 2 options for movement - either we spawn joints in specific positions(which means we simply make the joint instantly turn to a specific degree on its defined turn-axis) and risk collision breakage, or we apply force over a period of time and completely lose the ability of any precise movement. Eventually ,there will be made packages that provide much better movement options, but for the purposes of this project, a workaround was devised - when the simulation starts, it systematically goes from joint 0 all the way down to the final joint using recursion. When the trainer first reaches a joint it hasn't seen before, it would spawn this joint towards its outermost possible turn angle (either in the positive or negative value, it doesn't matter a lot) and then apply a turn force for the joint to swing around until it reaches its natural limit or hits a collision.

All of the positions that the joint was in during its turning are recorded except for the last few positions before the stoppage. This data is saved and the algorithm goes on to the next joint. What it then does, is for every previously recorded position, it will spawn joint 0 there and do the same "reset and swing" routine for the next joint, in this case - joint 1. So on and so forth, until the system determines and records every possible movement that doesn't have a collision.

#### 5.1.3.3 Recursion

One significant hurdle for the motor-babbler was dealing with a dynamic amount of joints. Part of the specification for this project is that a robot can have any number of joints. This would inevitably result in very messy and convoluted “if-statement” avalanche, so recursion was used instead. Below is given a code segment from the motor babbler.

To summarize it succinctly, when this loop fires for the first time, it sees that there is absolutely no data about the robot model from the simulation. It then takes the Base Joint(the very first joint in the robot model) and tells it to rotate smoothly from its right-most extreme angle towards its left-most. It is important to mention that whether it starts from the left or right isn’t especially important for any reason, this can be switched on the fly via hyper-parameters.

After the first joint has all of its rotation angles recorded (so, for example – all the positions between degree 90 and 180), the method terminates and fires again from the original for-loop that runs it. This time, it sees that the first joint has been calculated, so what it does is it starts a for-loop. From every recorded position of the Base Joint, the algorithm calls itself again and then sees if the next joint has had its data collected. If yes, the loop repeats and continues a layer deeper. If not – the joint in question has its movement calculated and the recursion terminates, before starting again and again, until all joints (listed in the hyper-parameters) are accounted for.

```
def recursive_motor_actuation_loop(position, parent_angle):

    # This loop fires at the beginning, when we have nothing in the array.
    if len(cascading_motor_values_array) < 1:

        run_rotation_loop(position) # Calculate all the mobility of joint 0/the base joint.

        for motor_position in joi_listener.recorded_data:   # Transform every position value into a Motor_Position_Tree object and append to the cascading_motor_val
            cascading_motor_values_array.append(motor_position_tree.Motor_Position_Tree(motor_position,[]))
        joi_listener.recorded_data = [] # Clear the recorded data from the listener.

    else:
        # Following the startup, one of 2 things happen - either we go another level deeper in our chain, depending on how far the simulation process is,
        # or we begin to calculate the currently pointed to joint since it itself doesn't have all of its positional data figured out yet.
        if position > 0:

            if len(parent_angle.child_values) == 0:    # If the current array isn't full, it's time to start recording motor turn data.

                run_rotation_loop(position)     # Runs the turning loop and records data.

                for motor_position in joi_listener.recorded_data:
                    parent_angle.child_values.append(motor_position_tree.Motor_Position_Tree(motor_position,[])) #Saves the data.
                joi_listener.recorded_data = [] # resets our joi_listener

                # the return statement breaks the recursion from continuing unnecessarily.
                return None

            else:

                for child in parent_angle.child_values: # We go even deeper until we find the empty layer we want to simulate next.

                    joi_listener.get_logger().info('Moving ' + joint_names[position] + ' to angle: ' + str(child.motor_value))
                    joi_position.publish_trajectory_msg([joint_names[position]], [angle.motor_value])
                    recursive_motor_actuation_loop(position+1, child)

        for parent_angle in cascading_motor_values_array: # This loop fires off whenever we're in position 0 of the recursive training loop.

            joi_listener.get_logger().info('Moving ' + joint_names[position] + ' to angle: ' + str(parent_angle.motor_value))
            joi_position.publish_trajectory_msg([joint_names[position]], [parent_angle.motor_value])
            recursive_motor_actuation_loop(position+1, parent_angle)
```

Figure 12: Code segment from motor-babbler, an example of recursion used to simplify a complex task

### *5.1.4 SOM*

A recap of what we've achieved so far:

- The ROS2 system is capable of tracking all(relevant) information in the simulation – joint turn values, segment XYZ positions, etc.
- The ROS2 system can interact with the joints in the simulation to make the robot arm move around.
- The ROS2 system has a hefty recursion-based algorithm which allows it to dynamically account for any amount of joints in a robot model, that is below 10 000 at least.
- The Gazebo environment is running anywhere between 2 or 30 times faster than normal time (or even more, given a powerful enough computer).
- The Gazebo environment simulates collisions.

With all of this, our data generation part is completed. The simulation does all the possible movements it can and records and organizes the numbers into a giant array. Each array entry consists of all the joint values plus the XYZ of the arm appendage tip at that time. One of these entries is called a robot state.

The ROS2 package then takes all that data, and feeds it into a SOM package (in this case, Somoclu) for training and organization. The main goal is to organize the neurons of the SOM in a way such that, if we want our robot to move from XYZ(1,1,1) to XYZ(1,4,1), it will simply jump from corresponding neuron to neuron between these 2 points, in the shortest way possible, to reach its goal.

Figures 9 and 10 showcase all of the XYZ positions of the appendage. This highlights all of the space that the robot arm can move within. If you recall from Figure 7, the model mimics (not fully) in structure and turn capacity what the human arm can do. If you take your own arm, and limit it to only turning your shoulder from left to right and only extending and retracting your elbow from left to right, you will notice that you more or less recreate the same pattern of movement as shown below.

Figure 13: Data Visualization from the top

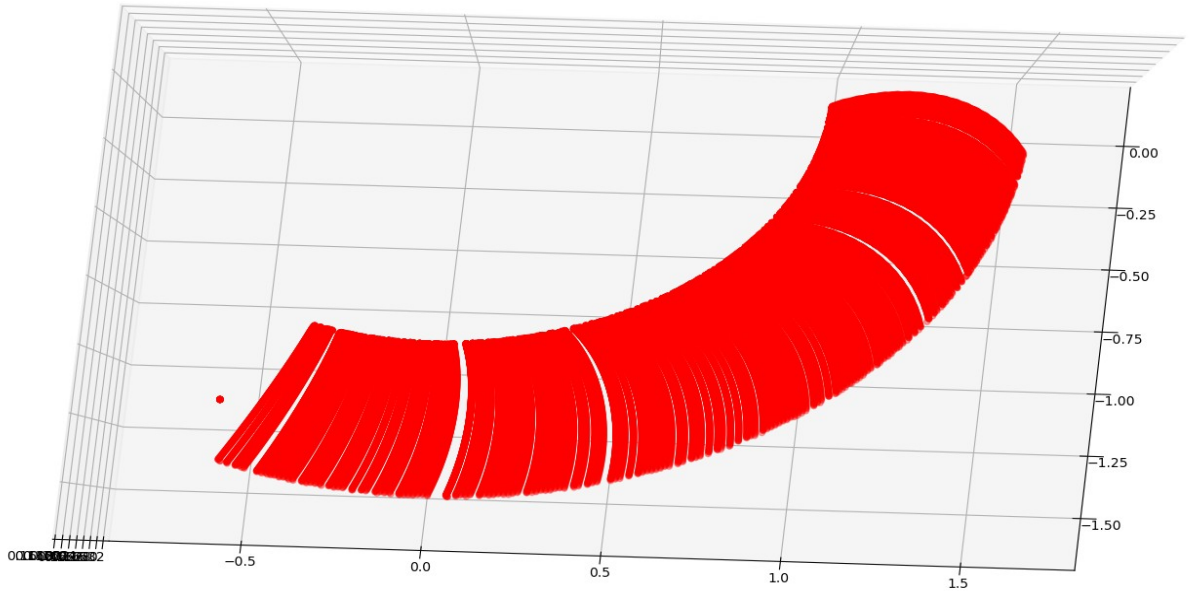


Figure 10 mainly just shows us very minuscule elevation differences – this is more-so just noise data. The model shown above is meant to only move along a 2d plane and not a 3d one, so the fact that these differences are so minute is a good thing – they can be attributed to jitters in the simulation or a slightly misaligned axis.

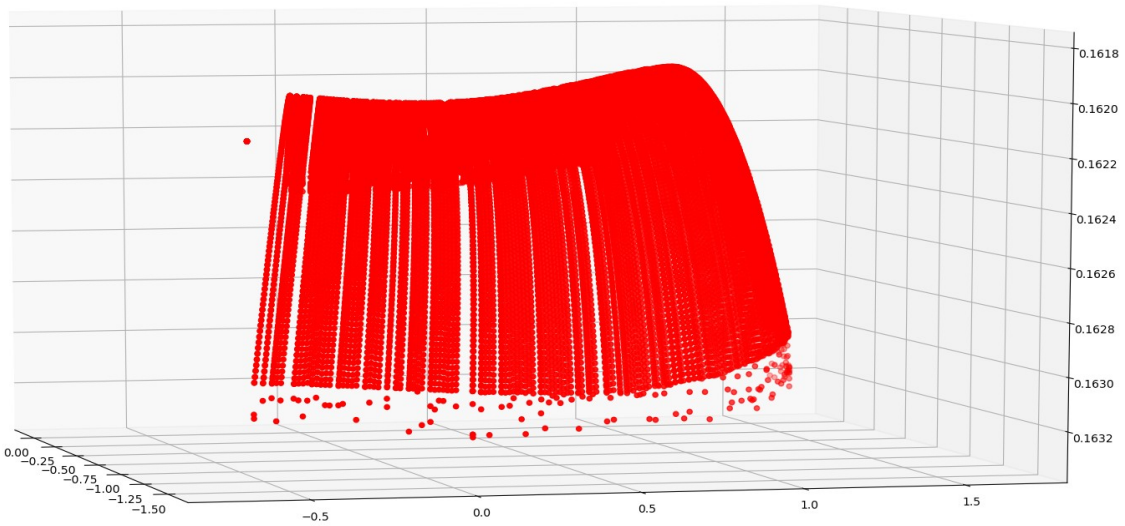


Figure 14: Data Visualization from the side

Lastly, in Figure 11, we can see a very condensed (for visual purposes) visualization of the neural network. The network of neurons is organized based on the XYZ and motor values of each neuron's data, so a 3d point with given motor values would be a neuron next to another neuron with a very similar XYZ and set of motor values. The end goal is to spatially organize all of our robot state data in such a way that this can be used for basic path planning. As can be seen below, due to the nature of a SOM, if we know the starting position of a robot arm and its desired destination coordinate, we can use the neurons in this map to quite literally go from neuron to neuron, drawing a line. The whole point of this is so that the robot arm can maneuver itself gradually and efficiently.

Again, this is a significantly condensed example. In this case, the actual amount of states the robot records is roughly 40 000. Generally there would be a larger neuron count so as to avoid large gaps in between the movements. However, the core concept remains the same – organize all the data into neurons, splay out the neurons in a 2d map based on proximity and then use those neurons to calculate paths. XYZ(1,1,1) becomes XYZ(4,3,1) with just a few jumps via basic proximity path calculation.

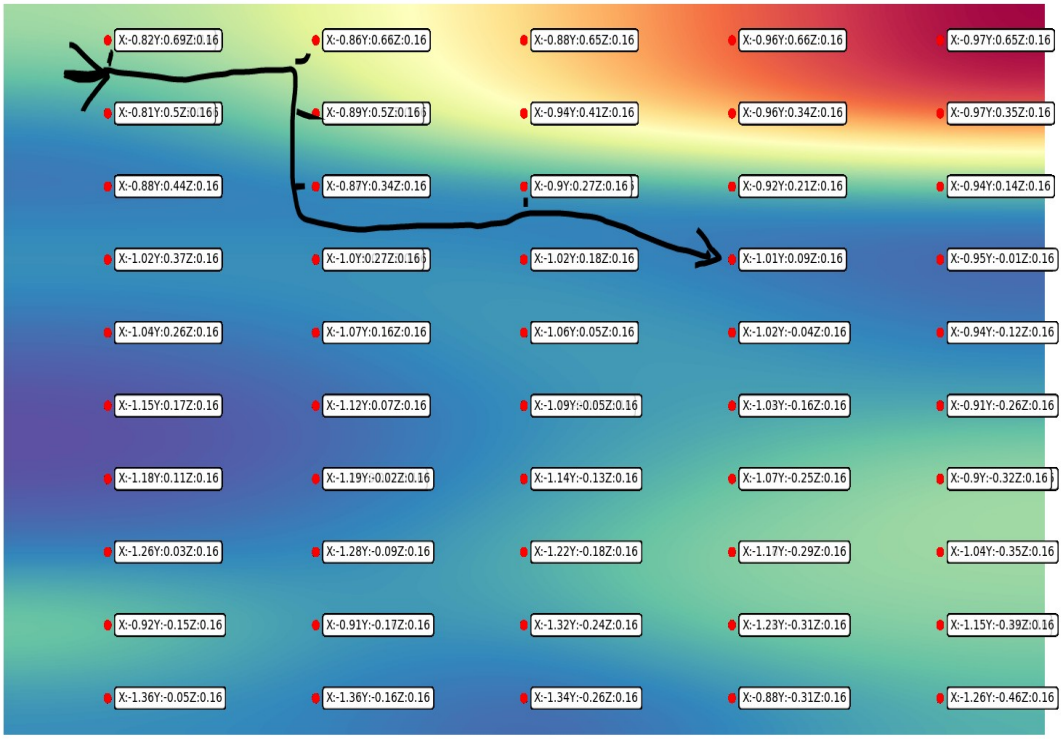


Figure 15: Path-finding example

It is worth mentioning that there were 2 other considerations for SOM packages, however they both had their issues and limitations. It was only upon the discovery and usage of Somoclu that good results were gained. Below is a small table to exemplify the findings:

*Table 2: Criteria table for selecting a SOM package*

| Packages | Retains training data after training? | Easy to use visualization tools/documentation? |
|---|---|---|
| MiniSOM | No | No |
| SimpleSOM | No | Yes |
| Somoclu | Yes | Yes |

What is meant by “retains training data” is that after training, MiniSOM and SimpleSOM don’t allow for looking up what states from the data set belong to what neurons(at least not quickly). This then makes a lot of problems and necessary workarounds, which are avoided via Somoclu.

### *5.1.5 Live Testing*

Outside of all the Simulation work, there was some experimentation done with a practical, real-life built robot arm(the same as the one shown in the first image). It was intended to be the final test for the motor-babbler results, however - ROS2 Foxy once again fails to support rudimentary features. For this testing to happen, the ROS2 Foxy environment needs to be able to communicate with an Arduino(or some similar type) micro-controller board. The micro-control board is simply an interface for sending electrical signals to the motors of the robot. However, ROS2 Foxy isn't supported in the package ros2arduino yet, and currently available workarounds are fairly unstable, so this live testing hasn't been a success.

A choice was made – due to time restrictions and the desire for the most important proof of concept, some simulation data would be manually collected, adapted and organized for a very rudimentary path-planning. This would then be put directly onto the micro-controller of the robot-arm just to verify that path planning, even extremely simple path planning, when done with the simulation data, would work in practice. Fortunately, the results were successful and the robot managed to follow a very simple, but still consistent circular movement pattern. What this meant was that, the motor-babbling system had been proven as a directly applicable concept.

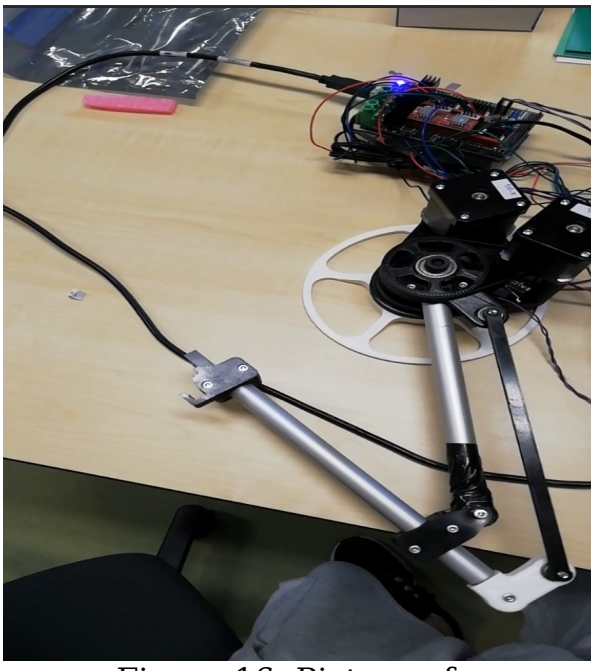

Figure 16: Picture of the robot arm during proof-of-concept testing.

## 6. Conclusion and Results

To give a brief summary of this project as a whole – the goal was to make a system that can train a specific type of modular robot arm. The system had to be dynamic and capable of automating as much of the training process as possible. After a lot of research and analysis, a design was planned out that separated the “brain” of a robot into 2 parts – 1 part that handles basic movement and one that actually learns/performs skills using the mapped out basic movement.

During Phase II (Design) some decisions needed to be made between how the entire system would work. It was concluded that , instead of mathematically calculating most of the simulation, using pre-existing simulation tools would be much faster and more practical, if not ultimately most efficient. Despite issues, the ROS2 environment was developed according to planning, a simulation environment in Gazebo was setup according to planning and both were developed to the point where they produce the desired simulation data for training a real robot.

While incomplete, the produced data from the simulations was used to create proof of concept SOM maps, as well as used to maneuver the real-life robotic model in a very minimal fashion. This was enough to solidly prove this project as a concept, and leads us to comparing what was ultimately achieved in this project.

### 6.1 Achieved Goals and Deliverables

To go over the goals of this project again, here is what was successfully completed:

- A system that can train any type of robot-arm that fits within the defined structure standards of this project. - DONE
- A system that can train robots in a simulated environment to save on time, using ROS2 as the main training environment. - DONE
- A proof of concept that such a system can work. - DONE
- A robot-arm that has been trained to move and perform a task with the training system. - NOT DONE
- A robot arm used for cutting/electrocuting weeds that has been trained with the developed system. - NOT DONE
- Full documentation of code. - DONE

- Final report of the project. - DONE
- Full project documentation that allows for easy replication of final results. - DONE

Unfortunately due to practical constraints, the student was not capable of finishing the entire project. However, enough has been finished to prove that this system has considerable value to offer.

This system is intended to be used for much more practical application with the briefly mentioned weed-cutter robot, however this goes beyond the practical capacity of this graduation project, and will be handled later on by GTL employees after handing off.

### 6.2 Recommendations

As closing recommendations for GTL, the student proposes the following:

- It would be wise to stick with the currently selected version of ROS2 (Foxy) for the time to come. While it is still new and therefore suffers problems with under-development, these problems are actively being solved and often have solid workarounds for the time being. While there can be debate for whether it would be worth going back to an older release of ROS2, it is certainly recommended not to go as far back as switching to ROS1.
- It is advised to look into the model importing capabilities of Gazebo and to test out how this would combine with the current simulation system. If GTL can confirm an approach to exporting their, for example, Solidworks models into Gazebo for quick simulation purposes, this would be a considerable benefit.
- Should this system ever be fully finished and deployed in a working capacity, it’s advised to use as strong of a computer as available. Gazebo’s ability to speed up simulation time seems almost entirely limited to the current PC’s hardware, so this can be a significant time saver.
- GTL needs to look into bridging the final gap that plagues this project – connecting a ROS2 environment to a micro-control board. There is proof that this has and can be done, however it is (as of the time of writing this) not as simple and straightforward as ROS1. This issue will only become simpler to solve as development of ROS2 Foxy continues, but the sooner a dedicated solution is applied, the sooner this project can be fully realized and applied to a working environment.

- Should GTL ever desire more complex second-phase learning of this learning system, it is entirely plausible to replace the Reinforcement Learning phase with something like Deep Q Learning. This will all boil down to what the company wants from this system in the future.

- It is very important to be aware of the fact that this simulation-based system will fail if it is run on weak hardware. It is very, very important to make sure that adequate hardware is being used for best and most stable results.

The most important conclusion that can be gained here is that ROS2 Foxy is indeed a very robust, time-saving environment. While unfortunately very fresh and somewhat under-developed still, GTL’s investment into learning it will only pay off with more and more value as time goes on. Should they finish this system and formally apply it to their work, they can expect a leap in finally not being reliant on “black-box” technology of other proprietary robot-arm manufacturers.

# Appendix

## Bibliography

1. Closed-loop acquisition of behaviour on the Sphero robot - Oswald Berthold∗ and Verena V. Hafner∗

∗Adaptive Systems Group, Dept. of Computer Science, Humboldt-Universität zu Berlin{bertolos|hafner}@informatik.hu-berlin.de

2. https://blogs.nvidia.com/blog/2018/08/02/supervised-unsupervised-learning/

3. Learning of Motor Control from Motor Babbling* - Tatsuya Aoki* Tomoaki Nakamura* Takayuki Nagai* The university of Electro-Communications, 1-5-1 Chofugaoka, Chofu-Shi, Tokyo 182-8585 Japan (e-mail: aoki@apple.ee.uec.ac.jp)

4. https://www.youtube.com/watch?v=qCn8lkacJz0

5. https://openai.com/blog/evolution-strategies/ - and all of its mentioned sources

6. https://www.youtube.com/watch?v=C4MUTIc-NB8

7. Self-Organizing Maps - Teuvo Kohonen

8. https://www.youtube.com/watch?v=JgvyzIkgxF0 - An Introduction to Reinforcement Learning

9. https://www.delta.tudelft.nl/article/robot-leos-first-steps

10. https://en.wikipedia.org/wiki/Evolution_strategy

11. https://en.wikipedia.org/wiki/Evolutionary_algorithm

12. http://www.scholarpedia.org/article/Reinforcement_learning

13. http://www.scholarpedia.org/article/Evolution_strategies

14. https://ai.googleblog.com/2020/04/exploring-evolutionary-meta-learning-in.html

15. https://en.wikipedia.org/wiki/Deepreinforcementlearning

16. http://www.scholarpedia.org/article/Deep_Learning

17. https://en.wikipedia.org/wiki/Motor_cortex

18. https://en.wikipedia.org/wiki/Cerebellum

19. https://en.wikipedia.org/wiki/Motor_babbling

20. A Motor Control Model Based on Self-Organizing Feature Maps – Yinong Chen - https://drum.lib.umd.edu/handle/1903/908

21. Bio-inspired Intelligent System Design – Jan Jacobs

## Phase I Artefacts

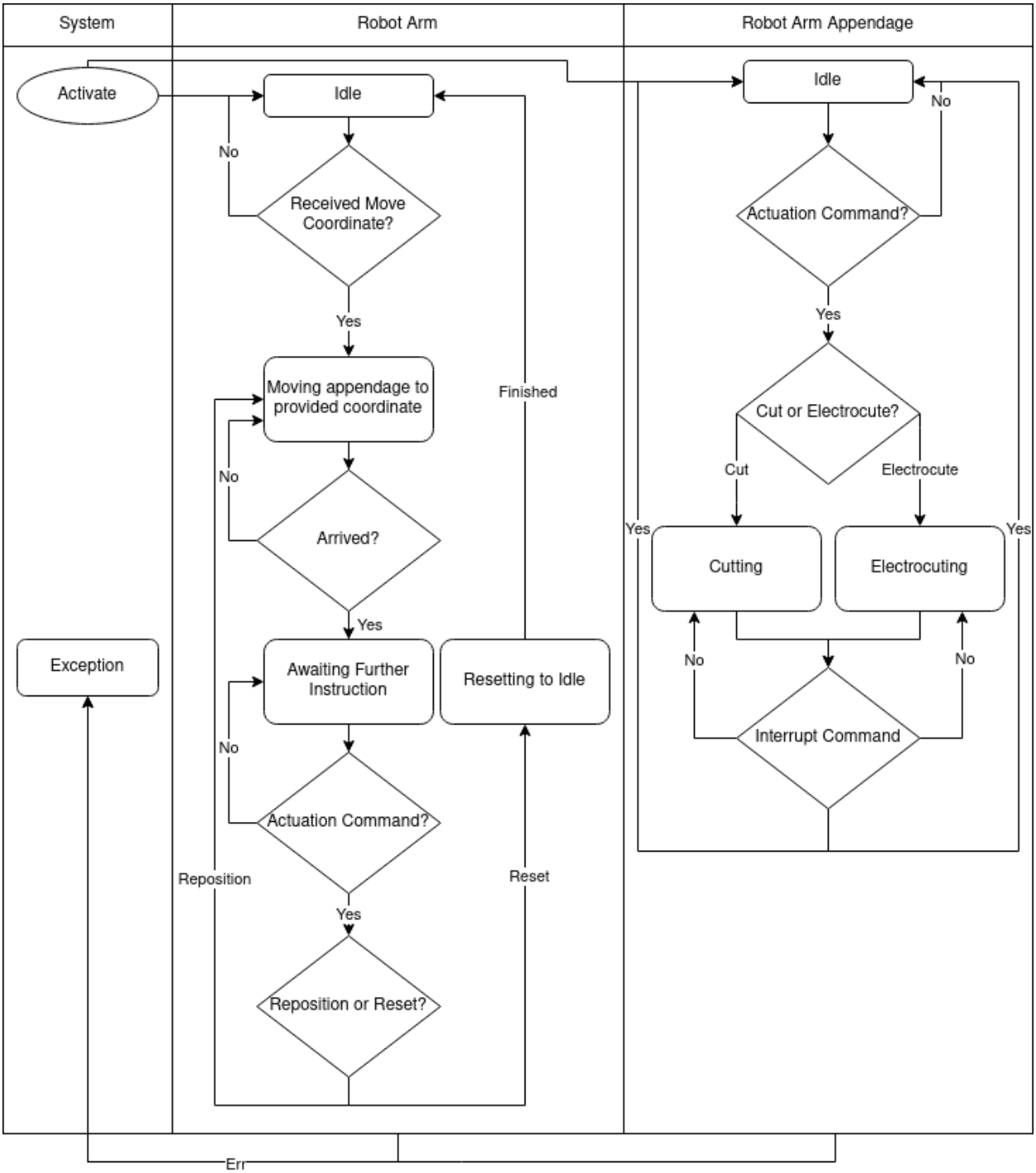

Figure 17: Cutter Robot State DIagram

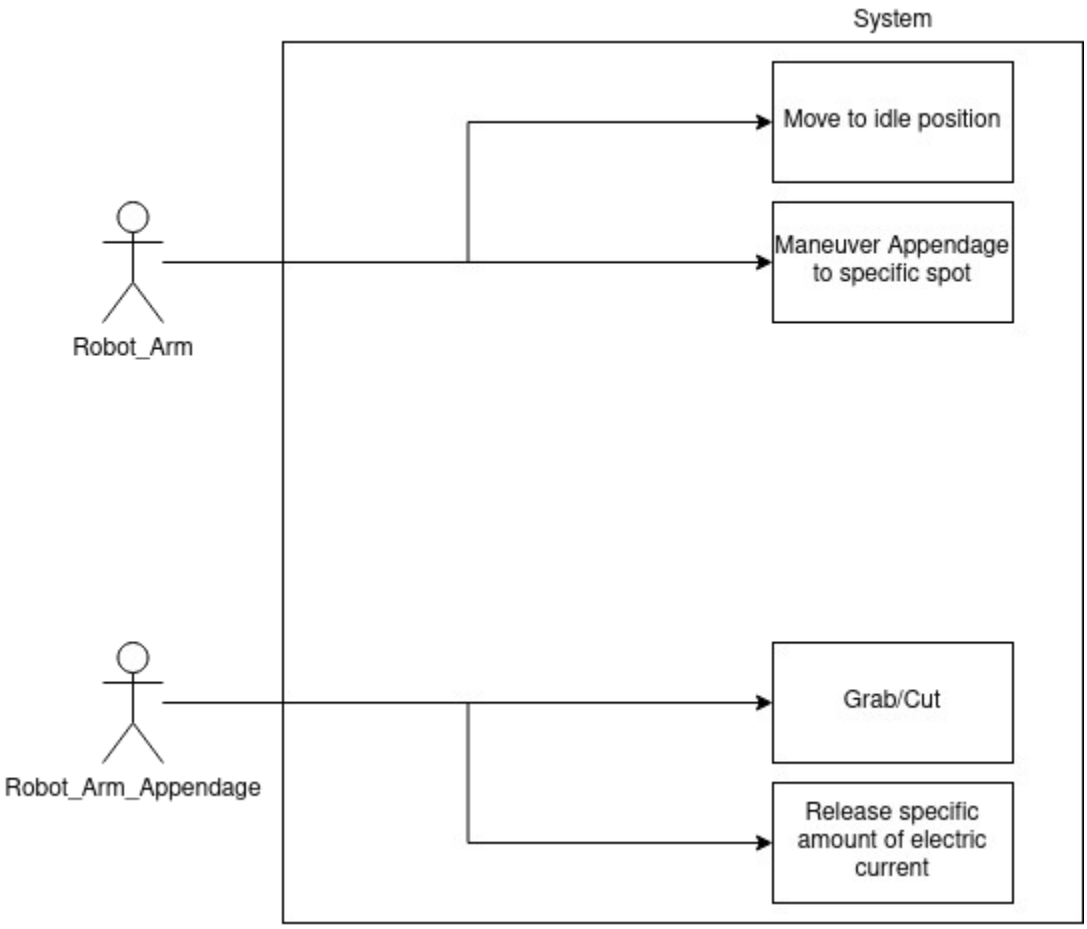

Figure 18: Cutter Robot Use Case Diagram

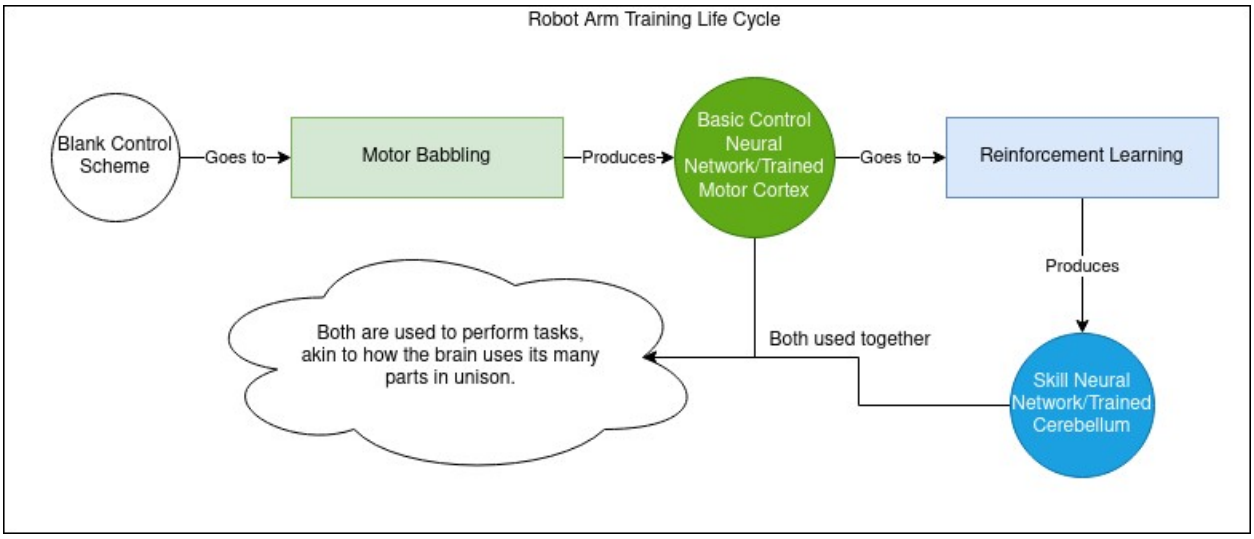


Figure 19: Proposed Robot Arm Learning Cycle from Researhc Document

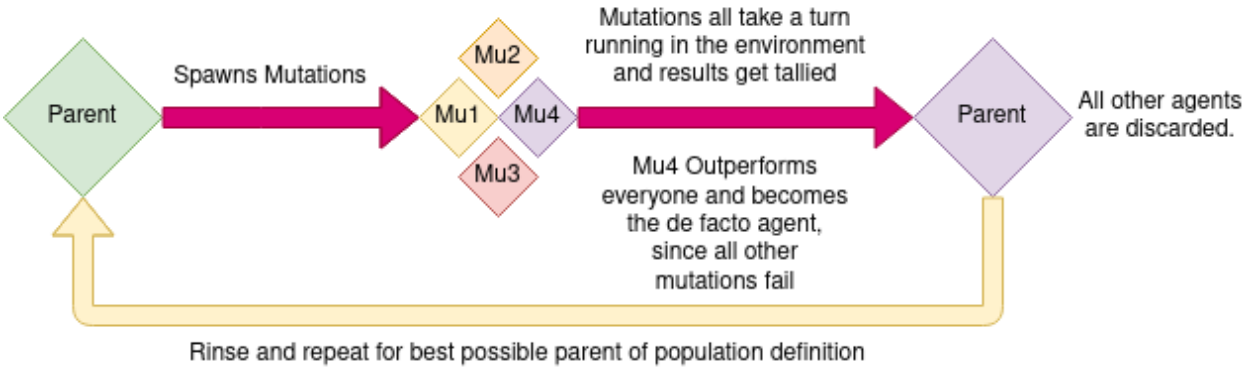


Figure 20: Visualization of evolutionary algorithm.

## Phase II Artefacts

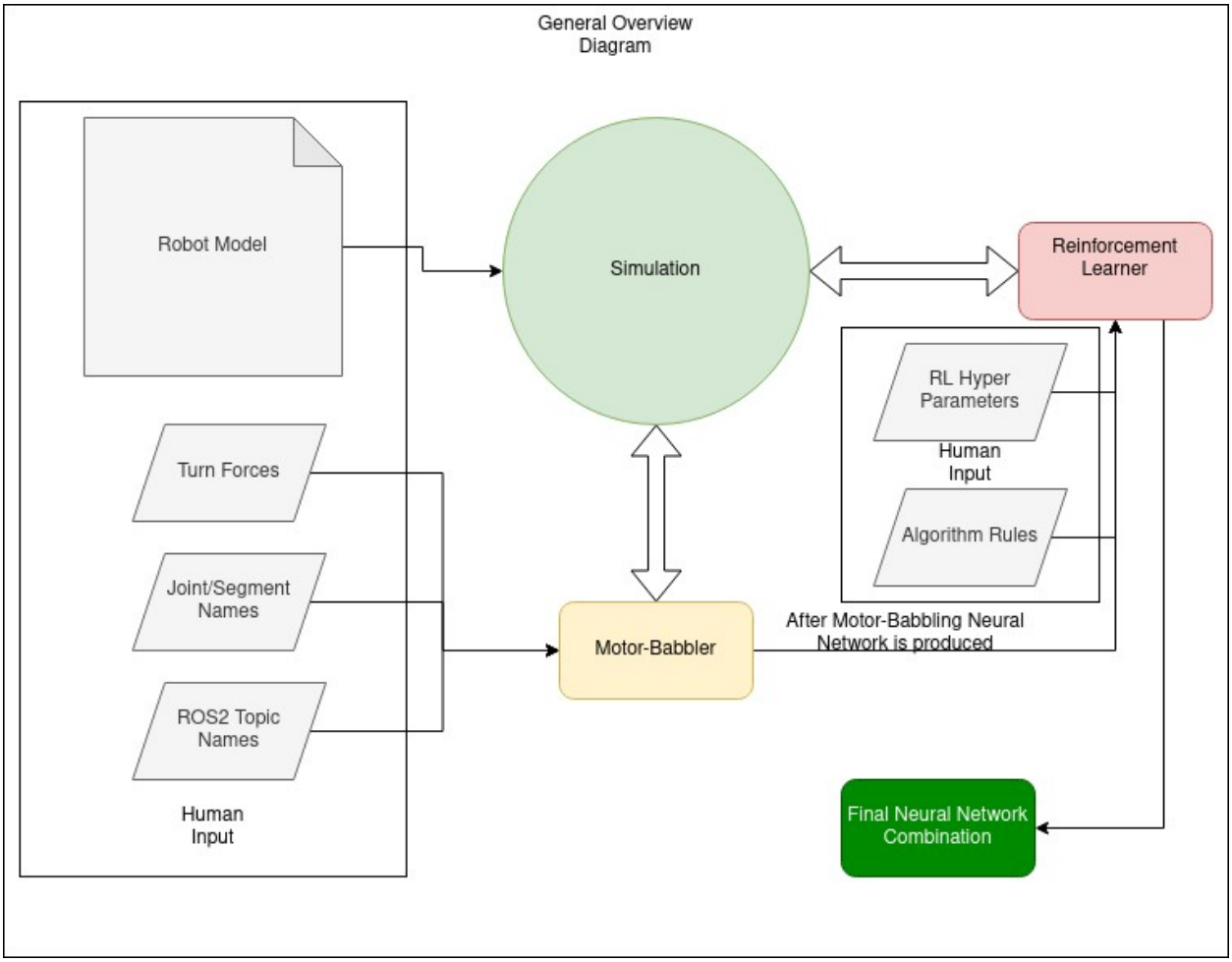


Figure 21: General Overview Diagram

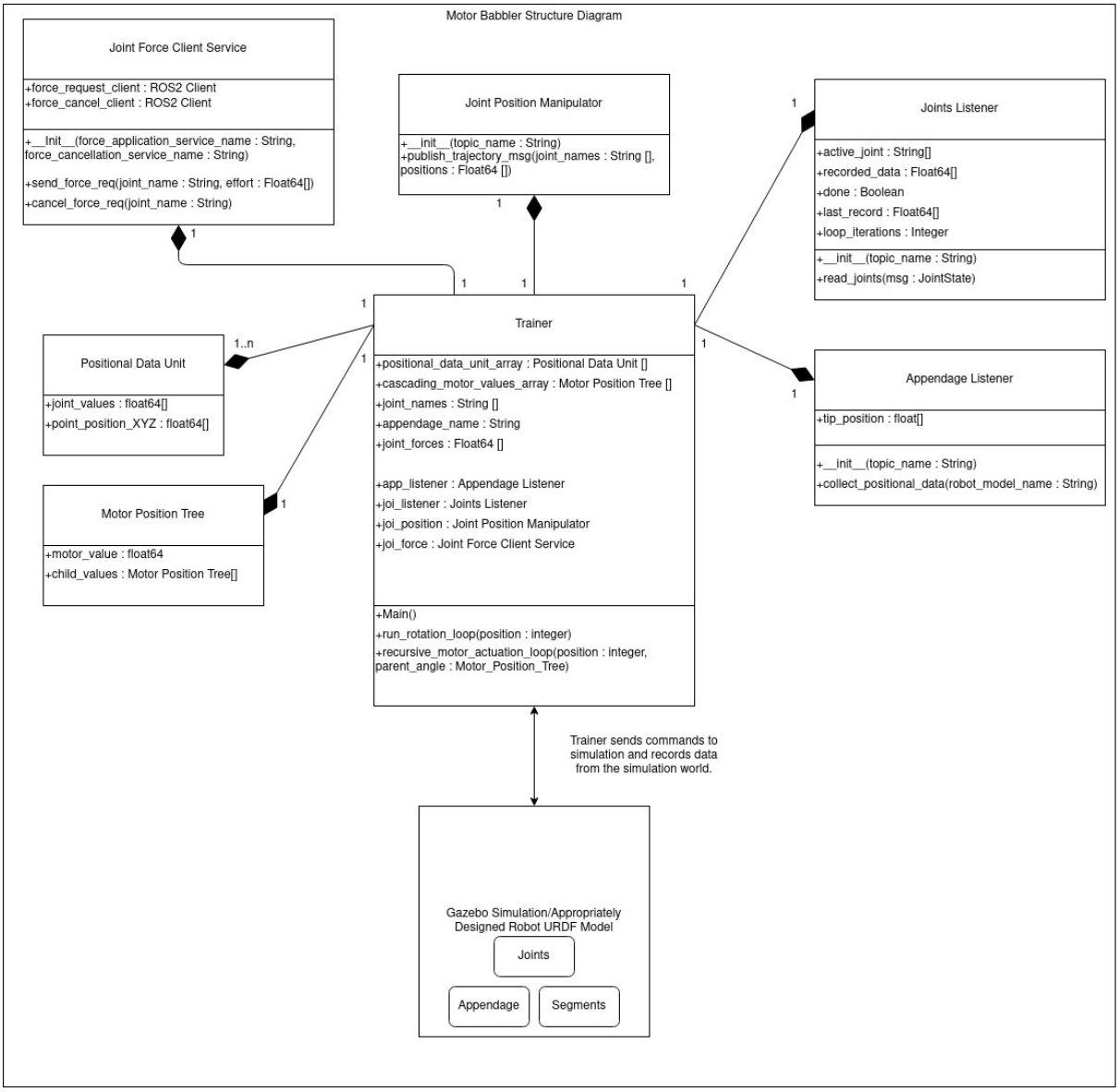

Figure 22: Motor Babbler Structure Diagram

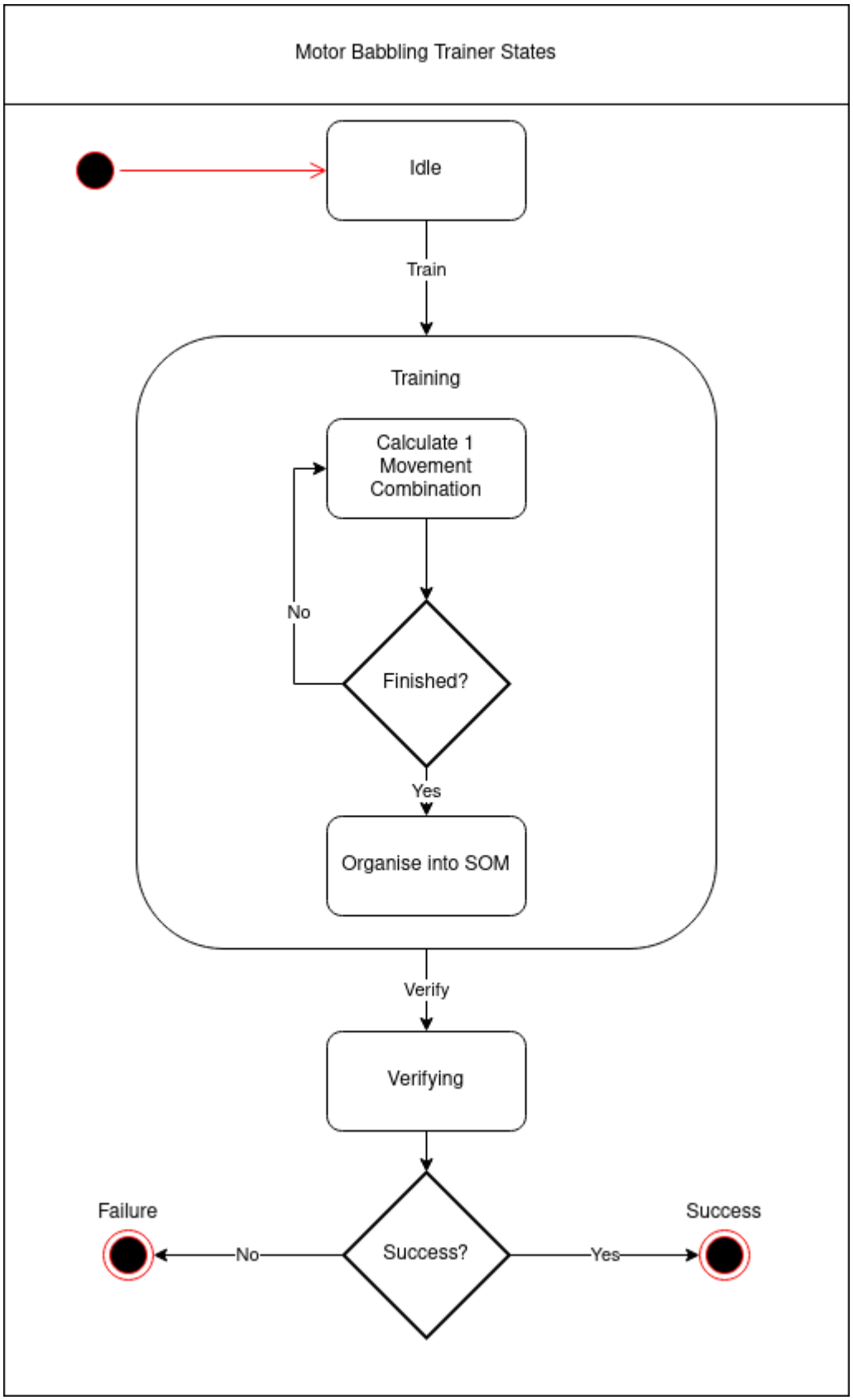


Figure 23: Motor Babbler State Diagram

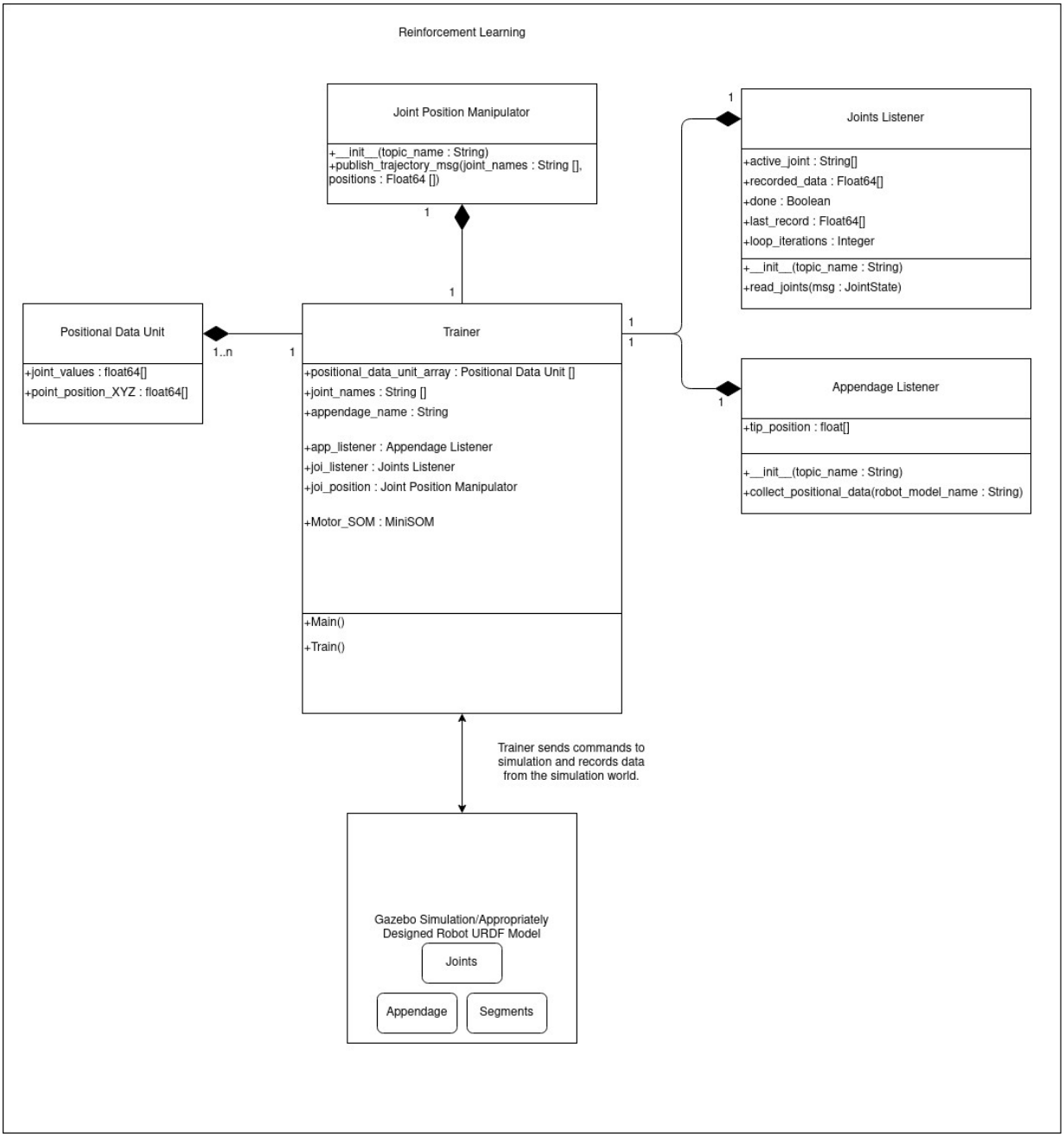

Figure 24: Reinforcement Learning Structure Diagram

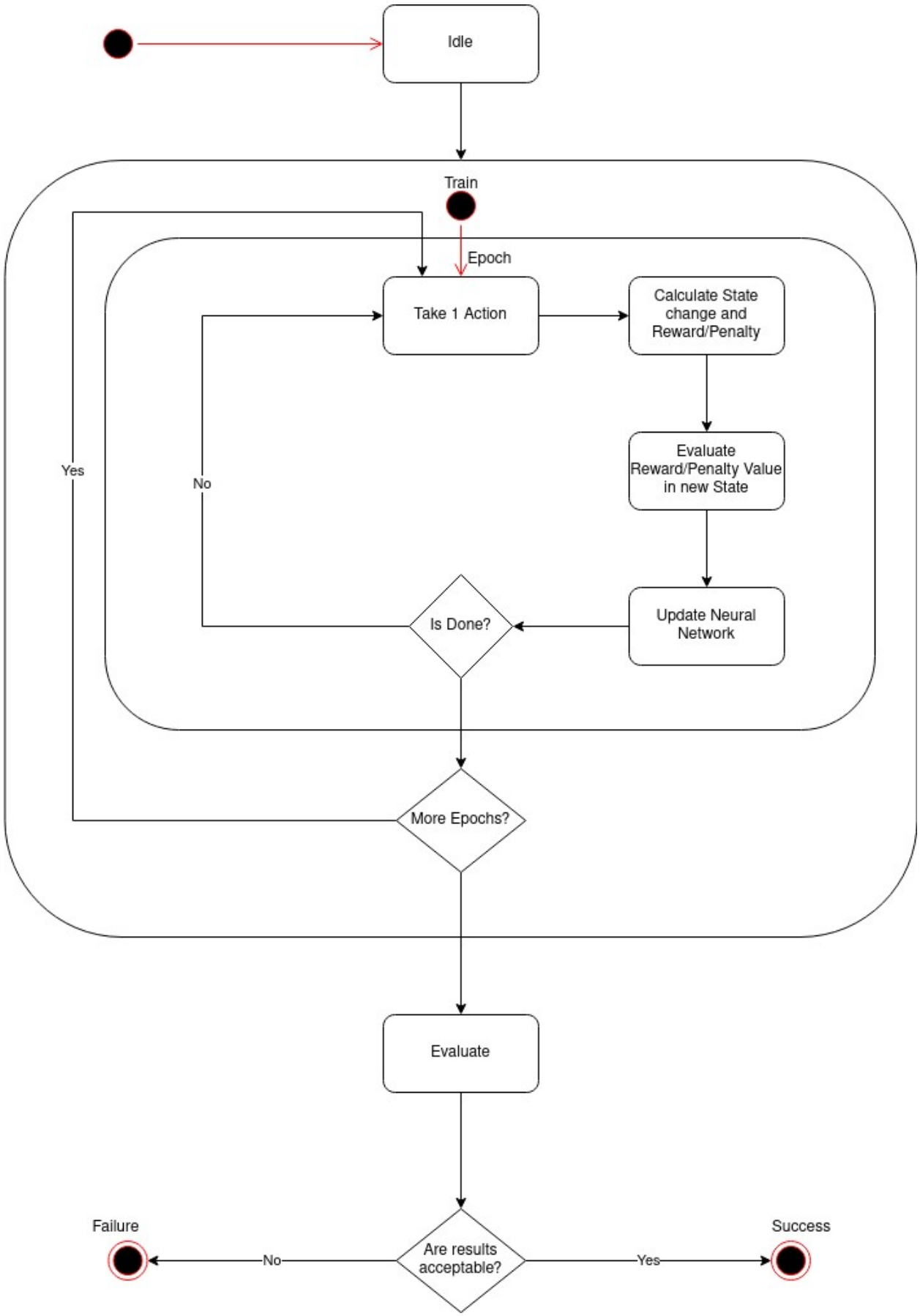


Figure 25: Reinforcement Learning State Diagram

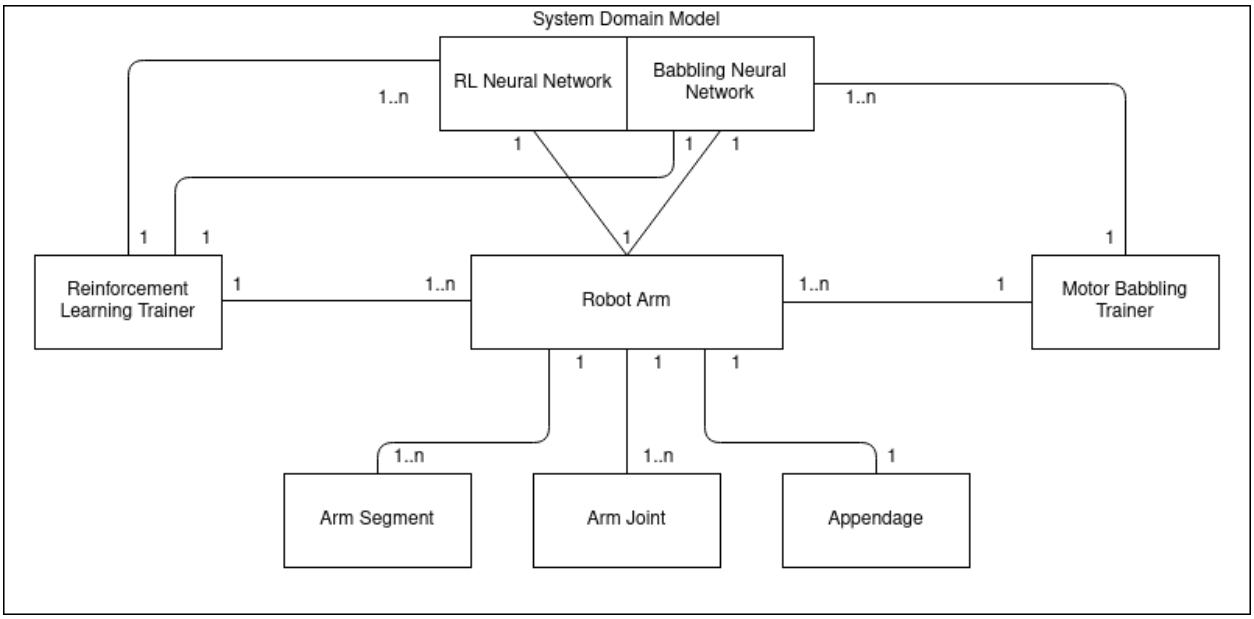

Figure 26: System Domain Model

## Final Supplementary Documents

Beyond here can be found all supplementary documents for this project.

## Table of Contents

# Research Document - ROS-based machine-learning in ROS2

by Valentine Ezekiev, 29714291.

## 1.Introduction

This document will serve several functions. It is for the graduation project "ROS-based self-teaching robot arm". The student needs to better understand the concepts behind machine learning, specifically the semi and unsupervised, motor-babbling variety of machine learning (in a ROS2 environment). Motor-babbling, specifically, is brought up because it's been recommended as a soft requirement/strong suggestion for this project. In this research document, research questions and context are outlined, then conclusions from various sources (research documents, YouTube videos, general research) are written down, in addition to any objections to said documents(if any) from the student. Once enough information has been gathered, this document will end with a conclusion/summary chapter. That chapter will act as a summation of all the important-to-the-project information that's been obtained.

## 2. Project Goal (Further Context)

The purpose of this project is to create an environment in which a robot arm can learn to perform given commands. For instance, take a robot arm with 2 degrees of freedom (2 connective segments with 2 rotating motor joints) and a grabber at the end. In the machine-learning environment, this robot arm should be able to first learn every movement that it could make, given its physical capacities - meaning, just to learn all the ways it can move in relation to itself without collision with itself. After the arm has learned what it can do in terms of the capabilities of its separate motor components, it then has to be able to put those fields of movement into skills. What this means is, if someone gives a learning objective to said arm to figure out how to reach a distinct point in 3-d space and pinch, the robot arm has to use the basic movement knowledge that it has to figure out how to do this.

This basic example scenario should be applicable to a variety of robot arms that generally share a modular structure to the one described above - 1 or more degrees of freedom and an appendage for a tip (for additional information on what exactly is considered an appendage in this project, please reference the Software Requirements Specification). The idea is to create some kind of system that can allow all of these basic robot arms to

figure themselves out and then figure out how to create skills out of their basic movement abilities, all while doing this in an unsupervised or semi-supervised manner. The robots will have to have their learning objectives set. Conceptually, this should serve as the guidelines within which the robots will attempt to discover information on how to perform.

The need for this complex learning system is due to the strive of GTL to create custom robot arms in the long run. They want to have a robust and mostly automated system to cut down on the man hours necessary to manually program robot arms.

## 3. Main research question and sub-questions

The primary research question of this project is "How to create an environment that can teach a variety of similar robot arms* to teach themselves how to actuate and perform tasks ,with set parameters, from scratch?". This is a complex question, so there are a lot of blanks in between that need to be filled in. Namely:

1. What is motor babbling? How far can it go, in terms of teaching a robot arm to perform a given complex task(id est picking something up)?
2. What other concrete varieties of semi/unsupervised machine-learning can be reasonably (meaning - avoiding technologies and concepts that are currently too unstable and under-developed for practical use. To further clarify, technologies/concepts that lack a proof of concept and historically practical and successful application in a ROS robotics context, generally would not be considered.) researched and utilized in this project?
3. What variety and/or combinations of semi/unsupervised machine-learning would best suit this project's goal?
    - 3.1 Can this method be applied to a variety of robot arms?
    - 3.2 Does this method allow for motor babbling?
    - 3.3 Can this method allow for one robot to learn and copy from another robot's neural network if they share similar features in design?
    - 3.4 How long would such a method take a singular arm in a virtual environment? How would time consumption scale with degrees of freedom?

3.5 How would the chosen method of machine learning be applied to the ROS2 environment? Does it even need to interact with ROS2?

3.6 What parameters would be needed to set learning goals? What would be the most modular approach?

3.7 Are there preferable alternatives?

## 4. Terms

Important Note - when referencing a "robot" or "robot arm", it is purposefully left vague. The concept of this project is that a wide variety of robot arms should be able to learn how to function. At best, a minimal standard definition that can be provided is 1 degree of freedom, affixed to a base of some sort and affixed with an appendage at the end of the robot arm with a connective segment in between.

ROS2 - Robot Operating System 2. An environment for programming robots.

SOM - Self Organizing Map. A collection of data points with comparable data values that can be laid out onto a flat 2 or 3-dimensional field and represent the pieces of data in a logical spacial organization.

Motor-babbling - a form of machine learning that mimics baby-babbling. Essentially, the robot records every possible joint combination that it can actuate and then saves and organizes that data into some form of neural network.

## 5. Research Conclusions

### Answering Question 1 - Defining motor babbling and it's limitations.

*Relevant references – [3] [9] [19]

Motor-babbling, as a concept, is derived from a process called baby-babbling. During their infancy, human babies can often be seen performing various combinations of random movements. This is described as them developing their own sensor-motor relationships - which can more simply be described as babies figuring out how their bodies move and how it feels to perform the various movements, as well as determining the limits of their movements. Motor-babbling is the same thing[19], however, applied in the field of intelligent robotics, where a robot would map out all of its movements into some form of neural network for later use. In more practical terms, a system could calculate every conceivable combination of its own joints(depending on the degrees of freedom) and map it out into an organized neural network[3]. This neural network can then be used by the machine's controllers later on as a map on how to perform smooth, sweeping movements and adjustments. The reason this would be necessary is due to a very practical problem that robot arms face - to move from point A to point B, an arm needs to be able to know how to link those 2 points by performing all the movements in between. If the motor system of a machine doesn't know how to gradually position to a different point of itself, it would simply break.

While motor-babbling is an excellent tool for mapping out the motor functions of machines[9], it alone is not a method that allows for the development of complex skills and patterns, by very definition. As a soft requirement of this project, it will need to be used, but it will need to be used in conjunction with another machine-learning method that can use the information provided by motor babbling to develop skills.

### Answering Question 2 - Listing plausible forms of semi and unsupervised machine learning to add to our motor babbling.

Since motor-babbling only solves a part of the robot-learning problem for this project, an additional learning method will be necessary to further develop "skills". In the context of this document, skills refer to complex sequences of basic movements with the aim to achieve objectives that go beyond the scope of merely "actuate to point B".

### *Reinforcement Learning*

*Relevant references - [2], [8] and [12]

Reinforcement Learning (RL) is a very popular method for training a neural network in the field of robotics(though not only here, it is prevalent in many other disciplines due to its generality)[2][8]. It revolves around the concept of using a singular agent (id est the robot arm) who is given a goal in an environment. It is classified as a third separate paradigm of machine learning, apart from supervised and unsupervised learning, but it can simply be viewed as semi-supervised machine learning. After having been given a goal(set of hyper-parameters) from an external, generally human, input, the robot will proceed to experiment in its environment and attempt combinations of actions that give it varying reward and penalty scores. Should the robot perform actions, be they singular or sequence-based, that yield negative, penalty results, the neural network will note down to avoid repeating those movement combinations in the future(at least in the given environmental context at the time of recording). And vice versa - positive rewards are remembered by the neural network as something worth re-actuating and optimizing. The point of this process is to maximize the notion of cumulative reward.

Image below for illustrative purposes.

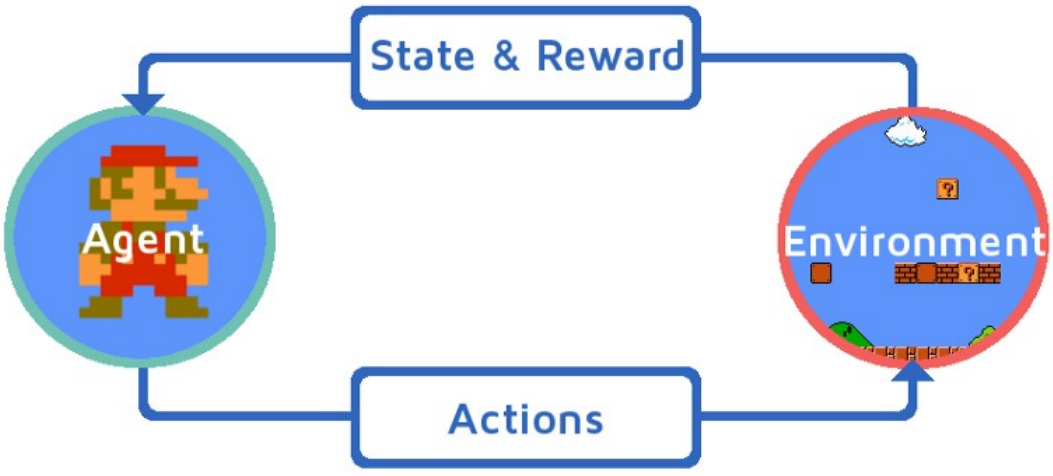


Figure 27: Basic Reinforcement Learning Loop

To give a brief, although in-depth, description of the working process of RL, a brief description of MDP (Markov Decision Process)[12] will be necessary.

MDP is a mathematical process used to deal with decisions made in known environments, the outcome to which is unknown. During a process, there are multiple time steps. For each time step, there are environment and agent states. An agent in the environment can choose to perform an action onto the current state. After that, the time step moves forward by one and the functions responds with a corresponding reward value for the imparted action and a change in states. An addition to MDP is partially observable MDPs. Very similar, however in this case, the agent can not observe the state of the environment it is interacting with. In a scenario like this, the agent assumes that the environment's dynamics are determined by an underlying MDP that it cannot observe. It must adapt its strategy to account for the probable states of the environment based on observations and observation probabilities, as well as the input from an underlying MDP. More succinctly put - this is the model that is used in unobservable/distorted environments. It is used for calculating the best probable solutions in such environments.

As described above, the core concept of MDP translates into RL - the agent enacts actions upon the environment and observes the changes in state and the reward values that it receives. It then strives to optimize its action patterns in relation to its environment so that it can optimize those rewards. With reinforcement learning come adjustable hyper-parameters that can, for example, direct whether the agent should prioritize short-term reward maximization or long-term reward maximization, as well as the ability to restrict and adjust what acceptable values of penalty the agent would consider worth risking. The work flow of RL is a balancing act of exploration and exploitation. The model has to explore its environment in order to obtain patterns and understand how it can perform a given task. It then must exploit its knowledge to continue receiving guaranteed rewards. However, there are often cases in which an exploitable pattern can be explored in order to yield a temporarily lower reward, but in the long run, yield an ultimately more efficient pattern which yields a greater total reward. It comes down to the needs and wants of the given training scenario/project.

The explanation of the mathematical algorithms is being left out of this document, partly due to brevity and partly due to the fact that it is not integral to the purposes of this project and research. As can be summarized above, RL is an excellent and robust tool for training/optimizing neural networks. Given appropriate configuration, it can provide an excellent tool for training robot arms and even do so in relatively brief periods of time, given a virtual environment.

### *Evolution Strategies*

*Relevant references - [5], [6], [10], [11], [13]

Evolution Strategies (ES) is an optimization algorithm that is actually a sub-class of a greater, general nature-inspired group of optimization and direct search methods, called Evolutionary Algorithms[10][11][13]. The general theme of these algorithms is the mimicry of natural evolutionary occurrences, such as reproduction, mutation,recombination and selection. What this translates to in general terms, is having multiple agents, all attempting to perform solutions, and then optimizing the algorithm according to the best performing agents, while discarding the poorly performing ones. This is simply a generalization, as the whole of Evolution Algorithms has a very large selection of singular algorithms and entire family sub-groups of algorithms. Some specialize in not giving a singular, most-optimal solution, but rather multiple viable solutions. Others focus on performing the same core task of mimicking populations and evolution, but also mimicking data attributes that would help with tracking things like social developments populations, knowledge sharing, individual and distributed fitness and so on. Many of these algorithms are ,theoretically, very applicable in the context of this project. However, due to the fact that almost all of them fail to have concrete and accessible uses in a robotics environment, especially a ROS2-based one, this removes them from the consideration as optimization algorithms for teaching the robot arms in this project. While theoretically viable, it would require a great deal of research and experimentation - and this would fall out of the scope of the goals of this project, as well as the skill set of the software engineering student developing it. An honourable mention goes out Evolutionary Multimodal Optimization - a means of determining multiple most-optimal solutions to a given problem. This exists as a means of dealing with problems where the most efficient solution would not necessarily be affordable, given physical and resource limitations. In theory, it would have been worth considering, as it could be an appropriate tool for teaching a robot arm multiple approaches to obtaining an objective, should it be unable of using the most optimal approach for due to a given variable reason.

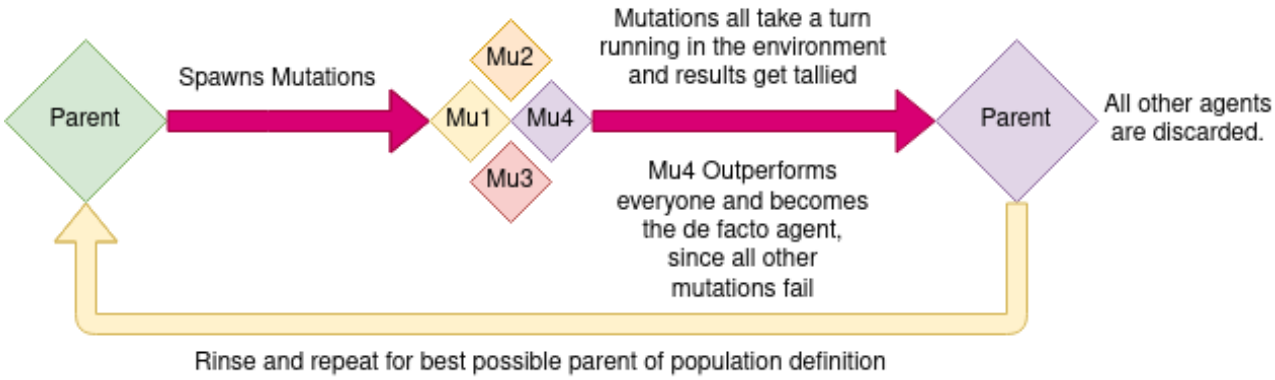

Figure 28: Basic Evolution Strategy Loop

Back to ES, they can be viewed as a somewhat opposite of RL - ES has multiple agents, anywhere up from a minimum of 2 - a parent and a mutation. Much like before, this works on a "reward score" system. If the mutation has a higher than or equal reward to the parent, it becomes the new parent and spawns a mutation of itself. This concept scales up to many agents being sent out towards a problem where they each attempt their own slightly different solution (mutation). Then the solution weight of all the "positive" scores gets summed up and the average solution for the problem, where initially it may start at random, has now adapted (in the even that this average mutated score is higher than or equal to their collective parent score). This approach has some excellent benefits to itself - in general, it results in a simpler implementation of the algorithm. Additionally to this, it allows for parallelization. Due to the fact that each mutation can be ran as its own thread in a virtual environment, this allows for the testing of hundreds of optimization calculations at a time. In source [5], an example was given of teaching a humanoid figure how to walk, in a virtual environment, in merely 10 minutes, due to the fact that nearly 1500 computer CPUs were utilized during that test.

Evolution strategies and the code shown across sources [5] and [6] make it a promising method for the secondary phase of training skills for a robot arm. The ability to run virtual, parallelized instances of training offer the ability for considerable scalability and the general simplicity of this algorithm marks it as applicable in many scenarios in this project and generally a competitive alternative to RL. However, the lack of concurrent and streamlined implementations of this as a package, lacking any presence in a ROS environment and lack of any solid usage in precision-based training all work against this method. It is also worth mentioning that, as mentioned in

[5], in singular machine instances, ES is known to be slower and less data-efficient than RL.

### *Deep Reinforcement Learning*

*Relevant references – [15], [16]

Deep learning comes from the same family of machine learning algorithms that RL comes from. This method of learning works off of the concept of having an indefinite "depth" of neural layers in a neural network, each having a fixed amount of neurons per layer. Generally used in image-recognition based-tasks, the idea is that each layer of the neural network can break down features detected on an input further and further, until an optimal minimum unit of feature is achieved. While a good and popular tool for training neural networks for image and language processing, in terms of robotics - it isn't designed to train a neural network to actuate, so it wouldn't find any use in this project. However, Deep Q-Learning (or Deep Reinforcement Learning) offer a combination of training both image and actuation neural networks. This setup often finds application in training robot arms where having a camera to guide the work of the robot would be a necessity. To summarize it a bit more succinctly, regular RL would teach a robot to perform optimal maneuvers in a smaller work area with fewer variables or - where the environment in which it operates can be considered more static and simple. A Deep Reinforcement Learning setup would allow a robot to, using complex external sensors and/or cameras, navigate and learn to determine optimal maneuvers in a much more complex and varied work space.

While Deep Reinforcement Learning offers more robustness and ability to handle complexity, it is, functionally speaking, RL with image processing attached. For the purposes of this project and its requirements, complex image and environment processing are not part of it, which would render this method of robot training an unnecessary addition in complexity.

### *Comparison Table - for visualization.*

The table below is meant to give an overview of the primary pros and cons of the considered Second-Phase machine learning algorithms. Second-Phase means that these are the algorithms that come after the motor-babbling.

Comparison Table for Second-Phase Training Method

| Criteria for Comparison | Reinforcement Learning | Evolutionary Strategies | Deep Reinforcement Learning |
|---|---|---|---|
| Already Applied in ROS? | Yes | No | Yes |
| Used in robotics Previously? | Yes | Not Extensively | Yes |
| Does not exceed requirements? | Yes | Yes | No |
| Can be adapted to work with variable arms? | Yes | Yes | Yes |
| Requires Little Prep Work? | Yes | Yes | No |

Figure 29: Summary Comparison Table

## Answering Question 3 - Combining the information into a viable proposed system for training variable robot arms.

*Relevant references for the human brain analogy – [17], [18]

Having collected and summarized the information from questions 1 and 2, the student has the necessary tools to train a robot arm to teach itself how to perform a given task via sequence of actuation. Bearing in mind the applicable and non-applicable applicants, the follow up training method that has been selected (for after Motor-Babbling) has been Reinforcement Learning, due to it's practical and well supported nature, being more usable and established in ROS2 than Evolution Strategies and less unnecessarily complex than its Deep RL counterpart.[17][18]

The student has concluded on using biomimicry as inspiration for the design of the training system, as biomimicry is often an excellent source of inspiration for systems design. In humans, the following has been observed - at a young age, human babies are known to motor-babble, in a sense, training their motor cortex to become familiar with all of their joints and

muscles and their respective abilities to turn, flex and contract. The motor cortex is the part of the brain responsible for, generally speaking, basic, more crude motor control and motor command execution. Over the course of their lives, humans train another part of their brain, the cerebellum, which is generally deemed as responsible for more finite, balanced and precise movements, such as walking or learning to swing a bat to hit a ball. The cerebellum serves merely as a secondary motor control unit, and shares no ability in sending signals for contraction directly to muscles in the body.

The image below denotes the the overall structure proposed to answer the primary question of this research document. Utilizing what has been learned, and copying from nature while optimising some things, the system below fulfills all requirements of this project and indeed allows for a robust general system that can train a robot from scratch. It is important to note that the suggested structure uses multiple neural networks. This is meant to mimic how the human brain is not one entire whole neural network, but rather, many logically separated neural networks that work together in a coordinated fashion.

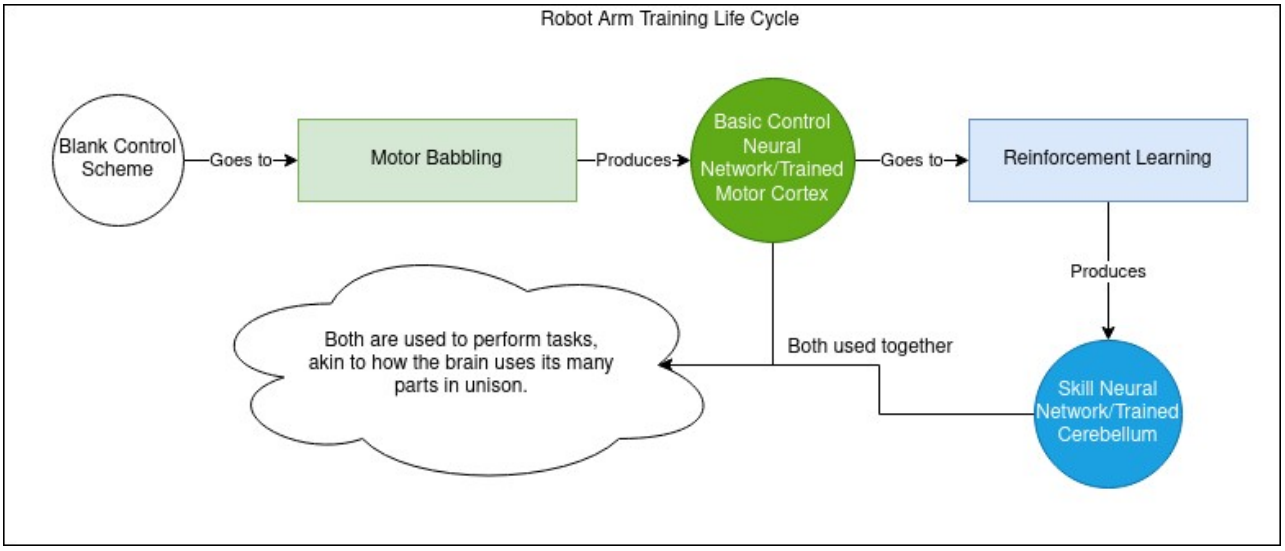

Figure 30: Final proposed system structure and work cycle.

This structure seems to very closely follow and offer a solution to the problems faced by this project. With motor babbling, the system would be able to discover all of it's possible movements - essentially to map itself out. In terms of the project, here is where some external human input will be

expected, to set the expected twist and bend limitations of each joint, before training can begin. Each possible combination of motor joints can be recorded as a piece of data, with the separate motor values being it's variables. This information can then be saved and fed into a self-organizing map (check reference [3]) neural network. Self-organizing Maps (SOMs), as shown in reference [7], specifically chapter 3.4 of the book, can serve as an excellent "motor cortex" of a robot. Once the map is organized, the machine would have a step-by-step mapping of all the motor combinations it would need to perform to reach any and all of its physical actuation points smoothly and efficiently. An added bonus of this setup is that the "motor cortex" of 1 robot arm can be ported over to a robot with identical degrees of freedom and at least similar angles of joint bending. So long as the number of degrees of freedom remains the same, as well as the length and shape of the connective inter-joint segments remains identical, the SOM can be scaled up to account for wider or thinner joint bend angles.

Past that point, using reinforcement learning, the machine can be taught to develop a second neural network (similar to the cerebellum) which would train it on how to perform complex sequences and combinations of the basic "maneuver map" that it learned during the motor-babbling phase. With adequate programming and determining of the hyperparameters, very little external input would be necessary for a robot arm to establish its cerebellum control scheme. While RL will not be as fast as the parallel capacity that, theoretically, Evolution Strategies would allow, it is proposed that it should not take longer than 10-20 hours in a virtual environment for most general robot arm training scenarios. This expectation is only based on other available videos and simple robot arm projects, so it is to be taken with a grain of salt.

In terms of motor-babbling, the ROS2 environment is not concerned, since the motor map can be calculated algorithmically, without the need of a virtual environment. As far as RL goes - ROS2 supports ros2learn and gym-gazebo2 for simulating and training a model, packages dedicated to reinforcement learning in a virtual ROS2 environment. The overall structure of this system would then fulfill all requirements posed by the project and research questions. It will inherently become a robust system, requiring only hyperparameter configuration to train a variety of physical robot arms (as long as they follow the defined standard of robot arm for this project). These hyperparameters will only concern the degrees of freedom, numbers and sizes of segments and bend angles of joints.

It will allow for at least partial neural network transference, due to the fact that if a robot arm shares identical degrees of freedom and limb segment length ratios with another robot arm, the mathematical changes in its joints

to reach various 3-dimensional spots will remain identical. While there may be viable alternatives to this system, given the fact that this is a very experimental project and that copying tried and tested structures from nature is an approved concept in the scientific community, the current structure leaves nothing additional to be desired, as of this being written. It is deemed to be sufficient for the requirements of the project.

## 6. Summary

There are many methodologies and approaches to machine learning. This makes for a field with no, necessarily, right or wrong answer to a question as robust as the primary question of this document. However, given the discoveries listed above and the requirements of this graduation assignment, putting together a feasible solution that fits becomes easy.

Using biomimicry, it's easy to copy tried and true learning methods from nature. The natural process of human brain development, as understood by this research, lends itself excellently to the exact issues faced by this project. Using biomimicry, just as human babies do, it is very practical and possible to map out all degrees of actuation in a robot arm and put them in an organized neural network thanks to motor-babbling. Then, continuing to mimic human brain structure, it makes sense to train a secondary neural network for motor-based skills (cerebellum). Using the popular robotics methodology of Reinforcement Learning, especially after having a full actuation map of a robot arm prepared, training a neural network to learn how to perform specific arm movements in a given environment becomes a compartmentalised and practically achievable task. With such a system in place, given appropriately robust hyper-parameter definition, it makes for a system that, in theory, would be capable of training a significant variety of robot arms that fit within a given category of structure.

## 7. Appendix


1 Closed-loop acquisition of behaviour on the Sphero robot - Oswald Berthold∗ and Verena V. Hafner∗

∗Adaptive Systems Group, Dept. of Computer Science, Humboldt-Universität zu Berlin{bertolos|hafner}@informatik.hu-berlin.de

2 https://blogs.nvidia.com/blog/2018/08/02/supervised-unsupervised-learning/

3 Learning of Motor Control from Motor Babbling* - Tatsuya Aoki* Tomoaki Nakamura* Takayuki Nagai* The university of Electro-Communications, 1-5-1 Chofugaoka, Chofu-Shi, Tokyo 182-8585 Japan (e-mail: aoki@apple.ee.uec.ac.jp)

4 https://www.youtube.com/watch?v=qCn8lkacJz0

5 https://openai.com/blog/evolution-strategies/ - and all of its mentioned sources

6 https://www.youtube.com/watch?v=C4MUTIc-NB8

7 Self-Organizing Maps - Teuvo Kohonen

8 https://www.youtube.com/watch?v=JgvyzIkgxF0 - An Introduction to Reinforcement Learning

9 https://www.delta.tudelft.nl/article/robot-leos-first-steps

10 https://en.wikipedia.org/wiki/Evolution_strategy

11 https://en.wikipedia.org/wiki/Evolutionary_algorithm

12 http://www.scholarpedia.org/article/Reinforcement_learning

13 http://www.scholarpedia.org/article/Evolution_strategies

14 https://ai.googleblog.com/2020/04/exploring-evolutionary-meta-learning-in.html

15 https://en.wikipedia.org/wiki/Deep*reinforcement*learning

16 http://www.scholarpedia.org/article/Deep_Learning

17 https://en.wikipedia.org/wiki/Motor_cortex

18 https://en.wikipedia.org/wiki/Cerebellum

19 https://en.wikipedia.org/wiki/Motor_babbling

20 A Motor Control Model Based on Self-Organizing Feature Maps – Yinong Chen - https://drum.lib.umd.edu/handle/1903/908

21 Bio-inspired Intelligent System Design – Jan Jacobs

# Self-Teaching Robot Arm - Software Requirements Specification

by Valentine Ezekiev - 2971429

## 1 Introduction

This document will outline the Software Requirements Specification for this project. Below is given context to explain the project and its problem in greater detail. If any unknown terms are encountered, refer to chapter 1.3 for clarification.

### 1.1 Context

The organization GTL generally has a focus on(but is not limited to) developing/researching the use of newer technologies and concepts for the fields of agriculture, mechanical engineering and informatics. That's where this project comes in - in partnership with Fontys HS Venlo, GTL has started a project, the purpose of which is to develop a self-taught (via machine learning and un/semi-supervised machine learning at that) robot arm with the given end-goal of weed removal from crops. The student responsible for this project also needs to achieve this on one of the current ROS releases(as of writing this, ROS2 Foxy distro is presumed the best option) as a proof of concept that ROS can be used to significantly speed up the process of in-house created robotics software with next to no costs added. The purpose for that is because ROS has quickly become a very popular tool for robotics programming and is an open-source tool at that.

In addition to the self-taught robot arm, the project also requires the development of a framework system, based around ROS2, that allows for the dynamic training of similar, modular robot arms. including the aforementioned arm. GTL desires the development of such a system as it allows them to create custom robot arms in-house and develop their motor skills with (ideally) relatively few man hours involved.

Added below are 2 images of the latest proposed robot-arm design, for visual-aid. It should be stated that these images are works in progress and are subject to significant change :

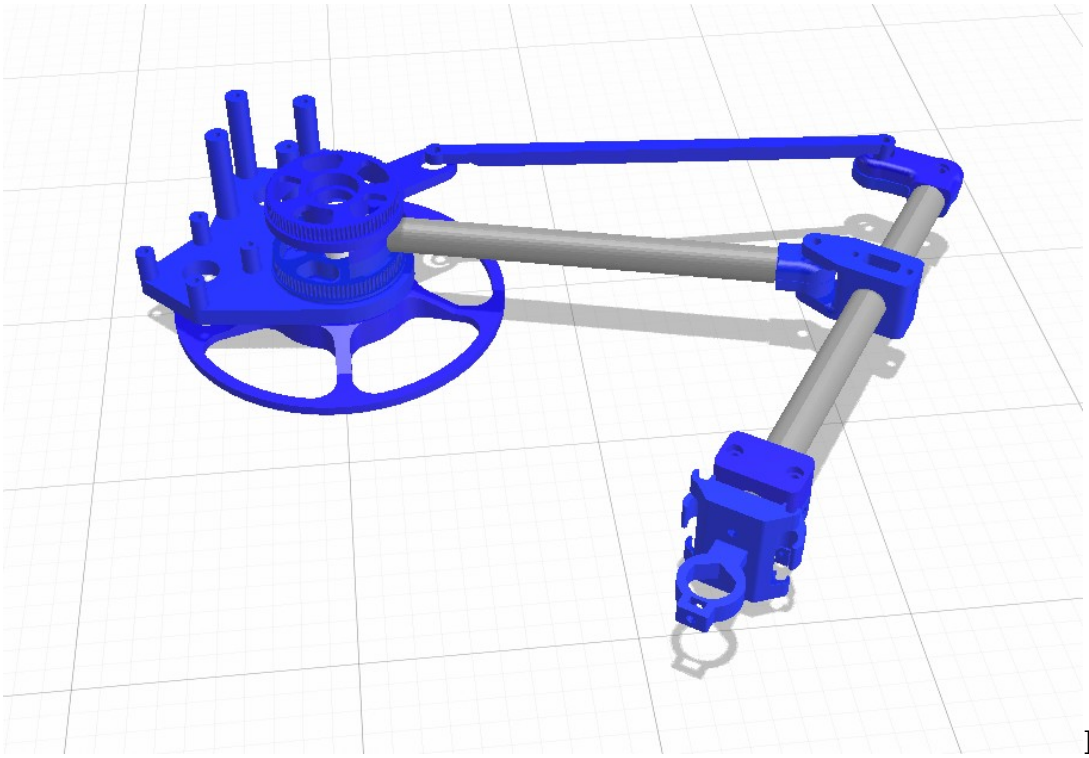

Figure 31: Early Proposed Arm Design

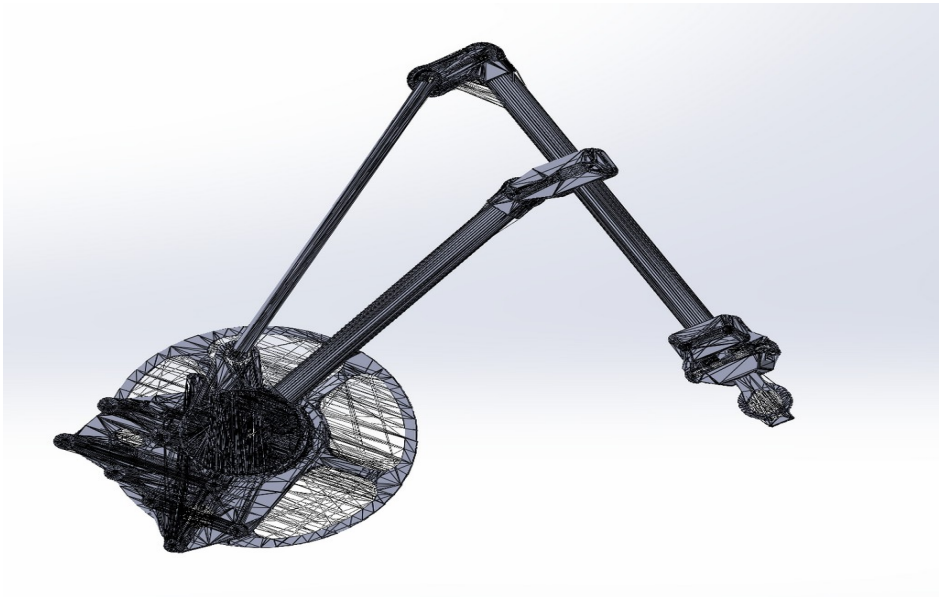

Figure 32: Early Proposed Arm Design

## 1.2 Scope

The scope of this project comparatively small - it regards only the robot arm, specifically the 3 general parts it will be comprised of - the segments, joints and appendage. Note - robot arms will not be limited just one of each of the 3 parts, save for the appendage. As of writing this, it is yet to be confirmed precisely what tool will be put on the end of the arm for the removal of weeds. It is expected to be something that can cut out weeds as a primary function, with a secondary function of delivering measured electrical shocks. The arm itself is expected to be attached to a larger(still altogether small) roving-type vehicle, however any functionality of the vehicle, as well as of visual object detection falls squarely outside of the scope of this project.

As far as the software scope goes, it is focused on 2 things:

A universal control scheme that can accommodate a variety of robot arms.

System of semi-supervised machine-learning to develop the controls/skills of individual robot arms with minimal human input.

## 1.3 Terms

GTL - GreenTech Lab

ROS - Robot Operating System

ROS1 and ROS2 - Essentially 2 different releases of ROS, ROS2 being the newer, more robust one. Historically, ROS2 lacked a lot of features and support, but it has finally achieved a level of completeness that makes it usable for general robotics and even usable in place of ROS1, the more established variant.

## 1.4 Goal

The goal of this project is to deliver a robot arm that has learned how to use its arm and given appendage to the best of its own practical availability. It must not be hard-coded or externally taught in nearly any capacity. The robot arm must learn how to efficiently work(meaning, remove weeds from crops) through its own forms of learning and experimenting. The robot arm should be able to adapt to and learn to overcome new challenges over the course of its "life", in addition to "remembering" how to continue overcoming the issues it had already learned to overcome.

All of the above listed has to be the result of some kind of complex unsupervised machine-learning setup. This setup should work with not just the 1 robot arm, but in-fact be modular.

# 2. Specification

## 2.1 Operating Environment

This robot arm is expected to be performing out in crop fields. Therefore the system will need to be able to account for mud, rain, wind, ground-level variation and temperature variation upon its given task of removing weeds.

## 2.2 Assumptions and Dependencies

It is assumed that the robot arm will be attached to an already functional roving-type vehicle which will provide power, visual input, stability and reach for the robot arm to perform its task(s).

It is assumed that direct sensory input will not be the job of the robot arm. Tasks such as object recognition fall outside of the spectrum of what the arm is supposed to learn.

It is assumed that this vehicle will be capable of maneuvering successfully and will not be influenced by weather effects such as rain water, mud, wind, variances in temperature or otherwise.

It is assumed that the robot arm will not need to deal with any unknown/mutated/highly specialized forms of weeds that would prove conventionally unsafe for removal.

# 3. Requirements

## 3.1 Hardware Requirements

1 Actuating robot arm with at least 1 degrees of freedom. Robot arm components are limited to:

> 1 or many connective inter-joint segments. These are not limited to straight line structures.
>
> 1 or many rotating motor joints.
>
> 1 simple appendage - the appendage must be simple in nature, having only states of being turned on or off and respectively affecting a designated area while on. Due to the fact that an appendage could theoretically be a nearly infinite amount of possible structures, it is deemed impractical to construct a universal system that accounts for complex and varied appendages.

1 Robot arm weed removing appendage - currently unspecified, but it must be capable of removing weeds at the very least via physically removing them or destroying them where they lay. Primarily it should do this via rotatory slicing, however a secondary functionality that it should have should be the ability to deliver an electric shock to kill a given weed plant.

The robot arm must only use single-axis rotating motors for its joints.

### 3.2 Software Requirements

A full control scheme for the robot arm and appendage based in ROS2. This means that ROS2 should be able to send actuation commands to each motor of the robot arm.

It must be able to actuate individual joints.

It must be able to activate and deactivate the appendage.

The robot arm should be able to fully interpret and execute given ROS2 complex commands(excluding movements that would physically block each other from happening or are simply outside the spectrum of movement for a given segment/joint)

A system that allows for the programmatic definition of modular robot arms.

A system that allows for the semi/un-supervised training of a robot arm that follows the structural requirements of this project.

Testing methods that verify the functionality of a robot arm, keeping in mind given requirements.

Code that is written in an extendable and modular way, allowing for easy modification.

A docker image of the final work environment, for easy replication later on.

Documentation of the software.

Design diagrams to provide an overview on the system and it's problem domains.

### 3.3 Machine Learning Requirements

The robot arm must be able to teach itself how to "correctly" move via semi/un-supervised machine learning. Correctly, in this scenario, means:

The robot arm must be able to extend to and operate within its determined physical range limit.

The robot arm must maneuver itself in a way that does not disturb or collide with anything on the rover around said robot arm.

The robot arm should have general collision avoidance (meaning, must not collide with self).

The robot arm must be capable of teaching itself how to move and how to do so in a desired way with minimal human input.

The robot arm must be capable of learning new behaviours in addition to existing ones, given a set of parameters and a learning objective.

The robot arm must be capable of altering learned behaviours, to accommodate for dynamic change in the requirements for already established behaviours.

The machine-learning setup must be dynamic. What this means is that it should be capable of accommodating a variety of robot arms that share the defined structure from 3.1.

### 3.4 Functionality Requirements

The robot arm must be capable of removing a weed that would impede the growth of human-planted crops without harming or impeding the given crops.

The robot arm must be capable of removing a weed physically via cutting. Currently cutting is deemed the best approach, as it is the most precise and least intrusive on the crop plants whilst also leaving the biological material of the weed in and on the ground. The roots will be able to rot this way and provide nutrition for the crop plants whilst the cut half can be used for mulch.

The robot arm must be capable of electrocuting weeds to the point of biological death.

The robot arm must not get stuck or awkwardly collide with the environment around it or the rover it is attached to.

# Design Justification

## 1.Intro

This document will serve as a brief explanation of note-worthy design decisions, so that it can be easy to see the reasons for systems being the way they are.

## 2.System Overview

The overall system is designed on the principles of bio-mimicry.

- Mimicking the processes naturally found in human babies, the system replicates the following - an initial training loop of motor babbling, where the robot arm, once it has all of its hyperparameters inputted by a human operator, will attempt all possible movements that it can perform, save those movements and their results as data and organise the data into a Self Organized Map. This SOM can then be used for sequential combinations of singular movements in order to perform broad, smooth motions with the arm. This replicates how babies train their motor-cortex at an early age by flailing their limbs about with reckless abandon. This teaches them how to move, walk, etc.

- Continuing to mimick from humans, reinforcement learning - humans generally learn via reinforcement learning when they try to learn a specific skill. By repeating an action multiple times with a given goal and measuring the results each time, a human zeroes in on the best approach to performing a given task. In similar fashion, the robot arm, once again with hyperparameters set by a person, will learn to perform a given task with given rules. Via many cycles of determining which sequences of actions reap the best rewards and which - the worst, the robot learns new skills. This is saved in a separate neural network, mimicking again how humans generally store more specialised learned skill in their cerebellum, separate from the motor cortex.

## 3.Motor-Babbling design

Important notes on the system:

- - The Appendage - the system is designed to work with any variety and combination of robot arm that follows the following structure - a varying number of degrees of freedom, with defined joints with their turn angles and defined connective segments and their lengths. Each robot arm is expected to be tipped with a simple appendage that can either be activated or de-activated, having a defined cubit area or point of action. Things such as fingered grabbers or human hand analogues are not considered for this system, due to the fact that they would significantly increase complexity. Making a universal robot trainer by an unqualified student is difficult enough, but making an appendage trainer system that can account for appendages of varying numbers of joints and segments would vastly complicate system design needlessly. But it is worth noting that in theory, it isn't anything new - it would still be just joints and connective segments. it's just that it would be very difficult to account for all the possible combinations of joints and segments that could make an appendage.
- - Only Works with single-axis ROTATING Motors - this system is only designed to work with rotating motors. While it could, for example, be designed to account also for the possibility of pistons, this falls outside of the scope for this project. In addition, accounting for more and more modularity significantly increases the complexity of this project, which is undesirable. It is meant to train robot arms, and following the logic of biological arms - they are based off of a motor-segment-motor-segment design and not pistons.
- - The simulation has to have gravity turned off - this is due to limitations of Gazebo in terms of providing joint objects that offer locking functionality like we would have in real life. This way, a joint can be placed any which way without succumbing to gravitational intertia, which, in terms of this simulation process, doesn't affect the end goal - the end goal being just calculating all the possible motor rotation value combinations and appendage positions relative to those joint combinations. IT IS IMPORTANT to note that this is primarily done due to severely lacking functionality from Gazebo and ROS2 as of writing this. It is a WORKAROUND and does not present the

absolute best conceivable solution, merely the most practical current-to-long term one given the circumstance.

- - Robot arms HAVE TO HAVE sequentially lessening weight for each consecutive motor and joint combo, starting from the base. So essentially - the first motor and segment of an arm will weigh 100 kilos each, the joint+segment after them will be 50, then 25 after that and so on. This is done as a workaround to a problem encountered with inertia. Due to the fact that Gazebo does not offer very realistic depictions of joints, a joint currently only has one of 2 ways to be moved - either by spawning it in a given rotation, or by applying a kinetic force that spins it, like a bicycle wheel. The problem with this is that during simulation, joints that go farther and farther from the base joint, while they are doing their own movements and calculations, affect the position of each preceding joint with their generated inertia. This causes obvious problems for determining from where to where a joint can turn given set positions of its preceding joints, so using this workaround - we negate the problem.

- Once again, this IS A WORKAROUND due to the fact that, as of writing this, Gazebo and the current iteration of ROS2 suck a lot and are missing a bunch of stuff. For all our practical purposes, this works just fine currently.

- - The simulation is set to run, essentially, as fast as systematically possible. This is done, obviously, so that time can be saved when running the simulation. As far as I can tell, this has no strange buggy effects on the physics of the simulation, and genuinely just increase the speed in a nice and useful way.

- - Using a combination of spawning motors in given positions and applying force - as mentioned previously, as of writing this, Gazebo and ROS2 are missing a lot of good features that allow for the straightforward simulation of joints. The current 2 primary ways of moving a joint suck for the following reasons:

- - Applied force just essentially makes a single-axis joint just spin like a free-flow bicycle wheel. It's great for simulating collision, but super imprecise otherwise.

- - Spawning joints in a given orientation is super precise and doesn't let them move around all willy nilly - the problem being, it also negates collision, and allows for the spawning of joint orientations that force objects inside of other objects.

- To deal with this, a compromise using both of the afore-mentioned systems was designed. First a joint is spawned at its farthest right possible angle and then a force is applied to it to turn to its left, until it either reaches its natural turning limit or collides with something. Then, this same joint is spawned in every previously recorded position that did not have a collision, and the joint after it has the same first 2 steps applied. This process cascades until all possible non-collision movement combinations are recorded. The reason spawning is used instead of just applying force on the first joint is because of previously mentioned issues - imprecision and inertia make this impossible.
- - It is very important to have all segments and joints inside of a simulation with a minimal weight of 1 kilogram. I encountered some weird bugs otherwise, where joints would clip off into the distance.
- - The system uses recursion and nested cascading objects to record all of its joint values in the simulation and save them to a file once they are fully processed. Experiments were done with arrays or even program-external files that recorded all the data, but all of these approaches ended up being messy, more code intensive and just way less succinct.
- - There are only 4 sets hyperparameters that need to be set for this entire motor-babbling system - names of the joints in the simulation, amount of force for each joint necessary to move in the simulation, the name of the appendage in the simulation and the names of all the different topics/services that everything will be posted to, as defined in the simulation world itself. Other than that, everything else is pretty automated, so as to save as much effort as possible for the person using this package.