



EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO
NACIONAL DE MÉXICO



Instituto Tecnológico de Chihuahua II
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

**Reinforcement Learning module for personalized assisted
pedaling in Human-Electric hybrid vehicles (Pedelec)**

TESIS

PARA OBTENER EL GRADO DE

MAESTRO EN SISTEMAS COMPUTACIONALES

PRESENTA

Carlos Ricardo Tonatiuh García Reyes

DIRECTOR DE TESIS	CODIRECTOR DE TESIS
DR. HERNÁN DE LA GARZA GUTIERREZ.	DR. RAFAEL SANDOVAL RODRÍGUEZ

CHIHUAHUA, CHIH., 5 DE JULIO DEL 2021

Dictamen

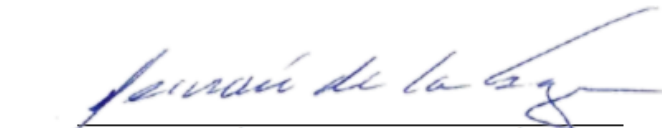
Chihuahua, Chih., 30 de agosto 2021

M.C. MARÍA ELENEA MARTÍNEZ CASTELLANOS
COORDINADORA DE POSGRADO E INVESTIGACIÓN
PRESENTE

Por medio de este conducto el comité tutorial revisor de la tesis para obtención de grado de Maestro en Sistemas Computacionales, que lleva por nombre "REINFORCEMENT LEARNING MODULE FOR PERSONALIZED ASSISTED PEDALING IN HUMAN-ELECTRIC HYBRID VEHICLES (PEDELEC)", que presenta el C. CARLOS RICARDO TONATIUH GARCÍA REYES, hace de su conocimiento que después de ser revisado ha dictaminado la APROBACIÓN de la misma.

Sin otro particular de momento, queda de Usted.

Atentamente
La Comisión de Revisión de Tesis.



DR. HERNÁN DE LA GARZA GUTIÉRREZ
Director de tesis



DR. RAFAEL SANDOVAL RODRÍGUEZ
Co-Director



M.I.S.C. JESUS ARTURO ALVARADO
GRANADINO
Revisor



DR. ALBERTO CAMACHO
Revisor

Dedicatoria

Esta tesis está dedicada a mi Madre Ricarda Reyes Duran, quien me dio el regalo de la vida dos veces. Gracias por siempre estar a mi lado, me rescataste de las cadenas de la obscuridad y perdición. Que este documento sea testigo de todo el esfuerzo y logros que jamás hubieran sido posibles sin ti.

Agradecimientos

Laura Madeline Gámez Cárdenas, gracias por compartir tu consejo y apoyo en los momentos difíciles, esta maestría no la podría haber terminado sin ti mi estimada amiga. Eres una raíz de vida para mí y muchos más.

Verónica Guadalupe Dávila Rodríguez, te agradezco por ayudarme a salir adelante y ser un pilar que me sustentó en circunstancias complicadas y días oscuros. Comparto este logro contigo mi estimada amiga, lo logramos.

Abstract

This document contains the complete research for a Reinforcement (RL) Learning Artificial Intelligence (IA) for the automatic control of a PEDELEC power assistance module. The RL IA will use trial-and-error methods to determine the best available assistance for the PEDELEC rider using as reference the Torque produced at the crankshaft and the rider's heartbeat at one-second intervals. This data will be transformed into an RL Environment where an agent will learn from no previous experience how to increase or decrease the assistance level to maintain the rider's heartbeat at a comfortable level. The implementation of the RL will use the principles of TD(0) control and SARSA Prediction to achieve the goal.

Resumen

Este documento contiene la investigación completa sobre una Inteligencia Artificial (IA) de Aprendizaje por Refuerzo (AR) para el control automático de un módulo de asistencia eléctrica PEDELEC. LA IA de AR utilizará métodos de prueba y error para determinar la mejor asistencia disponible para el ciclista del vehículo PEDELEC utilizando como referencia el Torque producido en el cigüeñal y los latidos del corazón del piloto a intervalos de un segundo. Estos datos se transformarán en un entorno de AR donde un agente aprenderá, sin experiencia previa, cómo aumentar o disminuir el nivel de asistencia para mantener los latidos del corazón del ciclista a un nivel cómodo. La implementación del AR utilizará los principios de control TD (0) y Predicción SARSA para lograr el objetivo.

Table of Contents

I INTRODUCTION.....	1
1.1. INTRODUCTION.....	2
1.2. PROBLEM DEFINITION.....	4
1.3. SCOPE AND LIMITATIONS.....	4
1.3.1 Scopes.....	4
1.3.2 Limitations.....	5
1.4 JUSTIFICATION.....	6
1.4.1 Pedelec sub-optimal power control design.....	6
1.4.2 CO2 Emissions.....	11
1.4.3 Sedentarism in Mexico.....	14
1.4.4 Chihuahua, Mexico. Alternative mobility plan.....	16
1.5 OBJECTIVES.....	19
1.5.1 General Objective.....	19
1.5.2 Specific objectives.....	19
II STATE OF THE ART.....	21
2.1 PHYSIOLOGICAL MONITORING.....	22
III LITERATURE REVIEW.....	25
3.1 WHAT IS RL?.....	26
3.2 ELEMENTS OF RL.....	30
3.3 RL GENERAL MODEL.....	34
3.4 THE MULTI-ARMED BANDIT.....	34
3.5 THE MARKOV DECISION PROCESS.....	40
3.6 THE BELLMAN EQUATION AND OPTIMALITY.....	46
3.7 DYNAMIC PROGRAMMING.....	47
3.6.1. Value Iteration.....	47
3.6.2. Policy Iteration.....	48
3.2.3 Generalized Policy Iteration (GPI).....	51
3.7 MONTE CARLO METHODS.....	52
3.8 MONTE CARLO EVALUATION (PREDICTION).....	53
3.8 MONTE CARLO OPTIMIZATION (CONTROL).....	54
3.9 TEMPORAL DIFFERENCE LEARNING.....	55
3.9.1 TD Evaluation (prediction).....	55
3.9.2 SARSA.....	57
IV DESIGN AND METHODOLOGY.....	60
4.1 SOFTWARE REQUIREMENTS SPECIFICATION (SRS) FOR T-MEGALORIDE AI.....	61
4.1.1 Product Perspective.....	61
4.1.2 Product Functions.....	62
4.1.3 User Classes and Characteristics.....	62
4.1.4 Operating Environment.....	72
4.1.5 Assumptions and Dependencies.....	72

4.2 EXTERNAL INTERFACE REQUIREMENTS	72
4.2.1 USER INTERFACES	72
4.2.2 Hardware Interfaces	75
4.2.3 Software Interfaces	76
4.3 SYSTEM FEATURES	77
4.3.1 Data acquisition.....	77
4.3.2 Emulator.....	78
4.3.3 Reward Module.....	81
4.3.4 Reinforcement Learning module.	83
4.3.5 GUI.....	84
4.4 IMPLEMENTATION.....	87
4.4.1 Data acquisition implementation.	87
4.4.2 Emulator implementation.....	89
4.4.3 Reward Module Implementation	94
4.4.4 Reinforcement Learning Implementation	95
4.4.5 GUI Implementation	106
4.4.6 Modified Bicycle for data acquisition.....	117
4.5 TESTING	118
4.5.1. Data acquisition.....	118
4.5.2 RL Agent performance	119
V FINDINGS AND DISCUSSION.	120
5.1 DATA ACQUISITION RESULTS	121
5.2 RL AGENT PERFORMANCE RESULTS.....	125
5.3 RESULTS DISCUSSION.....	135
CONCLUSION	136
BIBLIOGRAPHY	139

Table of Figures

Figure 1.1. Laufmaschine.....	1
Figure 1.2. Riese und Müller "Jetstream"	2
Figure 1.3. Indiscriminate battery use.....	8
Figure 1.4. Battery discharged before finishing the route.....	8
Figure 1.5. Exhausting ride due overstrain for the sake of battery charge saving.	9
Figure 1.6. Graph of CO ₂ emissions by the IEA in 2019	11
Figure 1.7. Population and number of registered cars in Chihuahua 2000-2010 INEGI.....	12
Figure 1.8. Proposal for the first stage of a bicycle lane network.....	18
Figure 2.1. Electrocardiogram in 1887.	23
Figure 2.2. Hip acceleration (ACC_{HIP}) and measured and predicted oxygen uptake ($\dot{V}O_2$).	24
Figure 3.1. RL general model.	34
Figure 3.2. Basic epsilon-greedy method for exploration.....	38
Figure 3.3. Sutton and Barto experiment for 10-armed bandit.	38

Figure 3.4. Average performance of 10-armed test bed.....	39
Figure 3.5. Markov transition probabilities diagram.	42
Figure 3.6. GPI interaction diagram.....	51
Figure 4.1. T-MegaloRide project.	61
Figure 4.2 T-MegaloRide UML project.	63
Figure 4.3. RL_entity class.	63
Figure 4.4. Emulator class.....	66
Figure 4.5. Application class.....	69
Figure 4.6. T-MegaloRide AI GUI.	73
Figure 4.7. T-MegaloRide AI GUI Input Selector.....	73
Figure 4.8. T-MegaloRide AI GUI Emulator Torque Control.....	73
Figure 4.9. T-MegaloRide AI GUI Data Reading.	74
Figure 4.10. T-MegaloRide AI GUI Environment Status.	74
Figure 4.11. T-MegaloRide AI GUI BPM Level.....	74
Figure 4.12. T-MegaloRide AI GUI Reward Discounts.....	75
Figure 4.13. T-MegaloRide AI GUI RL histogram.	75
Figure 4.14 ATmega328 development board.	76
Figure 4.15. Test bicycle.	118
Figure 5.1. Torque generation at 13.5 N.m average.....	121
Figure 5.2. BPM response at 13.5 N.m.....	121
Figure 5.3. Torque generation at 15.5 N.m average.....	122
Figure 5.4. BPM response at 13.5 N.m.....	123
Figure 5.5. Torque generation at 18 N.m average.....	123
Figure 5.6. BPM response at 18 N.m.....	124
Figure 5.7. Human-made Torque policy evaluation.....	125
Figure 5.8. Human-made policy BPM response.	126
Figure 5.9. First training phase Torque evaluation.	127
Figure 5.10. First training phase BPM response.....	128
Figure 5.11. Second training phase Torque evaluation.	129
Figure 5.12. Second training phase BPM response.	130
Figure 5.13. Third training phase Torque evaluation.....	131
Figure 5.14. Third training phase BPM response.	132
Figure 5.15. Human-made policy vs SARSA performance comparative.	133
Figure 5.16. Human-made policy vs SARSA BPM response.....	134

Index of Tables

Table 1.1. Physical-sports practice percentage distribution of the population aged 18 and over	14
Table 2.1 Performances of heuristic and ML approaches..	24
Table 3.1 Markov Table.	41

I INTRODUCTION.

I INTRODUCTION.

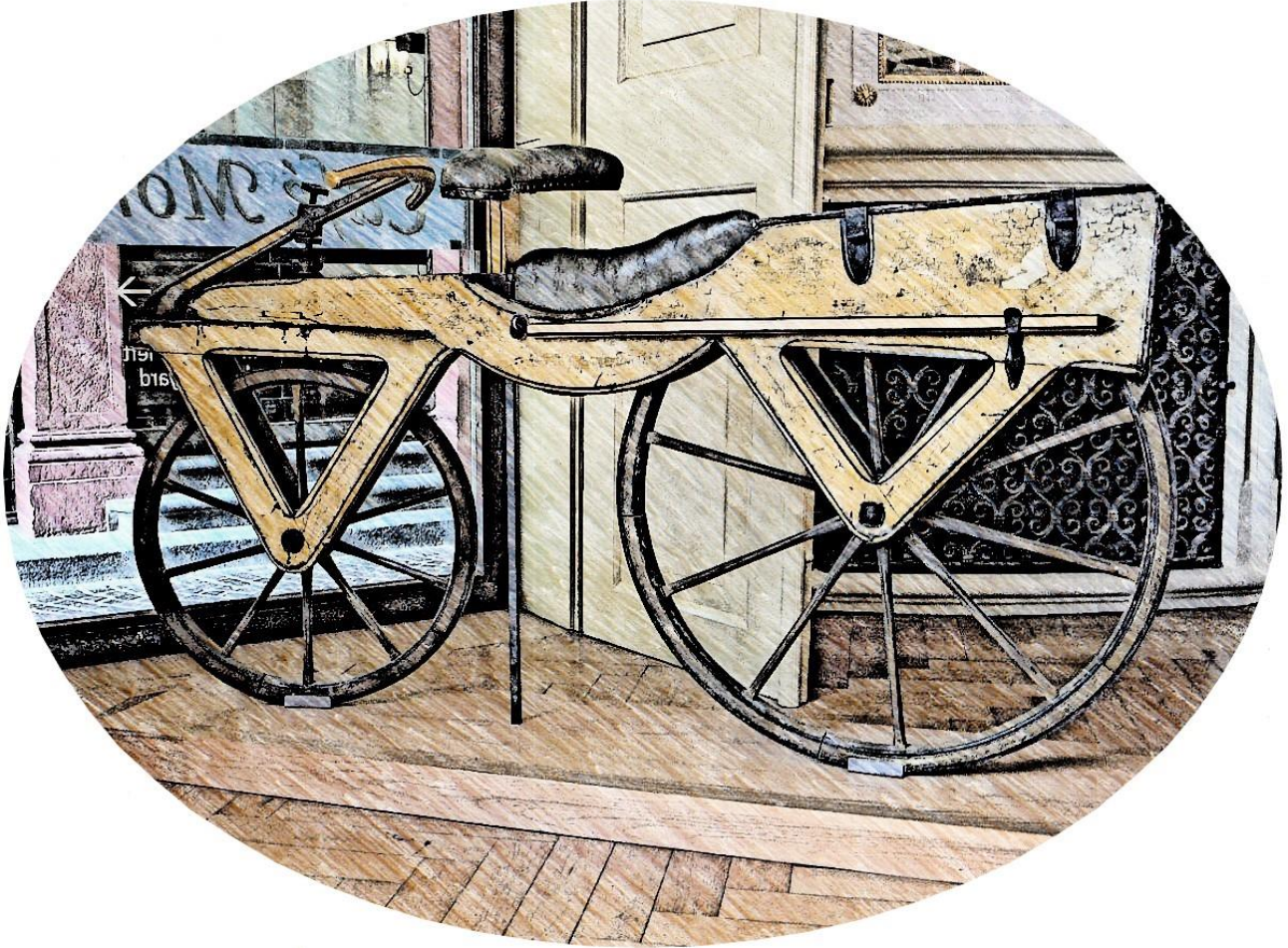


Figure 1.1. Laufmaschine¹

¹ Image from Gun Powder Ma CC BY-SA 3.0

I INTRODUCTION.

1.1. Introduction.

The bicycle, an invention that fascinated and redefined Human society from its conception in the 19th century, with the Laufmaschine ²(figure 1.1) *designed* by the German Baron *Karl von Drais* (Herlihy, 2004) considered the first human-powered, steerable, two-wheeled machine to be commercially successful, to the modern hybrid bicycles (Ballantine, 2001, p. 31) with electric assistance which results from the combination of the characteristics of highly specialized racing bikes (Ballantine, 2001, p.32), mountain bikes (Ballantine, 2001, p. 35) and very comfortable cruiser bikes (Ballantine, 2001, p. 28).

The impact of this vehicle in terms of technological advances is historically noteworthy, taking into consideration that some of its basic components were key parts for the development of the automobile (Heitmann, 2009, p. 11). Still, today they are an example of industrial advancement in terms of lightweight and resistant materials, more efficient mechanical energy transmission systems, the addition of pedaling assistance through electric motors, and in consequence, the experimentation with new battery technologies capable of extending the autonomy per charge.



Figure 1.2. Riese und Müller "Jetstream".³ Pedelec designed for the e-call a bike⁴ service by the German railway (Deutsche Bahn) in Berlin

2 From the German language: running machine.

3 Image from Chrischerf CC BY-SA 3.0..

4 Call a Bike is a dockless bike rental system, operated by Deutsche Bahn (DB) in several German cities.

I INTRODUCTION.

In 1986, Michael Kutter designed and produced the prototype vehicle designated as the Pedal Electric Cycle (alias dictus⁵ Pedelec), a bicycle-based vehicle where a revolutionary feature was incorporated: pedaling assistance provided by a small electric motor. This vehicle has a very specific feature where it is dictated that the amount of assistance is given in response to the pedaling effort and the maximum speed allowed, the latter being specified by local traffic laws for each country and/or geographical region. In Chihuahua City, Mexico, the speed is capped at 30 Km/hr (Chihuahua City Urban Development Plan: VISIÓN 2040, 2016, p. 192). This means that if the cyclist stops pedaling or the maximum set speed has been reached and/or exceeded, the assistance is immediately interrupted.

This kind of vehicle has had a great boom precisely due to the property of automatic pedaling assistance interruption, since it allows the vehicle to maintain the classification of a bicycle and not an automobile, mainly in Europe (Directive 2002/24 / EC of the European Parliament and the Council, 2002).

Despite all these properties and technological integrations, there is still a great area of opportunity for the improvement of the vehicle, as would be the case of the selection of the level of assistance, since this mechanism currently depends entirely on the manual interaction with the cyclist, who has to continually readjust the magnitude of the assistance in discrete levels (1-3 or 1-5 levels typically) preset by the manufacturer. This control model has an inherent disadvantage: the indiscriminate use of the capacity and charge of the battery that powers the motor. This can potentially drain the battery unnecessarily fast. It turns out to be a fascinating idea to have the ability to travel distances in the order of kilometers with an effort equivalent to walking a few blocks, using a system that automatically selects the proper assistance. The idea of implementing a Reinforcement Learning (RL) Artificial Intelligence AI to support automatic assistance power selection (a long-known issue) to a vehicle powered by the combination of Human and Electric power sources turns out to be quite a novel concept. It falls far from the research of light and resistant materials or efficient mechanical energy

5 From Latin, "also known as"

I INTRODUCTION.

transmission methods; but rather a vehicle with decision-making capabilities that is capable of offering great comfort to its cyclists using two key elements into consideration: Physical characteristics of the cyclist and sudden changes in the terrain/environment.

1.2. Problem definition.

Currently, commercial Pedelec controls rely entirely on the rider manual selection of the proper power assistance level, ergo,⁶ there is no guarantee of the proper use of available battery power nor an adequate level of physical strain on the rider, this leads to diverse undesired outcomes:

- **The Pedelec runs out of battery before finishing the route.** This is a typical problem related to the overuse of the Pedelec assistance capabilities.
- **Rider increased physical strain due to unpowered assistance system.** If a Pedelec runs out of battery power, the rider will have to provide the total power required to move the Pedelec, including the currently idle electric component's weight.
- **Unpleasant route due to overzealous battery power conservation.** At any given moment, the rider might choose to reduce the assistance power for the sake of battery power saving, which potentially leads to unnecessary rider's physical extra effort, defying the very purpose of having a power assistance module.

However, the environment is merely the beginning of a possible solution, which is why this research aims to analyze the feasibility of creating a profiling system that autonomously adjusts to each cyclist through policies generated from the constant use of the vehicle.

1.3. Scope and limitations.

1.3.1 Scopes.

- A sampling of fitness aptitudes in a volunteer test subject through specialized equipment developed for this purpose.

6 From Latin: therefore.

I INTRODUCTION.

- Creation of prototype policy as a baseline for the learning process of the RL SW module.
- Monitoring and RL agent training SW, with a GUI for visual inspection.
- Reward SW module with a separated policy for hearth beat and another for torque generation rewards. The module will provide a conjugated final reward.
- Emulator SW module that will behave in a similar style as the volunteer test subject (changes in Hearth rate when a Torque production level is changed). This module will provide all the required data for the RL agent training.

1.3.2 Limitations

- The implementation, functionality, mode of operation and communication interfaces with the hardware devices (HW) are outside of the interest in this research; Hence, the information provided in this document is limited to deal with such topics as Black Boxes ad referendum. The availability of the required data is to be considered as processed and available at all times. Such HW components include but are not limited to:
 - Torque sensor.
 - Photoplethysmography sensor.
 - Sensor signal processor interface (ATmega328P).
 - Communications and interface protocols (Serial-Over-USB).
- The operation of the assistance control module is not part of this research since the know-how of the electronic/electromechanical area is not within the scope or related to the IA. This device will be considered an external Black Box module ad referendum, and it will simply be considered the transmission of a scalar feedback value through the communication interfaces described above.

I INTRODUCTION.

- This research does not consider the details or the construction of the Pedelec vehicle to which the integration of the AI module is intended.
- The data obtained from the sensors are not to be considered of medical-grade reliability. Therefore, these measurements will not be recommended for any use other than the ones described in this research.
- The rider's requests for increased assistance to the AI module will be considered a bona fide.⁷ This is due to the fact that although variables and indicators can be observed, it will be the cyclist who will grant the final judgment in the comfort of the assistance.
- If the rider intentionally misuses the assist request system, the battery use optimization will be compromised, and under-performing results are to be expected.

1.4 Justification.

1.4.1 Pedelec sub-optimal power control design.

A Pedelec vehicle has all the electrical/electronic control components required by the pedaling assistance functionality, and such controls regulate the specific speed of the electric motor that acts as an auxiliary power supply. Some of these controls constrains are:

Presence of Revolutions Per Minute (RPM) on the pedals. Which is given as a discrete value: [Yes/No].

Assistance level. Selected by the rider, this acts as a power divisor from the incoming motor power source. This ranges between values of 1-3 or 1-5 (depending on the manufacturer).

Speed within limits. Taken as a discrete value [Yes/No], it depends on the manufacturer's maximum speed.

These values are combined in a single assistance function, which is later transposed into its counterpart in the field of electronic controllers.

⁷ From Latin: in good faith / honestly / genuine / real

I INTRODUCTION.

This methodology is quite robust in projects where the power supply has well-established operating characteristics. In the case of a modern electric motor manufactured with high-quality standards, the manufacturer provides all the required information through reliable documentation. Nevertheless, what happens if the power supply to be controlled has divergent⁸ and undocumented characteristics? What would the case for a living being? (Specifically, a Human being within the context of this research).

Given the irrefutable fact that each human being is unique, therefore, it is feasible to infer that the physical capacities of each human are unique in the same way, and not only that, these capacities constantly vary over time. This concept is so genuine that it can be considered the pillar from which the sports competitions have derived: evaluation and categorization of athletes through their physical performance.

This circumstance makes it challenging to create a mathematical model with a unique control function (one fit all) capable of giving specific power feedback for each individual. Therefore, the need for the power assistance selectors presented to the rider to determine the appropriate power at any given time. Also, it is implicit that it is the rider's responsibility to monitor and administrate the power level, and therefore, any overuse of the battery due to unnecessary assistance power is a direct rider's fault.

This assistance power micromanagement model is far from optimal since there are absolutely no constraints on the rider's decision of assistance power. This stance assumes two unwanted scenarios:

8 That are different for each case

I INTRODUCTION.

First, if the rider opts for a rather high level of assistance and the route to cover is uncomplicated (no steep hills, pavement, and easy to drive road), the battery charge available will be drained at an undesired (and probably unintended) high rate (figure 1.3), dramatically increasing the risk of an early battery discharge. This event would be a great inconvenience since, in addition to the rider's weight, it would be necessary to carry the electrical equipment (motor, controller, battery, and other components) as dead weight⁹ (figure 1.4).



Figure 1.3.¹⁰ Indiscriminate battery use.



Figure 1.4. Battery discharged before finishing the route.

⁹ Weight of inert things, it has to be carried or hold at the expense of an external force.

¹⁰ *Royalty free Computer vectors created by pch.vector*

I INTRODUCTION.

Second, if the rider opts for a level of assistance too low due to the inherent worry about draining the battery before finishing the intended route to cover, the rider is quite likely to experience a rather unpleasant crossing (figure 1.5). In a worst-case scenario, the rider could even finish the route with a substantial unused battery charge but quite exhausted (Which defies the principle of the comfort provided by the Pedelec vehicles, to begin with).

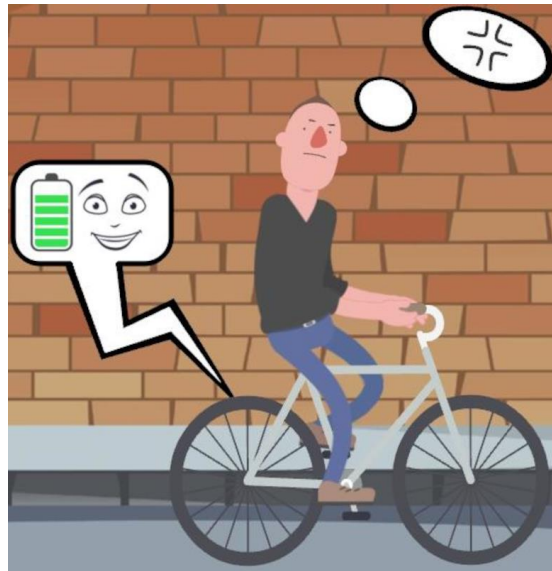


Figure 1.5. Exhausting ride due overstrain for the sake of battery charge saving.

One approach to this dilemma is to increase the power when there is a high slope and decrease it when it is low. A sensor can be attached to the Pedelec to determine the current slope and automatically adjust the assistance power level. In this implementation, the Rider does not exert too much effort, and neither does too much battery is wasted. However, how much is "too much" of effort for the Rider?

A new variable needs to be included in the dilemma: the Rider itself. In order to provide a perspective of the nature of the Rider, three different stands can be analyzed:

- A Rider who uses his Pedelec for weekend rides in the park.
- A Rider who uses his Pedelec to travel to and from work/school five days a week.

I INTRODUCTION.

- A Rider who attends extensive physical training and enjoys off-road mountain trails on a regular basis.

An effort to quantify how much is "too much", using the feedback from these individuals will very likely produce unreliable results due to the divergent capacities of each subject. This would force the original approach to add an assistance control to select what kind of Rider is intended to use the Pedelec, but this will be a fallback to the original dilemma, the need for manual input provided by the Rider.

In short, we need an assist control that can determine when the level of effort is too high or too low for each potential Rider in an intrinsic design. This requirement alone would prove to be an impressive task. How can an autonomous system determine the required assistance power feedback to every possible Human being?

It is in this precise area where RL methodologies are ideal: difficulty in creating mathematical models, but feasibility both in the acquisition of data to be evaluated and expected results, which later serve as discriminators for RL training.

The scope of personalized automatic assistance control lends itself well to applying the RL paradigm. Reinforcement learning is learning what to do, how to map situations to actions, so as to maximize a numerical reward signal (Sutton and Barto, 2018, p. 1).

Given that there are electronic accessible and economical devices (known as gadgets) that allow the evaluation of specific characteristics of the cyclist's physical state, like the heart rate and torque produced in the vehicle's pedals, that provides additional information that can be integrated with the already established control variables of the Pedelec, a well-defined environment can be created. This means that all the hardware constraints meet.

1.4.2 CO2 Emissions.

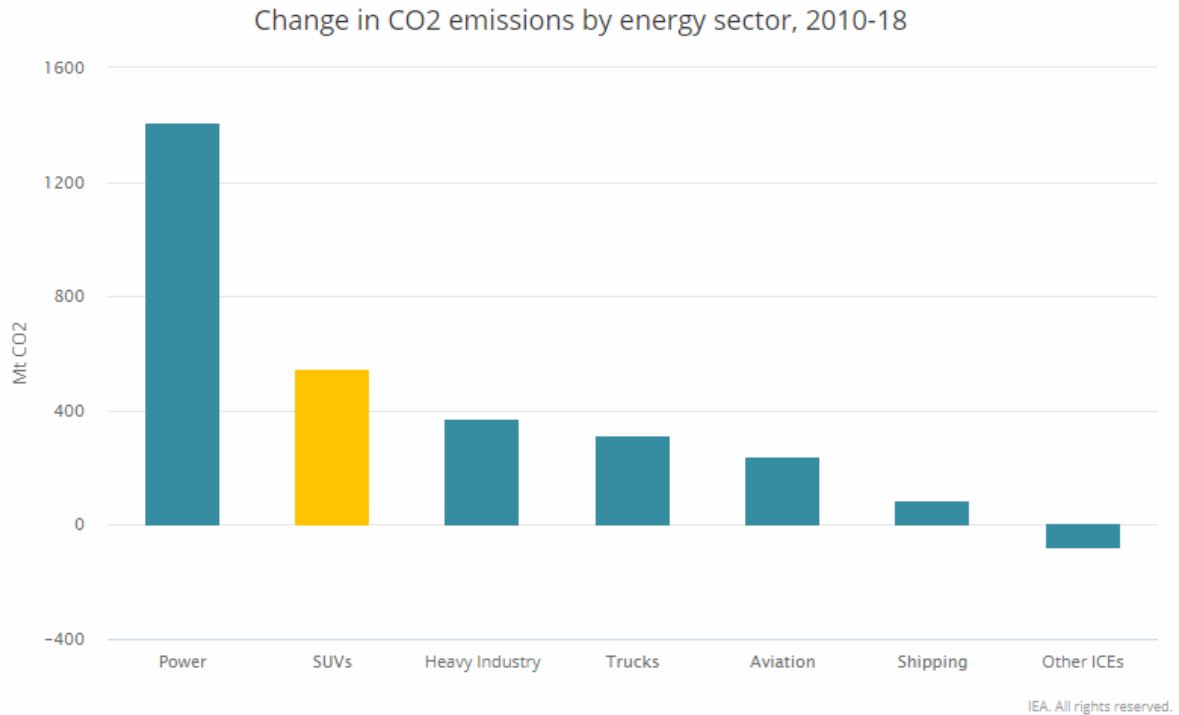


Figure 1.6. Graph of CO2 emissions by the IEA in 2019

According to the International Energy Agency¹¹ despite the new findings in the development of alternative energy engines, internal combustion vehicles represent almost a quarter of the daily global demand for oil (Cozzi, 2019) “[A] dramatic shift towards bigger and heavier cars has led to a doubling of the share of SUVs¹² over the last decade. As a result, there are now over 200 million SUVs around the world, up from about 35 million in 2010 (Cozzi, 2019)”.

This situation leads to a great negative impact on CO2 emissions (figure 1.6), “The global fleet of SUVs has seen its emissions growing by nearly 0.55 GtCO₂¹³ during the last decade

11 An autonomous intergovernmental organization based in Paris established within the framework of the Organization for Economic Cooperation and Development (OECD) in 1974 in the wake of the 1973 oil crisis.

12 *Sport Utility Vehicle*.

13 Un gigatonne es mil millones de toneladas. "GtCO₂e" es una abreviatura de "Gigatoneladas de dióxido de carbono equivalente"

I INTRODUCTION.

to roughly 0.7 Gt CO₂. As a consequence, SUVs were the second-largest contributor to the increase in global CO₂ emissions since 2010 after the power sector, but ahead of heavy industry (including iron & steel, cement, aluminium), as well as trucks and aviation.”

Narrowing the statistics available to the state of Chihuahua, Mexico, we find that in the last population census granted by the National Institute of Statistics and Geography (INEGI) in 2010, the number of people between 15 and 69 years old registered amount 2,196,150. In the same year, 1,214,904 cars are registered in the state (figure 1.7), which gives us a surprising number of 0.55 cars per person. This information leads us to a simple inference, for each residence with 4 inhabitants with ages between 15 and 69 years old, there would be 2 automobiles. For the year 2018, 1,599,601 (INEGI, 2019) cars were registered in the state, showing a positive tendency.

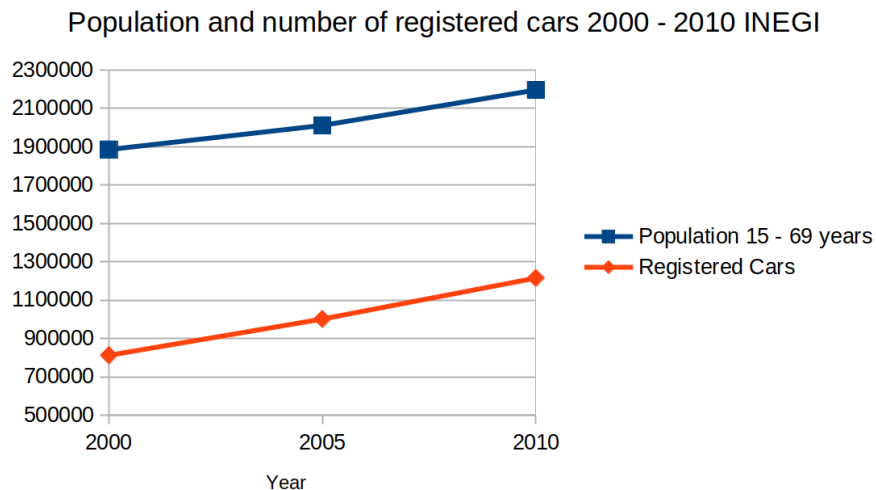


Figure 1.7. Population and number of registered cars in Chihuahua 2000-2010 INEGI

It should be noted that the number of automobiles given does not contemplate a large number of irregular cars present in the State since its own irregular denomination makes it impossible to provide reliable amounts. However, it allows us to claim that the values shown here are below the real scenario.

I INTRODUCTION.

The data presented allows us to see a correlation between the increase in population and the number of registered cars, which infers an impact on CO2 emissions with a positive slope, which is worrying since they are direct data from the State in which the current research is conducted.

On the other hand, it offers us a perspective of the State of Chihuahua's population economic stance, speaking specifically in the procurement of internal combustion vehicles, despite current adversities, such as the continuous increase in gasoline costs. Ergo, there is an area of opportunity for the insertion of low-cost Pedelec vehicles in the market.

The advantage of a Pedelec vehicle against its internal combustion engine vehicle counterparts is, in simple words, abysmal since electric vehicles batteries can be fully charged by green energy technologies in any of its variants and any combination. This advantage still puts Pedelec vehicles far ahead of internal combustion engines, restricted to the accessibility and local fuel pricing (either gasoline or petrol).¹⁴

Given the advantages already described, a vehicle with the proposed characteristics could be considered as a "feasible" substitute for similar vehicles that use internal combustion engines (like motorcycles). This, coupled with the global effort to change to vehicles with clean energy systems, Pedelec vehicles are inherently given the ability to recharge batteries through solar, wind, hydraulic/hydroelectric systems, etc. While providing the benefits of exercise minutes on the daily routine as a plus.

This is of paramount importance since CFE¹⁵ currently offers the photovoltaic services contract in Mexican homes where "[...] in addition to saving on your energy consumption expense, you will contribute to the use of clean technologies for the generation of electric

14 Any of various flammable, volatile liquid mixtures of hydrocarbons, primarily octane, obtained from petroleum and used as a solvent and fuel for internal combustion engines. In general, petrol also contains additives such as anti-knock compounds and corrosion inhibitors.

15 From the Acronyms in Spanish Comisión Federal de Electricidad (Federal Electricity Commission)

I INTRODUCTION.

energy, in the use of renewable energy sources and therefore, in the conservation of the environment. (CFE, 2019)".

This research intends to create interest within the local population in the Pedelecs market by further increasing the current commercial Pedelec engaging benefits, employing state-of-the-art RL to improve the long-standing battery range issue. The first approach intended is the diffusion of resulting findings on the scientific community, which will serve as a cornerstone for further interest and research on all related fields of electric vehicles, ever-increasing the Mexican technology scope on the field.

1.4.3 Sedentarism in Mexico.

Table 1.1. Physical-sports practice percentage distribution of the population aged 18 and over, by age groups and gender (MOPRADEF, 2019)

Percentage distribution of the population aged 18 and over, by age groups and gender, according to the condition of physical-sports practice in their free time, Level of sufficiency, and condition of Physical-sports practice in the past								
November 2018								
Age and Gender groups	Population aged 18 and over	Physically active			Physically inactive			
		Total	Level of physical-sports sufficiency		Total	Physical-sports practice in the past		
			Sufficient¹	Insuficiente²	Undeclared³		Once Before⁴	Never⁵
Urban aggregate of 32 areas with 100,000 and more inhabitants	38 856 716	41.7	52.4	44.6	3.0	58.3	72.6	27.4
18 to 24 Years	6 311 438	49.8	56.4	42.7	1.0	50.2	80.8	19.2
25 to 34 Years	7 846 913	45.3	52.1	45.9	2.0	54.7	78.6	21.4
35 to 44 Years	7 936 013	42.2	50.3	43.5	6.2	57.8	76.8	23.2
45 to 54 Years	7 080 415	38.9	51.5	45.5	3.1	61.1	70.5	29.5
55 to 64 years	5 241 674	39.5	49.8	47.1	3.1	60.5	65.5	34.5
65 Years or more	4 440 263	29.9	54.8	42.7	2.5	70.1	60.1	39.9
Male	18 530 183	48.4	48.9	48.2	2.8	51.6	85.0	15.0
18 to 24 Years	3 334 761	59.0	53.2	46.8	0.0	41.0	83.6	16.4
25 to 34 Years	3 885 467	53.3	46.5	52.4	1.1	46.7	84.0	16.0
35 to 44 Years	3 670 473	45.9	43.0	48.8	8.2	54.1	89.8	10.2
45 to 54 Years	3 363 585	45.5	51.2	47.0	1.9	54.5	83.8	16.2
55 to 64 years	2 312 783	42.7	42.1	54.2	3.7	57.3	85.9	14.1

I INTRODUCTION.

65 Years or more	1 963 114	36.6	62.9	33.5	3.6	63.4	81.2	18.8
Female	20 326 533	35.6	56.6	40.1	3.3	64.4	63.6	36.4
18 to 24 Years	2 976 677	39.6	61.6	35.8	2.6	60.4	78.7	21.3
25 to 34 Years	3 961 446	37.4	59.9	36.9	3.2	62.6	74.6	25.4
35 to 44 Years	4 265 540	39.0	57.7	38.2	4.1	61.0	66.9	33.1
45 to 54 Years	3 716 830	33.0	51.9	43.6	4.6	67.0	60.6	39.4
55 to 64 years	2 928 891	37.0	56.8	40.7	2.5	63.0	50.9	49.1
65 Years or more	2 477 149	24.6	45.2	53.5	1.3	75.4	46.1	53.9

MOPRADEF considers “physical-sporting sufficiency” when the population aged 18 and over comply with frequency (days), duration (minutes), and intensity (moderate or strong) per week according to the World Health Organization (WHO) recommendations to obtain health benefits.

1 With a sufficient level of physical-sports practice: They carried out physical-sports practice in their free time last week, at least three days a week, and accumulating a minimum time of 75 minutes with vigorous intensity or 150 minutes with moderate intensity.

2 With an insufficient level of physical-sports practice: They carried out physical-sports practice in their free time last week, less than three days a week or not accumulating the minimum time of 75 minutes with a vigorous intensity or 150 moderate intensity.

3 With an undeclared level of physical-sports practice: They do carry out physical-sports practice in their free time, but last week they did not.

4 They practiced some time ago: They do not do physical-sports practice in their free time and declare that they have practiced any of these before.

5 Never done physical-sports practice: They do not do physical-sports practice in their free time and declare that they have never done any practice before.

The estimates that appear in this table are colored according to their level of precision: High, Moderate, and Low; taking as reference the coefficient of variation CV (%). A Low precision requires a cautious use of estimation in which the causes of the high variability are analyzed, and other indicators of precision and reliability are considered, such as the confidence interval.

Level of estimated precision levels:	
High - CV in the range of (0,15)	
Moderate - CV in the range of [15, 30)	
Low - CV from 30% onwards	

This research also aims to be a potentially beneficial factor on the Mexican community health, where the culture of pedal vehicles is present on a day-to-day basis as a medium of urban transport, utility, and leisure trips in general. An intelligent vehicle that adapts to the user's physical capabilities could softly increase the daily exercise quota, increasing the riders' physical conditioning (over time) of those who choose to use a Pedelec vehicle as an alternative to current mobility options like taxi services.

I INTRODUCTION.

1.4.4 Chihuahua, Mexico. Alternative mobility plan.

The objective of the alternative mobility strategy of the city of Chihuahua is to create adequate conditions of infrastructure, equipment, and cycling culture to allow the bicycle to be positioned as a viable and safe alternative for daily transport. (Plan de Desarrollo Urbano de la ciudad de Chihuahua: VISIÓN 2040. 2016).

Currently, there are works and plans to improve and delimit sections on the streets of Chihuahua City towards the improvement of alternative transportation, which results in a great benefit to the investment of cutting-edge technologies that improve the performance of such alternative vehicles and offer yet another attractive feature for the general interest of the population.

“For the adequate planning and programming of the bicycle road network, a Bicycle Mobility Plan (PMB) must be carried out from which a general program of actions is derived that guarantees the success of the general bicycle road system and its adequate use as an alternative in the intermodal mobility network, as well as its correct and effective interconnection. For the above, the following general objectives are proposed:

- Make Chihuahua the leading bicycle mobility city in northern Mexico.
- Make bicycle mobility a public policy.
- Generate social and infrastructure actions to encourage the use of bicycles as an effective, safe, and attractive mode of mobility.
- Promote and articulate actions within current efforts in non-motorized mobility and public space.” (Plan de desarrollo urbano de la ciudad de Chihuahua: VISIÓN 2040. 2016).

“As a result of the analysis works of the current situation of the city, population density, areas identified as priority attention (ZAP), DENUES, BRT system, mobility corridors, analysis of proposals in various specialized workshops in non-motorized alternative mobility carried out

I INTRODUCTION.

as CICLO CIUDADES and ITDP Mexico, a first stage is proposed for the network of bicycle roads that guarantee its use as an efficient and competitive means of transport, so that during its execution and consolidation process the previously proposed Bicycle Mobility Plan, that promotes the continuity of the system (figure 1.8).

This first stage corresponds to the sections described below:

1. *Ciclovía Picacho*.- it connects the north campus of the UACH with the north terminal of the BRT system.
2. *Ciclovía Cantera*.- It connects the existing network of bike roads in the western area of the Periférico de la Juventud with the Palomar and Zona Centro area through rights of way, restrictions and parks along the Cantera road.
3. *Ciclovía Teófilo Borunda*.- It connects the metropolitan park of the 3 dams with the Sacramento linear park through the Teófilo Borunda road, passing through the polygon of the historic center.
4. *Ciclovía Tecnológico*.- It connects the Riberas del Sacramento I and II neighborhood to the north with the Historic Center by Av. Tecnológico, passing through the UACH north campus, the BRT North Terminal, ITCH, the UACH campus and Deportiva.
5. *Ciclovía Ch-P*.- Connects the southwest ZAP with the central ZAP through the Ch-P road, taking advantage of the railroad rights.
6. *Ciclovía Pacheco-Nueva España*.- Connects the central ZAP with the southern ZAP, this being the most prominent priority area of attention in the city, taking part of Pacheco Avenue and Nueva España as a route, passing through the southeast sub-center, bus terminal, Deportiva Sur, Hospital Infantil, South Terminal of the BRT, southern police headquarters, CBTIS 122, crucial industrial zone and commercial corridors.

I INTRODUCTION.

7. *Ciclovía Equus*.- Connects the Aeropuerto I bike-way along the entire Equus avenue and connects several neighborhoods to the city's east.
8. *Ciclovía Aeropuerto I*.- Connects the Nueva España bikeway with the Equus bikeway.”
(Plan de desarrollo urbano de la ciudad de Chihuahua: VISIÓN 2040. 2016).

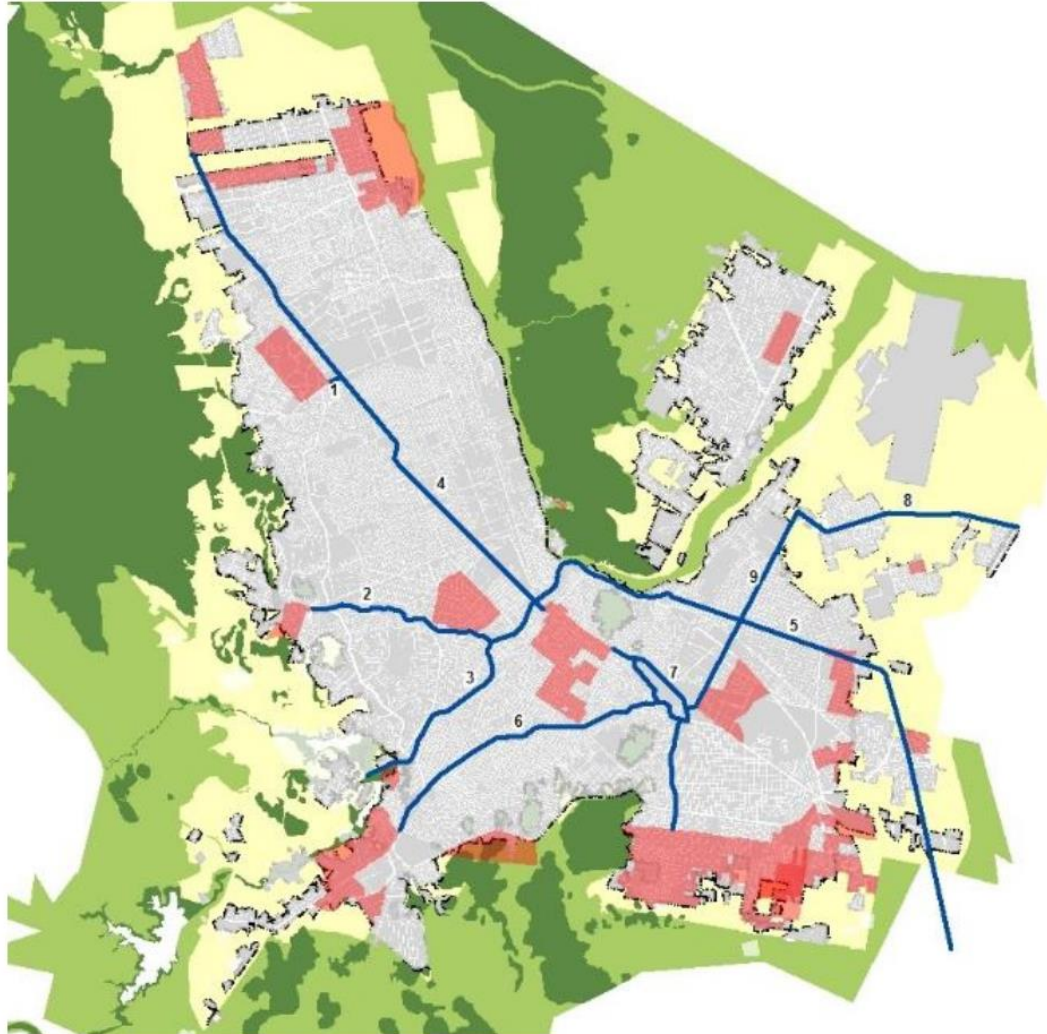


Figure 1.8. Proposal for the first stage of a bicycle lane network. By Urban Development Plan of the City of Chihuahua: VISION 2040 (fifth update) 2016. ■ First stage cycle paths, ■ ZAP and hubs.

I INTRODUCTION.

The Chihuahua Alternative mobility plan comes as yet another potential benefit for the research on Pedelec technologies since there will be local cycle paths where the intelligent Pedelecs can move within protected areas where internal combustion counterparts can't.

1.5 Objectives.

1.5.1 General Objective.

Develop a Reinforcement (RL) module capable of provide personalized feedback to the control module of a Pedelec assistance system, to enhance it with the ability to automatically adjust the assistance, using the cyclist's physical condition as a reference.

1.5.2 Specific objectives.

Adapt a generic mountain bicycle for sampling.

A generic mountain bike will be equipped with the necessary sensors for the compilation of statistical data from test subjects.

Create a SW interface to capture the sensor data from the test bicycle.

A SW interface will be created using the high-level programming language Python 3.8.

Test subject supervised sampling.

A sampling session will be performed on an individual volunteer who must perform a series of tests to obtain physiological data reference.

Initial policy creation.

A prototype policy will be created to represent a human-picked set of actions that would be considered good behavior. The policy will use the results of the test subject sampling as a reference.

Development of SW module for calculating rewards.

I INTRODUCTION.

The module will have two rewards sections, one for the produced/requested torque and another for the hearth beat readings. Both rewards will be combined to determine the total reward for any given state.

Development of a SW interface for the cyclist to provide feedback.

An interface will be developed where the cyclist can request an ad lib¹⁶ adjustment to his current torque requisition.

Development of an emulator of the cyclist behavior.

Use the gathered data provided on the sampling to create an emulator to be used as a reference to train the RL agent. This module is a necessity due to the lack of additional test subjects as a consequence of the SARS-CoV-2 (COVID-19) pandemic.

Development of SW module to monitor and provide specific feedback to the assist pedaling unit.

This module will come into action when starting a trip in the vehicle or emulator. It will be in charge of monitoring in real-time (a 1-second response) the physical conditions of the cyclist and provide the necessary feedback to the pedaling assistance control module.

Development of SW GUI for RL agent training.

A GUI will be developed to unite all the different modules of the project in order to execute the proper training of the RL agent. This means this GUI will contain the emulator/USB entry, the RL module, and the reward module integrated.

¹⁶ As much and as often as required.

II STATE OF THE ART

II STATE OF THE ART

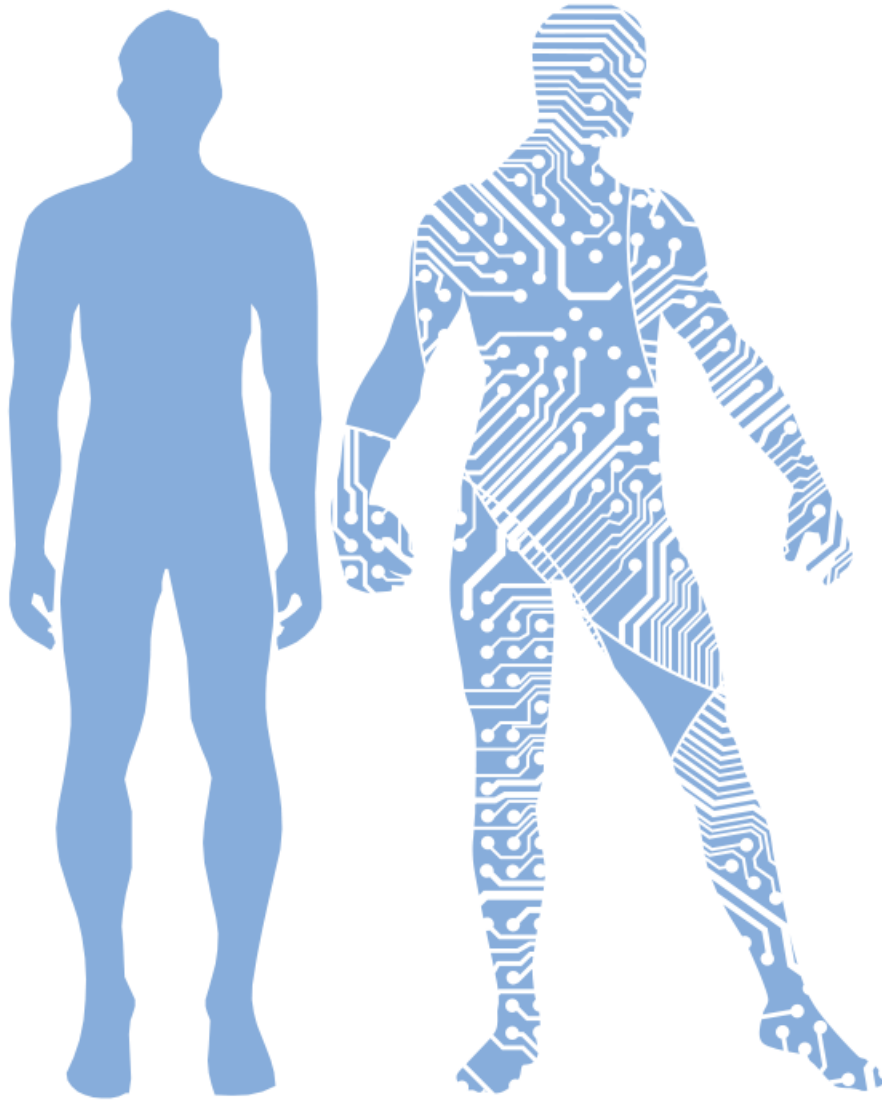


Image provided by <http://www.free-powerpoint-templates-design.com/>.

II STATE OF THE ART

The study on the field of AI has grown quite significantly, to the point that it surpassed the investigations in Computer Science (CC) in 2018 according to the SCOPUS, their DB shows an increase in 700% of AI articles since 1996, while CC has had only 500% in the same period of time (Shoham and et al. 2018). These data give us a clear idea of the immense extension and diversity of applications that are registered annually in the AI area.

Given the vast research diversity in AI, there is too many states of the art fronts to engage. Therefore, it was determined to present a selection of articles carefully selected to act as backbone¹⁷ of the ML methods and HW (Particularly the gadget¹⁸ category), and their successfully tested reliability.

This approach's main idea is to provide confidence in the selected HW for this project through positive results provided by internationally published work on the AI area.

2.1 Physiological Monitoring

The technology development on physical monitoring has been an active niche since the inception of basic electronics, like the Electrocardiogram machine invented by Augustus Waller, that used a pair of basic electrodes strapped in the front and the back of the chest and connected to a Lippmann's capillary electrometer was able to obtain a reading of the human heartbeat (Waller, 1887) shown in figure 2.1, to the current wearable technology (like smartwatches) that can provide real-time readings of several physical indicators like heartbeat, blood pressure, pedometer, among others.

¹⁷ A term used to denote the base or the most solid part of something.

¹⁸ A small mechanical or electronic device or tool, especially an ingenious or novel one.

II STATE OF THE ART

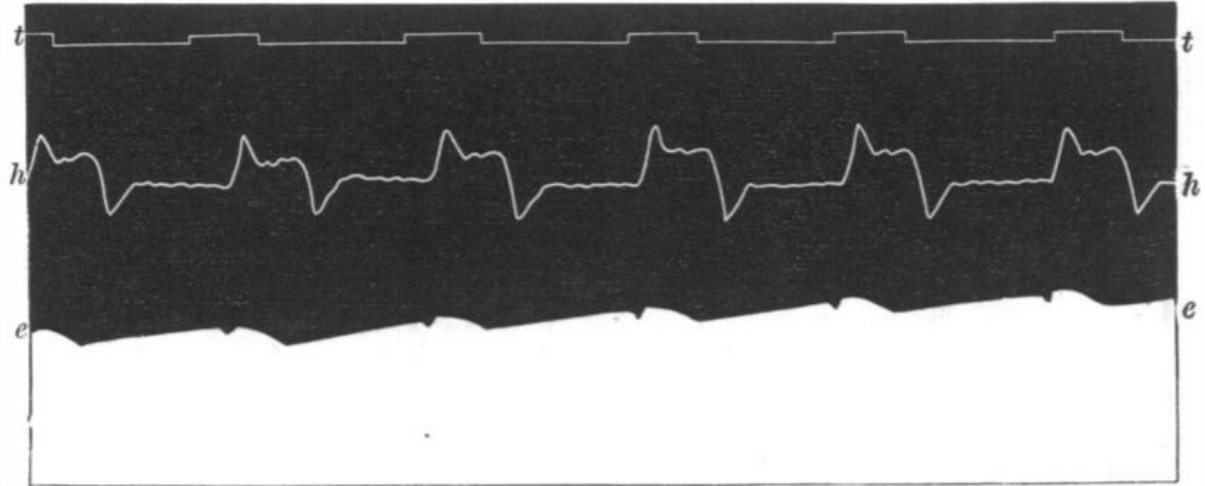


Figure 2.1. Electrocardiogram in 1887, man heart led off to electrometer from front and back of chest, *e* electrometer, *h* cardiograph and *t* time in seconds (Waller, 1887).

These advancements, the low-cost and availability on the market have opened the doors to the research of human physical activities on the fields of AI with great success. One of the latest cases include the research on emerging wearable Physiological monitoring technologies (Beltrame et al., 2018), where they established the relation between different human body systems: cardiovascular, respiratory, and muscular systems (Beltrame and Hughson., 2017) and the consequent prediction of human oxygen intake dynamics ($\dot{V}O_2$) during unsupervised activities of daily living (μ ADL) based on a random forest regression model (Breiman, 2001).

Their research HW included a hip accelerometer, three-lead ECG electrodes, and two respiration bands integrated into a smart shirt (Hexoskin; Carré Technologies, Montréal, QC, Canada) and those suffice to achieve great results (figure 2.2) concluding that aerobic system dynamics could be successfully mined from wearable sensors that provide continuous data for physical activity, breathing, and heart rate during unsupervised activities of daily living.

II STATE OF THE ART

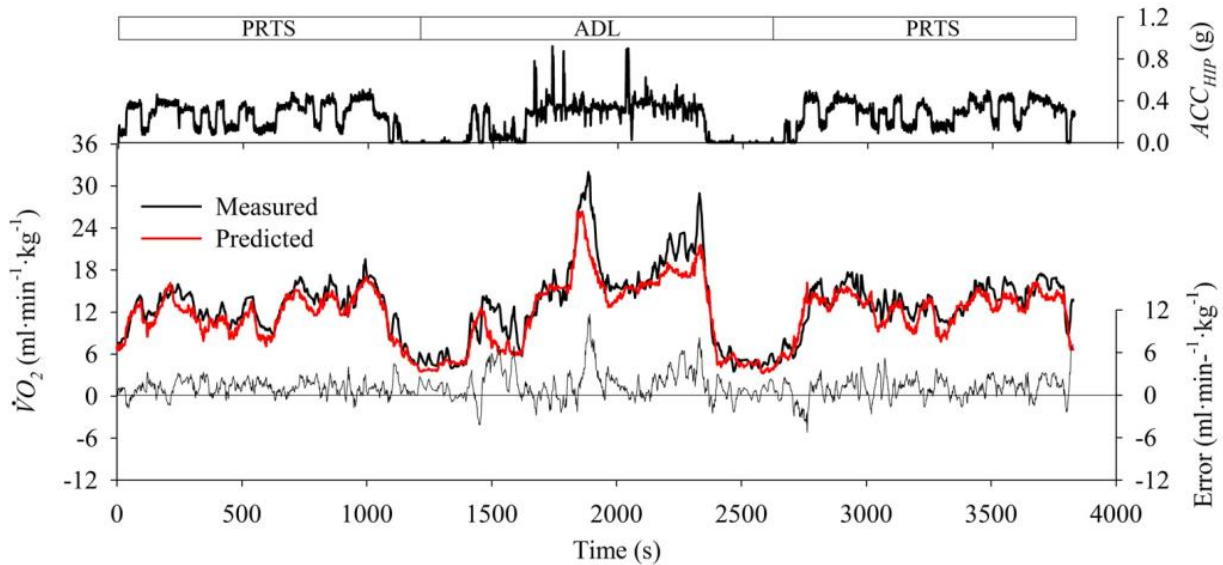


Figure 2.2. Hip acceleration (ACC_{HIP}) and measured and predicted oxygen uptake ($\dot{V}O_2$) response of representative participants during two pseudorandom ternary sequence protocols (PRTS) separated by simulated activities of daily living (ADL). The error of estimation shows a small positive bias when predicted $\dot{V}O_2$ for this participant was derived on the basis of the ensemble averaged predictor algorithm (Beltrame et al., 2018).

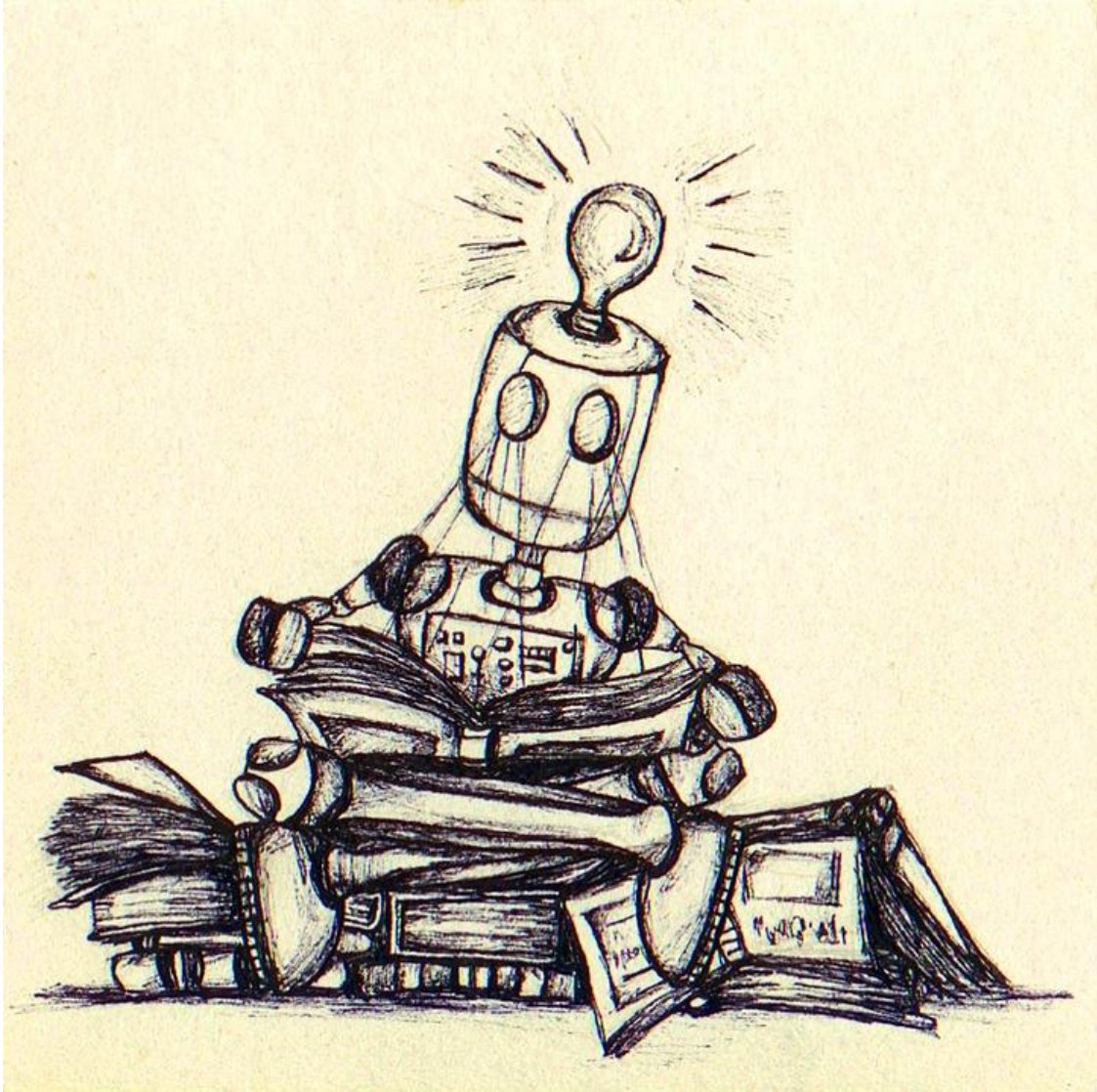
There is also the Dynamic time warping and machine learning for signal quality assessment of pulsatile signals research (Li, Clifford. 2012), where the researchers successfully use a ML technique (multilayer perceptron MLP Neural Network) to obtain a 95% accuracy detecting heartbeat pulses through a Photoplethysmography (PPG) sensor readings on real-time. The MLP proved to be superior to a static Signal Quality Index (qSQI), as shown in table no 2. These results reassure the reliability of the wearable PPG technology for complex ML research.

Table 2.1 Performances of heuristic and ML approaches. Acronyms: Accuracy (Acc), sensitivity (Se), specificity (SP), and Positive predictivity (PPV) (Li, Clifford. 2012).

Method	# Of Inputs	Training Performance (%)				Test Performance (%)				Notes
		Acc	Se	SP	PPV	Acc	Se	SP	PPV	
qSQI	1	88.1	88.3	87.4	95.9	91.8	94.7	80.6	95.0	qSQI _m =0.36
MLP	6	97.5	98.4	94.5	98.4	95.2	99.0	80.6	95.2	Hidden Nodes: 10
MLP	4	97.1	98.6	92.1	97.7	92.4	96.7	75.7	93.9	Hidden Nodes: 10

III LITERATURE REVIEW

III LITERATURE REVIEW



3.1 What is RL?

“Reinforcement learning is learning what to do ‘how to map situations to actions’ so as to maximize a numerical reward signal (Sutton and Barto, 2018, p. 1)”. this is the initial concept usually found when searching for the meaning of RL and, although this definition is concise, it does not fully detail what is behind those words, situations? Actions? Reward signal? What does it mean?

It is usual to think that all IA concepts are highly complex, but those usually are just simplified representations of natural phenomena that everyone experiences through daily activities, so, what is RL? In an expanded answer, we can define it as “a situation where an intelligent being is confronted with a problem it has never experienced before, the being observes what is the relation of the environment with the problem, analyze all actions that are available at the moment, perform one of the actions, evaluate the consequences of taking such action and finally, remember it for a future recurrence.” It is important to note a key factor for this concept relies on a trial-and-error and a posterior reward, ergo, it is what discerns the RL from other AI technologies.

We can review easy examples, like teaching a dog a new trick. There is no conventional way to let the dog know what the trainer wants him to do. All that is available is the interaction with the dog, use of command words, visual signs, and of course, many treats. Through the means of repetition of the commands, visual signs, and the occasional treat, the dog will try to take some action and wait if it rewards it with the delicious premium; this will go on to the point where the dog will recognize the commands and signs and will perform the action it is more likely going to result into the desired price.

Nevertheless, RL is not limited to dogs, Humans face the same situations throughout their lifetime. For example, every time someone is learning how to drive the RL is present, even though the learner has access to an extensive compendium of literature, it cannot simply use it

III LITERATURE REVIEW

to substitute the required mechanical muscle memory which only experience can provide, that first time when the engine is engaged, and the gas pedal is pressed a complex environment taking place, an intelligent being is taking a trial-and-error action by applying a certain amount of force into the pedal and perceiving if it was strong enough or way too much though the motion produced.

Although the concept and its rationale seem straightforward, there are some undesired trade-offs on the RL. The most prominent is the exploration-exploitation dilemma (Sutton and Barto, 2018, p. 3), a peculiar effect related to the inability to be entirely certain about getting the best reward given the action taken. If an intelligent entity took action given a specific environment and received some good reward, does it mean it is the best action? Should this action be taken forever on? Actions never taken before could bear greater rewards? Is the known action always good?

The greediness represents RL's exploitation facet, which means choosing an action known to provide a good reward and keeping on to it. Meanwhile, the exploration is the willingness to choose a different action at a potential loss, but simultaneously, a potentially higher reward, a gamble of sorts.

Examples of the dilemma can be found on daily basis decisions, for example, a person goes to a new restaurant to eat at lunchtime, the first two times it visited the place his menu pics were not particularly good or bad, but the third time he loved the pasta, so it becomes the default choice for the following three times. So far, this covers the principles of RL, trial-and-error until something good happened. Now, let us develop the situation in some interesting subsequent possibilities:

Picking always the pasta.

This makes exploitation (greedy) scenario, take an action known to provide a good reward. As in most real-life situations, a good reward cannot be taken for granted. There is always an unknown probability that, actually, the pasta might not be as good as expected. Why? For

III LITERATURE REVIEW

many reasons: Different chef, ingredients not fresh, salt mistaken for sugar, change of ingredients provider, change on the recipe, etc. This unexpected situation leads to take a different stance on greediness, the action of selecting the pasta is no longer the best action, now, is the action “with the higher probability” of a good reward, how good is the probability? It will be most likely unknown for the client since what happens in the kitchen is far from his reach. Now a new element has to be taking into consideration at the time of choosing the meal.

Picking the pasta only three times a week and the rest of the days try something new.

This approach covers the exploration facet of RL by giving a chance to take different actions $\frac{4}{7}$ of times a week. This approach will ensure that over time better discriminators are present when choosing the plate of the day.

Analyzing this approach further, some other factors can be taken into consideration. First, the fact that we choose a particular dish on a specific day and it provided a good/bad reward doesn't necessary means that it is a good/lousy dish right away, as reviewed before, it might have been the case of a "bad day" for that dish (as with the pasta). This opens yet another variable. Are certain dishes better on certain days?

The random exploitation-exploration lunch game

To fully explore the menu, one approach is to play a little game every day at lunchtime. The game consists of randomly picking the plate with specific rules presented over the following lines. Since the pasta is so far the favorite plate, it will have a $\frac{4}{7}$ chance, and the other $\frac{3}{7}$ will be a complete random plate to keep things interesting (sort of a gambling game). After a couple of weeks of playing the game, new information becomes available; for some reason, on Sundays, the sushi is delicious (behind scenes and unknown to the client, the Sunday is the fish resupply day, the rest of the week is frozen fish), so for Sundays, the rules suddenly change! The chances are rearranged as follows: $\frac{3}{7}$ times pasta, $\frac{2}{7}$ sushi, and $\frac{2}{7}$ a random choice.

III LITERATURE REVIEW

Thanks to the game and some luck (a Sunday with a random pick where sushi was selected), this phenomenon was discovered. This example reassures the importance of exploration: finding better rewards over time.

As the weeks pass, surprising discoveries are made: taco Tuesday, Friday pizza extravaganza, Monday charbroiled hamburgers, seasonal dishes, etc. Each discovery made, will move the chances of selecting a dish for that day, it might even surpass the pasta as a favorite!

This game of random dishes provides excellent insight into all the dishes available and, most importantly, the dish with the most probability of satisfaction for the day.

Finishing the game

There will be a point along the way where all dishes have been properly evaluated, and there will be not that much of a need to keep on randomly select the day's lunch. If the lasagna has been tested four times and never improves, there is no much sense in keep picking it just because of a bad roll of luck. However, stop the game would be wise in the long term? Probably not, since the recipes might change at some point, or a more skilled chef might take over that particular dish, and there is also the possibility of new menu items along the way.

Several methodologies exist to solve this issue, and a couple of examples are: reducing the changes to pick a random dish over time but never reach zero chances (Keep the adventurous spirit), another one is taking into consideration which dish has not been picked over a long time while choosing a random dish and see if something has changed since the last time. RL can be a never-ending continuous task since the environment and actions are subject to change.

Bootstrapping restaurants

On a final note, RL is known for having the remarkable ability to bootstrap experience. At any given moment, there could be a desire to visit a different restaurant. In that case, there is no

III LITERATURE REVIEW

need to start the game of exploitation-exploration from scratch. There is already some experience on several dishes that might be on both restaurant menus, so it is only a matter of adding the new restaurant to the game and adjusting the current experience further. There is always the possibility that it is better to visit a particular restaurant on a specific day of the week to eat a dish that tastes better or even a different dish that is not found at another location.

3.2 Elements of RL

RL elements are the standard way to define and provide the solid framework that is required to implement this methodology. Each component covers a particular function required to implement the mathematical backbone that will sustain the principles that the RL engulfs. This backbone will serve as a pattern to translate the RL methodology into a computer programming language.

Model

The model represents the world where the RL is going to be implemented and operated. The model can be taken directly from the natural world environment or imitate, as much as possible, such environment behavior by considering the key factors that are immersed in the problem at hand. Models are used for planning (Sutton and Barto, 2018, p.7), and since the primary and ultimate goal of an AI entity is to take decisions, the model provides the field and rules where such decisions are to be taken.

Types of RL environment

- *Deterministic*. In this type of environment, the outcome of an action taken (the decision) is known right away. For example, in chess, all pieces have specific rules for movement, and the result of the movement is known immediately.
- *Stochastic*. In this environment, there is no certainty of the results of making a decision. Here probability is the central ruler. For example, in a coin toss, it does not matter which side is the coin at the moment of the toss (supposing that there is no foul

III LITERATURE REVIEW

play taking place). The chances are 50/50, and in order to know the outcome, a little bit of time must pass for the coin to fall and stop.

- *Fully observable.* As the name suggests, this is an environment that is completely visible at all times. For example, in chess, all the pieces' positions are known before and after taking action.
- *Partially observable.* In this environment, only a certain amount of information is known at any given moment. For example, the game of Black Jack, the dealer and the player know their respective hand, except for one card.
- *Discrete.* An environment where the number of actions available is limited. For example, chess has a finite number of moves until the game is finished. Also, in a game of throwing darts, darts are given a small number, and once used, the game finishes.
- *Continuous.* An environment that never reaches an end. For example, heat control in a building, this task will never set on a specific amount since the natural weather, the number of people on the building at any given time never allows for a terminal optimal heat control amount.

Agent

The agent is the intelligent entity that is making decisions after evaluating the environment. In this particular case, we can define this entity as an artificial one, and this means it is an abstract conception embedded in a computer programming language that is executed in a computer itself.

An important note is that the “intelligence” that is being alluded to this artificial entity is due to the ability to choose actions. In RL, this entity is given no instruction on what to do, which is a critical difference from other AI methodologies. The learning is given in an area between what can be considered supervised and unsupervised learning.

III LITERATURE REVIEW

In supervised learning, all problems presented have the inputs and expected results clearly labeled, so the entities such perception can learn knowing exactly how much of an error needs to be adjusted when taking a bad decision at all times. This approach is quite helpful to achieve concrete goals, but at the same time, it might be its core weakness. An entity of this kind can achieve a single goal at the time; if the goal changes, retraining is mandatory.

On the other hand, unsupervised training AI ML is given absolutely no feedback, this kind of AI is intended to search patterns. A tremendous amount of data is fed to this entity, and by itself, it recognizes patterns that might be peculiar or previously unknown. These entities are considered to be more powerful than supervised ones, yet, it lacks a proper final goal. Therefore, it is hard to evaluate how the entity evolves or possibly gets stuck since it depends on the given data.

RL does not provide any insight into the model, yet it rewards the agent actions, which can be considered a very small labeled data, yet it is so little information that it cannot be considered supervised. Also, RL has a specific goal in its conception. This allows the evaluation of performance in a very straightforward manner. These attributes of RL allow it to take the best of both worlds (supervised and unsupervised), supporting it to a significant span of applications in the diverse areas of engineering.

Policy

A policy (usually denoted by the symbol π) can be interpreted as the “experience” of the entity “a policy is a mapping from perceived states of the environment to actions to be taken when in those states.” (Sutton and Barto, 2018, p.6). At first, a policy might be an empty table of the states and actions to take or filled with random actions since there is no knowledge at all, and as a constraint of the RL, an action must be taken.

As decisions are taken over time, the ones with the better results are saved on the policy table, thus, providing memory and better decision-making capabilities in repeated situations due to past rewards.

III LITERATURE REVIEW

Reward Signal

As stated before, RL is goal-oriented. Therefore, the Reward Signal comes as the point of reference of how good or bad is an action taken on each state. This reward is the primary motivator for improvement since RL will always be looking for the best possible reward in the long run. This reward usually comes in the form of a numeric value, which can be negative/positive or solely positive (where bad actions provide no reward) and is provided by the environment (the model representing the problem).

Value Function

A Reward Signal provides us (generally speaking) a fast value return for the action taken. On the other hand, we have a Value Function that describes “how good it is to be in some particular state”. Even though the concept might be similar to the Reward Signal, there could be a humongous amount of difference between their values, for example, if a person is sitting in a hot frying pan, taking the action of moving away from it will result in a high Signal Reward, yet, the fact of being sitting on a frying pan in the first is going to be particularly low, so much, that it is simple, not worth even considering the high reward from moving away from it, since, well, it will not make up for the loss.

Another example is getting ice cream, there could be an ice cream shop right away and another crossing the park. The catch is that the favorite flavor is on the ice cream shop passing the park. In this case, moving from the first ice cream shop to the park would yield a low or even null Reward Signal, yet, the value function will be good since it places us next to the desired destination. Hence, it is a place we would like to move on to subsequent trials.

Taken both examples into consideration states that brings the entity closer to the goal will have high value, even if the Reward Signal is not really noteworthy.

3.3 RL General Model

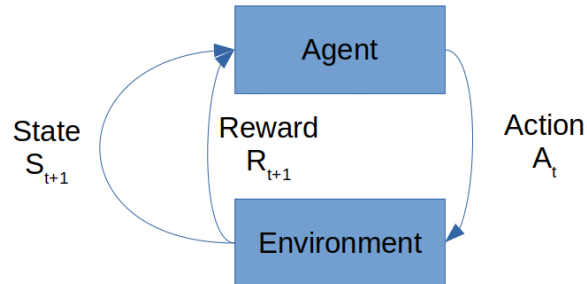


Figure 3.1. RL general model.

As a summary, the RL model (figure No. 3.1) is a cycle between the Agent taking the action that is most likely to provide the greater Reward, followed by the Environment moving the Agent to the next state and providing a Reward for the action taking and finally the Agent updating the Policy (experience) with the provided Reward.

3.4 The Multi-armed Bandit

The “multi-armed bandit”¹⁹, sometimes referred to as the k-armed bandit, is a fundamental learning problem closely related to RL, a precursor of sorts. This problem has a simple premise where an entity is presented with a k number of actions to choose. Each time an action is selected, a reward will be provided. This reward will be taken from a stationary probability distribution that depends on the action selected by the entity (Sutton and Barto, 2018, p. 25). The ultimate goal for the entity is to maximize the expected total reward over a period of time or number of trials (known as steps).

In other words, this problem can be seen as a sort of slot machine with several levers to pick from, one at the time (hence the multi-armed part of the name). Each lever will have a probability to provide or not a reward and how good it will be, as in a typical slot machine. The goal for the player is to accumulate the maximum amount of money (reward) from the machine.

¹⁹ Name related to the slot gambling machines colloquially referred to as “one-armed bandits”.

III LITERATURE REVIEW

Since the player does not know the probabilities of succeeding for each lever, trial-and-error will be the approach to solve the problem, and of course, a bit of memory. The player must remember how many times each lever has been pulled and how many times it provided earnings and how much profit it was each time. This detail is rather crucial since it is feasible for one lever to provide gains often, yet at a meager yield, whereas another lever might prove to produce less succeeding chances, yet, the gains are pretty significant.

Formalizing the problem, each of the k actions has an expected (mean) reward (considering that only one arm can be pulled at the time). This expected reward will be referred to as a value for the action. The time step will be denoted as t so each action can be represented as A_t (action taken from the possible actions at the time t) and the reward given as R_t (Reward given at the time t). Having these variables in place, the value of an arbitrary action a , denoted $q^*(a)$ is the expected reward (Sutton and Barto, 2018, p. 25) given that action a was selected:

$$q_* \doteq \mathbb{E}[R_t | A_t = a] \quad (3.1)$$

The values for each action are unknown for the player (otherwise, this would be a trivial matter), yet, the player can have its own estimations based on the experience gained through time. Such estimations have already been established as the value of the action. Hence, this can be formally denoted as $Q_t(a)$ (the best-known action a , taken on time t). This problem aims to have the value of $Q_t(a)$ as close as possible to the real $q_*(a)$.

Assuming that the game is continued and all *action values* are kept accordingly, there will be a point where at least one of the action values is going to be, in some amount, more prominent than the others (taking into consideration that the probability distribution for each action is different, otherwise this will be a trivial exercise).

At this point, a previously established behavior comes back to be formally defined. If the player uses the knowledge at hand and starts to take only the action providing the higher value, this turns the player into a greedy one, and this behavior is defined as **exploitation**.

III LITERATURE REVIEW

The exploitation of an action can ensure some profit at the end of the trial, yet, there is a trait in the universe colloquially referred to as “luck,” and it comes in two flavors, bad and good, formally speaking, the probability of each action will always have a 5% chance to fall into an extreme of the distribution (2.5% at each side). This universal trait proves to be the critical flaw in the exploitation behavior since there is no certainty that the number of trial-and-error time steps provided the best $Q_t(a)$ at the moment the player got greedy. It is quite possible that in the initial trials, a strike of bad/good luck unfolds a poor $Q_t(a)$, and the final reward proves to be far from the maximum possible.

To mitigate this undesired effect, the player might choose to keep trying different actions sporadically, update the action values, and perhaps find an action that yields better results than the one initially taken. This behavior is defined as **exploration**, and the goal of this behavior is to try to improve the rewards in the long run.

Although the exploitation provides a certain amount of reassurance of profit, the exploration will come with costs, since each time a non-greedy action is taken, there is no way to know if the new action will yield a good reward, and if it does, there is also not know if such gain is equal or greater than the greedy one. Also, exploration is not immune to the phenomena of “luck”. There is always the chance that the player picked the best action early on, therefore, all steps invested in exploration would result in wasted profit.

Exploitation vs. exploration is an entire topic by itself. This research will provide the approach taken to handle this situation in later sections.

Action-value Methods

Formalizing the concept of the value for each action, there is a need to keep a record of all rewards for all actions taken over time on each step to have the estimations for each action. This process is known as the Action-Value methods. One of such methods is to simply maintain the mean reward for action as in equation (3.2):

III LITERATURE REVIEW

$$Q_t \doteq \frac{\text{Sum of all rewards when } a \text{ taken prior to } t}{\text{Number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}} \quad (3.2)$$

As stated before, $Q_t(a)$ represents the best-known action value defined by the sum of all the rewards divided by the number of times that action has been chosen. The $\mathbb{1}$ symbol works as an auxiliary denoting that only one action can be taken at the time. Hence, only one $Q_t(a)$ can be updated at the time. If $\mathbb{1}$ is true, then the action is updated. As a note, if the denominator is zero, $Q_t(a)$ is given a random number (although most literature uses a zero) and updates the number of times to 1 to avoid the intrinsic error.

Although this approach to maintain the action-value estimation is rather simplistic, it has the certainty of eventually achieving the convergence between $Q_t(a)$ and $q^*(a)$, this thanks to the law of large numbers, which states that as the denominator increase to infinity, it will eventually match the probability distribution.

Selecting the best action

Following the formalization of terms, the selection of the action will be presented on a formula. As before, a simple method will be followed, that is, from all actions available, choose the one with the highest value as shown in equation (3.3). If there is more than one action with the same high value, break the tie by choosing a random action from those on the dispute.

$$A_t \doteq \underset{a}{\operatorname{argmax}} Q_t(a) \quad (3.3)$$

This approach only covers the greedy behavior which is not something desired, especially at early stages. From now on, this dilemma will be handled by adding a simple rule, a small number (greater than 0 but smaller than 1) will be chosen, this number will be designated as epsilon (ϵ), each time an action is going to be taken a random number will be generated (ranging from 0 to 1), if the number is lower than ϵ , then a random action will be taken, else, the greedy action is going to be taken. This is a simplistic approach from the ϵ -greedy methods that are available to handle the dilemma of exploitation vs. exploration. It is far from being the

III LITERATURE REVIEW

best method, but it provides an initial way to handle the issue and experiment with the theory in the early stages.

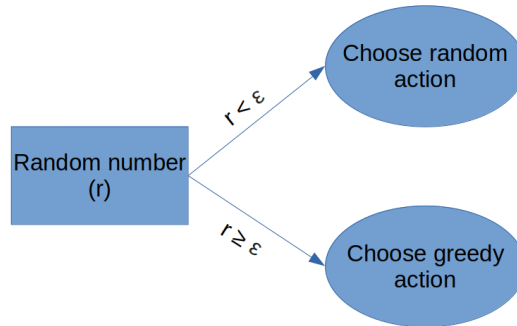


Figure 3.2. Basic epsilon-greedy method for exploration.

The 10-armed Testbed

To assess the early concepts, Sutton and Barto created a small test using a 10-arm bandit and using the greedy selector with ϵ -greedy exploration. Below is shown the distributions they provided for each arm.

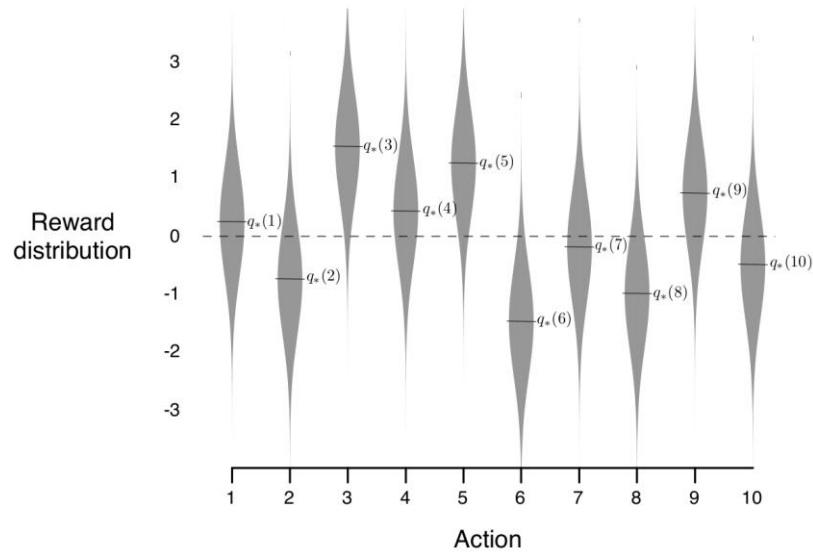


Figure 3.3. Sutton and Barto experiment for 10-armed bandit (Sutton and Barto, 2018, p. 28).

III LITERATURE REVIEW

Each arm uses a Gaussian distribution, as shown in the gray areas (figure 3.3). Also, the $q^*(a)$ is given for each action. The test consisted of 1000 steps using different ϵ values on each episode for the ϵ -greedy method ($\epsilon = 0$, $\epsilon = 0.1$ and $\epsilon = 0.1$). The results are shown below on the image 15:

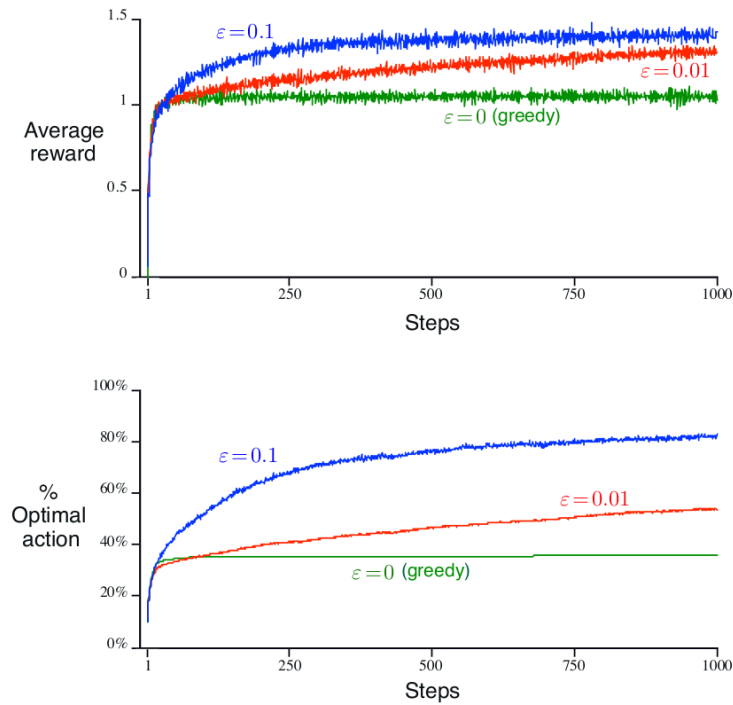


Figure 3.4. Average performance of 10-armed test bed. (Sutton and Barto, 2018, p. 29)

Analyzing the results, there are several exciting developments to review. Starting with $\epsilon = 0$ (this is to be considered a pure greedy approach), it can be appreciated that once an action value reached a superiority over the other actions, it got no more development (flat line) due it simply keeps using the same action in the rest of the time steps.

In the $\epsilon = 0.01$, there is a minimal gap where it underperformed the greedy approach, this is a result of the “cost” related to the exploration of bad actions, but it catches on to the greedy approach considerable fast (on less than 250 steps) and then it surpasses it and ends up with

III LITERATURE REVIEW

more prominent rewards at the end of the experiment, making up for the losses and even providing more gains at the long run.

Finally, the $\epsilon = 0.1$, which proved to be highly superior to the other approaches, not only behaves equally that the greedy method but right away surpasses it. Also, it is visible that between steps 500 and 750, it finds the closest $Q_t(a)$ to $q^*(a)$, flattening the reward from there onwards. This also gives us an insight into how far the greedy approach was to the actual optimal values, and the $\epsilon = 0.01$ was gradually reaching for it, yet, the time steps given proven not to be enough to reach the optional action-value.

3.5 The Markov Decision Process

The Markov Decision Process (MDP) is considered a step forward from the multi-armed bandit, since MDP mathematical methods cover the influence of current action selection on future states and actions, rather than simply take the reward of the current action (as in the multi-armed bandits), this will be a common scenario on non-trivial RL problems where the agent requires of a sequential amount of actions in order to achieve the desired goal. Most of the RL problems can be represented as a variant of the MDP due to the highly desired foresight attribute.

The Markov chain

The MDP requires some primary constraints to be covered to understand the later mathematical framework fully. The first one is the Markov Property that states that the future is only ruled by the present, disregarding the past completely. In other words, the current state and actions are the only ones taken into consideration. Although this might seem counterintuitive due to the evident deviance from traditional probability and statistics fields, it is important to remember the exploration and nonstationary problems common in real life. The past might manifest an undesired bias obscuring changes in the environment.

Although this approach does not fit every single problem, for example, while throwing a dice, the current shown number does not influence what number will be on the next throw.

III LITERATURE REVIEW

The Markov Chain is a probabilistic framework that strictly follows the Markov Property. Therefore, it is intended to predict the future states, not caring for past states. In order to explain how a Markov chain is developed, let us take a small example, with a parking lot. Let us suppose a parking lot on the busiest part of the city has three states: low occupancy (0 – 25% occupancy), medium occupancy (26 – 70% occupancy), and high occupancy (71 -100% occupancy). Most clients prefer a low occupancy due is easy to pick a parking slot and/or one that is closer to the pedestrian entry/exit.

Moving from one state to another (one hour to the next one) is defined as a transition, and one way to analyze the transition probabilities is to arrange them in a table, known as a Markov table:

Table 3.1 Markov Table.

Current Occupancy	Next Occupancy	Transition probability
Low	Low	0.2
Low	Medium	0.7
Low	High	0.1
Medium	Low	0.1
Medium	Medium	0.4
Medium	High	0.5
High	Low	0.1
High	Medium	0.1
High	High	0.8

There is also the option to create a Markov diagram to represent the transition probabilities (figure 3.5) from one state to another in a visual fashion:

III LITERATURE REVIEW

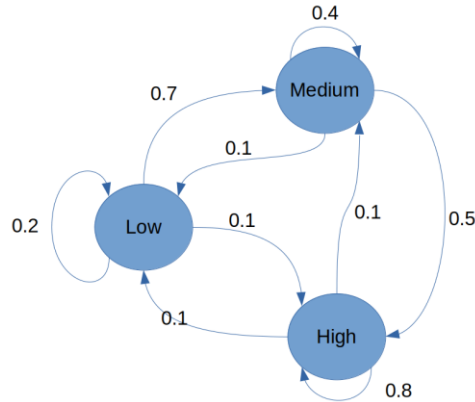


Figure 3.5. Markov transition probabilities diagram.

MDP

The MDP takes root on the Markov chain and expands its mathematical core to fully embrace the RL methodologies. “MDPs are meant to be a straightforward framing of the problem of learning from interaction to achieve a goal (Sutton and Barto, 2018, p. 47)”.

The MDP framework works on well-defined elements for the theoretical representation:

- **S.** All the states that the agent can be at any given time (depending on the problem) might be finite or infinite.
- **A.** All actions that an agent can take in order to move from one state to another.
- **t.** Current time step.
- **r.** Reward provided (also known as reward probability). Since the MDP has a time step behind trade-off, it is usually denoted as R_{t+1} .
- **$p(s', r|s, a)$.** MDP dynamics function p , is defined as the probability of moving from state s to s' and the reward r , given that action a was taken. Defined formally on equation 3.5

$$\circ \quad p(s', r | s, a) \doteq P_t\{S_t = s', R_t = r | S_{t-1} = s, A_{t-a} = a\} \quad (3.5)$$

The usual behavior form of an MDP comes in the following pattern:

III LITERATURE REVIEW

State – Action – Reward – State

This can be expanded using the time step t as an index:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \dots$$

The function p (the dynamics of the environment) can be used to obtain all information that might be required:

- **State-transition probabilities.** Defined as the probability of moving to state s' given that action a was taken on state s . It is important to denote that the function p (equation 3.6) is a three-argument function:

$$p(s'|s, a) \doteq \Pr\{S_t = s' \mid S_{t-1}, A_{t-1} = a\} \quad (3.6)$$

- **Expected reward with state-action tuple.** A two-argument function r (see equation 3.7) can be defined as the expected reward given the previous state s_{t-1} and the previous action taken a_{t-1} :

$$r(s, a) \doteq \mathbb{E}[R_t \mid S_{t-1}, A_{t-1} = a] \quad (3.7)$$

- Expected reward with state-action-next-state triplet. If required even a function r (See equation 3.8) with the three-argument can be obtained:

$$r(s, a, s') \doteq \mathbb{E}[R_t \mid S_{t-1}, A_{t-1} = a, S = s'] \quad (3.8)$$

Returns and episodes

As stated before, on an MDP the main goal is to maximize the total amount of rewards R instead of the immediate reward. The formula (equation 3.9) for such goal is rather simple:

$$R_t = r_{t+1} + r_{t+1} + r_{t+1} + \dots + r_T \quad (3.9)$$

It is important to note that r_t is not taken in the formula since the first reward is generated after the first action has been taken, so r_t would be simply nonexistent. Also, the last reward to be considered will be the last $t \in T$. A second important concept is the “backward” nature of the

III LITERATURE REVIEW

sum. It might not be apparent right away, yet, this formula works from T back to $t+1$. On the following lines, this concept will take a more intuitive meaning.

Episodic and continuous tasks

Episodic tasks, as the name implies, have a terminal state. Therefore, the RL agents are given an initial state and work their way to reach this terminal state. This kind of task ensures the certainty of a finite number of actions taken (proving that the task does have a real way from the initial state to the terminal state). Therefore, the R calculation becomes relatively straight forward as seen before.

On the other hand, continuous tasks do not have an implicit terminal state. These tasks are in a perpetual search of maximizing the return (For example, an autonomous lighting controller for a 24/7 building). All the theory covered so far fits this task, yet we encounter a problem when trying to calculate the return because there is no terminal t ; consequently, there will be a never-ending sum of rewards. This approach is not practical.

Discount factor

The discount factor is an additional scalar required for a continuous task. It provides a framework on how to bargain with the infinite nature of the rewards accumulated. It is represented by the lower-case gamma symbol (γ), and its principle is to create an increasing discount on future rewards (that is $t+1$ onwards).

The γ is a numeric parameter that can range between the values of $0 \leq \gamma \leq 1$. If $\gamma = 0$, then the agent is considered “myopic,” this means that the agent only cares for the immediate reward, and the future rewards are all but disregarded. On the other hand, if $\gamma = 1$ means that all rewards are equally important. The modification to the return formula is given in equation 3.10:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \gamma^4 r_{t+5} + \dots \quad (3.10)$$

III LITERATURE REVIEW

At first sight, this approach does not show how the γ helps on the infinite steps dilemma. It has to be developed further on:

First, by extracting the common γ from all rewards starting from γr_{t+2} in equation 3.10. This will change the equation as represented in 3.11

$$R_t = r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + \gamma^3 r_{t+5} + \dots) \quad (3.11)$$

Now, what is inside the parenthesis is just R_{t+1} (see equation 3.12), that is, all the sub-index numerators are simple moved one time step ahead.

$$R_{t+1} = r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + \gamma^3 r_{t+5} + \dots \quad (3.12)$$

Substituting 3.12 into 3.11 we finally get an equation (3.13) that removes the unknown number of future time steps.

$$R_t = r_{t+1} + \gamma(R_{t+1}) \quad (3.13)$$

The policy function

“Formally, a policy is a mapping from states to probabilities of selecting each possible action.” (Sutton and Barto, 2018, p. 58). Usually, a policy is denoted by the symbol of π (π), and it contains the probability of taking an action a when the agent is in the state s : $\pi(a|s)$. In other words, it is a set of preconceived instructions.

State value function

As the name states, this function specifies “how good” it is for an agent to be at a specific state given the policy π . It is defined at equation (3.14):

$$V^\pi = \mathbb{E}_\pi[R_t | s_t = s] \quad (3.14)$$

State-action value function (Q function)

The Q function evaluates how good is for an agent to take action a given that it is currently at state s , following the policy π . Although similar at the core, the state-value function and the Q

function provide two different values. The first gives the value of a state while the former the goodness of the actions on a state. The Q-function is defined on equation 3.15:

$$Q^\pi(s, a) = \mathbb{E}_\pi[R_t \mid s_t = s, a_t = a] \quad (3.15)$$

3.6 The Bellman equation and optimality

The Bellman²⁰ equation is a mathematical framework used universally to solve MDP, which means it is at the heart of the RL. To avoid confusion, the term “solve an MDP” refers to finding the optimal policy and value functions for a given MDP. Considering that there can be many different policies, it is also true that there can be different value functions for each policy.

For starters, the optimal value function (equation 3.16) denoted by $V_*(s)$ is the one that yields the maximum reward against all other value functions:

$$v_*(s) \doteq \max_\pi v_\pi(s) \quad (3.16)$$

The Bellman equation for a value function given policy π is defined on equation (3.17):

$$V_\pi(s) = \mathbb{E}_\pi[r_{t+1} + \gamma V_\pi(S_{t+1}) \mid S_t = s] \quad (3.17)$$

As well, optimal policy contains the optimal Q function denoted by q_* and defined on equation (3.18):

$$q_*(s, a) \doteq \max_\pi q_\pi(s, a) \quad (3.18)$$

Having both optimal value-state and Q functions, it is possible to rewrite q_* in terms of v_* as shown in equation (3.19):

$$q_*(s, a) = \mathbb{E}[r_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \quad (3.19)$$

²⁰ Named after the American mathematician, Richard Bellman

3.7 Dynamic programming

Dynamic programming (DP) is a technique used to solve complex problems, and to do so, it breaks down the problems into smaller ones, calculates and stores each individual solution; reducing the need of recompute on larger iterative problems.

This technique is ideal for solving to compute optimal policies given a perfect model of the environment as a Markov decision process (MDP) (Sutton and Barto, 2018, p. 73). In general, the DP will be used to systematically evaluate 2 algorithms: Value Iteration and Policy iteration.

3.6.1. Value Iteration

As the name states, the algorithm evaluates the value function of a state. Before starting the subject, it is assumed that the first value functions of a policy are entirely random (or all have a value of zero) due to the need for some value at the first run. Therefore, the chances of having the optimal value function will be rather low, and consequently, the need to iterate through the value functions and improve its value until the optimal value is found. Having the optimal value functions would allow for an optimal policy search relatively easy.

It is also required a stop signal, this signal (θ) comes in the form of a decimal value ($0 < \theta < 1$) that will serve as a reference for stopping the iteration, this means that eventually, the changes in the value function in t and $t-1$ will be particularly small, hence, no need to keep looking. Therefore, when the difference between $V(s)_t$ and $V(s)_{t-1}$ is smaller or equal than θ , the iteration will be stopped and take the latest value function as the best one. All this in order to save a good deal of computational time.

The final requirement is to implement the bellman equation (3.20) in order to obtain the maximum value function on each iteration:

$$v(s) = \max_a \mathbb{E}[[rR_{t+1} + \gamma v(S_{t+1}) | S_t = s, A_t = a] \quad (3.20)$$

III LITERATURE REVIEW

Value iteration algorithm

```
Initialize theta threshold
Initialize V (s), for all s ∈ S+, arbitrarily except V(terminal), it will have a value of 0

Repeat
  # Reset the maximum difference found between V(s)t and V(s)t-1
  maximum_difference = 0

  # iterate through all states
  For each state ( s ∈ S)
    # Get a copy of the current value function
    previous_value_function = V(s)

    # Apply Bellman equation to update the value function
    V(s) = maxaE[Rt+1 + γV(St+1) | St = s, At = a]

    # Calculate the difference between the updated value function and the previous value function
    current_difference = absolute( previous_value_function - V(s))

    # if the current difference is greater than the maximum difference
    if current_difference > maximum_difference
      # Update the maximum_difference
      maximum_difference = current_difference

# Repeat until the maximum difference is lower or equal to theta
Until (maximum_difference <= theta)
```

3.6.2. Policy Iteration

The second algorithm uses entire policies for evaluation and improvement and has a relatively straightforward principle, evaluates a given policy π_n , improves policy π_n , and moves to policy π_{n+1} . This is repeated until the optimal policy π^* is found.

Policy evaluation

The first step is to have a framework to evaluate the value function of a given policy π (sometimes referred to as policy prediction), which is just a rework of the already established algorithm (equation 3.21) where the policy π is used to obtain the action to take in the given state:

$$V_{\pi} \doteq \mathbb{E}_{\pi}[R_{t+1} + \gamma V_{\pi}(s_{t+1}) | S_t = s] \quad (3.21)$$

Iterative policy evaluation algorithm

```
get π, the policy to be evaluated
Initialize theta threshold
Initialize V (s), for all s ∈ S+, arbitrarily except V(terminal), it will have a value of 0
```

III LITERATURE REVIEW

```
Repeat
# Reset the maximum difference found between V(s)t and V(s)t-1
maximum_difference = 0

# iterate through all states
For each state ( s ∈ S)
# Get a copy of the current value function
previous_value_function = V(s)

# Apply Bellman equation to update the value function given policy π
Vπ(s) = Eπ[Rt+1 + γVπ(st+1) | St = s]

# Calculate the difference between the updated value function and the previous value function
current_difference = absolute(previous_value_function - V(s))

# if the current difference is greater than the maximum difference
if current_difference > maximum_difference
# Update the maximum_difference
maximum_difference = current_difference

# Repeat until the maximum difference is lower or equal to theta
Until (maximum_difference <= theta)
```

Policy improvement

Our reason for computing the value function for a policy is to help find better policies (Sutton and Barto, 2018, p. 76). Now that the value functions are known, there is the chance to determine if certain states should take an action that is not the one provided by the given policy π . This is where the Q function (AKA state-action) comes at hand (equation 3.21):

$$q_{\pi}(s, a) \doteq \mathbb{E}[R_{t+1} + \gamma V_{\pi}(s_{t+1}) | S_t = s, A_t = a] \quad (3.21)$$

The principle is simple: if $q_{\pi}(s, a)$ has a greater value than $V_{\pi}(s)$, we have found a better policy. Therefore, we have a $V_{\pi'}(s)$ greater than or equal to $V_{\pi}(s)$. To formally define this a greedy policy π' is defined on equation (3.22):

$$\begin{aligned} \pi' &\doteq \operatorname{argmax}_a q_{\pi}(s, a) \\ &= \operatorname{argmax}_a \mathbb{E}[R_{t+1} + \gamma V_{\pi}(s_{t+1}) | S_t = s, A_t = a] \end{aligned} \quad (3.22)$$

III LITERATURE REVIEW

Policy iteration algorithm

Now that both evaluation and improvement are defined, it is possible to create an iterative algorithm. As reviewed before, slicing the evaluation and improvement into minor problems and solve them in an iterative and sequential procedure will dramatically decrease the computational power required. At the end of each iteration, there will be a policy both evaluated and improved:

```
# Initialization
# create a value function that belongs to the real numbers and create the policy  $\pi$  for all states taking the actions from
# the Action set arbitrary for all states
 $V(s) \in \mathbb{R}$ 
 $\pi(s) \in A(s)$  for all  $s \in S$ 

get  $\pi$ , the policy to be evaluated
Initialize theta threshold

# Policy improvement iterator
Repeat
  # Policy evaluation iterator
  Repeat
    # Policy evaluation
    # Reset the maximum difference found between  $V(s)_t$  and  $V(s)_{t-1}$ 
    maximum_difference = 0

    # iterate through all states
    For each state ( $s \in S$ )
      # Get a copy of the current value function
      previous_value_function =  $V(s)$ 

      # Apply Bellman equation to update the value function given policy  $\pi$ 
       $V_{\pi}(s) = E_{\pi}[R_{t+1} + \gamma V_{\pi}(S_{t+1}) | S_t = s]$ 

      # Calculate the difference between the updated value function and the previous value function
      current_difference = absolute( previous_value_function -  $V(s)$ )

      # if the current difference is greater than the maximum difference
      if current_difference > maximum_difference
        # Update the maximum_difference
        maximum_difference = current_difference

    # Repeat until the maximum difference is lower or equal to theta
    Until (maximum_difference <= theta)

  # Policy improvement
  # Define a stop trigger to know when to break the loop
  policy_stable = TRUE

  # Iterate on all states
  For each state ( $s \in S$ )
    # make a copy of the current policy action-value
    current_action =  $\pi(s)$ 
```

III LITERATURE REVIEW

```
# Update the policy using the greedy  $\pi'$  derived from the Q function
 $\pi(s) = \operatorname{argmax}_a E[R_{t+1} + \gamma V_{\pi}(s_{t+1}) \mid S_t = s, A_t = a]$ 

# If the current action is not equal to the updated action
if (current_action !=  $\pi(s)$ )
    # The policy change, therefore, it is not stable yet, another loop will be required until no
    # change is made
    policy_stable = False

Until (policy_stable == TRUE)
```

3.2.3 Generalized Policy Iteration (GPI)

Both value and policy iterations provide a methodical means to approach the optimal policy, yet, both algorithms have an inherent disadvantage: they require complete sweeps before an improvement can be made. This undesired feature carries a heavy burden when the state-action tables are considerably long.

Hence the necessity of enhancing those algorithms, and such enhancement comes into the form of a GPI framework. This notion applies to all methods that implement the general idea of allowing the policy evaluation and the policy improvement to interact “independent of the granularity and other details of the two processes (Sutton and Barto, 2018, p. 86)”.

This framework is at the heart of RL theory, the endless search for policy improvement and updates of the known experience. This process is described by Sutton and Barto on the diagram shown in figure 3.7:

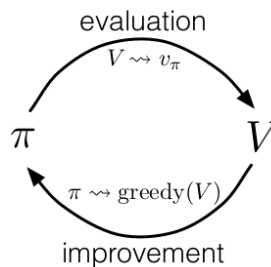


Figure 3.6. GPI interaction diagram.

If the Bellman equation is applied both, at the evaluation and the improvement, then optimality is granted. This process will be repeated until the changes on the evaluation and improvement values stop changing (also known as stabilization).

3.7 Monte Carlo Methods

So far, all algorithms and methods covered have considered a complete knowledge of the environment (rewards and action probabilities), yet this assumption is rarely found in real life other than trivial matters.

For example, calculating the probabilities of getting a specific number on a regular dice, will be $1/6$, and that is pretty much what there is to it. If a person is requested to take the action of throwing a specific number, the odds will remain (unless foul play is at hand).

On the other hand, a baseball coach can instruct the next man at the bat to perform a home run, much needed to win the game, yet, the probabilities of such action are pretty hard to be known beforehand. The environment has so many entries like the player accuracy, exhaustion, the direction and speed of the wind, the pitcher ability, the mood of both players, even the exact shape and wear of the ball could be a critical condition.

These (along with many other) environmental values will make a notable arduous task to provide the exact probability of the requested action, much to the dismay of the coach and the team fans (even more to the poor soul requested such feat).

Event at such unfavorable conditions RL methods still hold true, the trick is found in what we could consider experience. “Although a model is required, the model need only generate sample transitions, not the complete probability distributions of all possible transitions that is required for dynamic programming (DP) (Sutton and Barto, 2018, p. 91)”.

Monte Carlo methods are changes in our approach to solving the Bellman equation that only require return samples from the environment: take actions and see what happens until the task at hand is completed. This implies a constraint; even if the environment is entirely unknown, it is essential to have a well-defined and reliable return value.

Monte Carlo (MC) methods are changes in our approach to solving the Bellman equation that only require return samples from the environment: take an action and see what happens. MC relies on averaging the sampled returns. Therefore, some constraints are implicit: The return

III LITERATURE REVIEW

value must be well-defined and reliable, the exercise of episodes to acquire the experience, and finally, all episodes must lead to an eventual termination (a requirement to traceback the returns). “Monte Carlo methods can thus be incremental in an episode-by-episode sense, but not in a step-by-step (online) sense (Sutton and Barto, 2018, p. 91)”.

3.8 Monte Carlo Evaluation (Prediction)

As stated before, Monte Carlo methods rely on averaging the state value function, and this means that in due time the state value function for each state will approach the actual value. This concept bootstraps the ideas already shown where the discount factor γ provides a way to trace back all future rewards to any initial state).

The averaging facet of the Monte Carlo method brings a special consideration: If a policy requires to pass through a particular state more than once. Is the state updated every time? Or only once for the episode?

Both ways have a granted convergence with the actual probability distributions, so there is no real drawback or advantage attributed to any of the two methods. This document will refer as every visit update for the research at hand.

Monte Carlo Evaluation Algorithm

The algorithm is not so different than the previously described, the only real change is the averaging of the state values.

```
# Get the policy to evaluate
get policy_pi

# Initialize
# Set all state function values an arbitrary real number for all states
V(s) = random(R) for all s ∈ S

# Create an empty list of total rewards per state
returns[s] = nil for all s ∈ S

# Get the number of episodes
get no_episodes

# Iterate trough episode-by-episode
for (no_episodes)
  # Generate an episode following policy_pi: S0, A0, R1, S1, A1, R2, ..., ST-1, AT-1, RT
  episode append (policy_pi)
```

III LITERATURE REVIEW

```
# Reset the final Reward value
final_reward = 0

# Loop through the last time step T to initial t = 0
for (step = T down to step = 0)
  # Calculate current Reward for step = t
  final_reward =  $\gamma$ (final_reward) + Rstep+1

  # Append or update the state Sstep with the latest value
  # Note. Logic is part of the every visit update rule
  returns[Sstep].append = final_reward

# Finally average the rewards for the Sstep
V(Sstep) = average(returns[Sstep])
```

3.8 Monte Carlo Optimization (Control)

Monte Carlo optimization has only a minor difference compared with previous GPI algorithms due to an issue related to the policy input requirement. Since the transition probabilities for every state-action are unknown, the policy becomes the de facto choice, disabling the ability to implement any exploration as it is.

The first approach to handle this situation is to implement the concept of **exploring starts**: The episodes start in a state-action pair, and every pair has a nonzero probability of being selected as the start. (Sutton and Barto, 2018, p. 96).

This model grants that at least all actions have a chance to be selected, yet, this approach is relatively poor since there is always the bad luck phenomena lurking on every action-value improvement that could deter or eliminate future chances of being selected, sealing that specific door into oblivion. For now, this approach will be used in order to create an algorithm. Further on, in this dissertation, better models will be introduced to handle this issue.

Monte Carlo Optimization algorithm

```
# Initialize
# Create a random policy  $\pi$  for all states
policy_pi(s) random a  $\in$  A(s) for all s  $\in$  S

# Create random state-action values for all state-actions combinations
Q(s,a) random  $\in$  R for all s  $\in$  S and a  $\in$  A

# Create an empty list for the Reward calculation using a tuple index using all s  $\in$  S and a  $\in$  A
returns[s,a] = nil for all s  $\in$  S and a  $\in$  A
```


III LITERATURE REVIEW

```
# Get the number of episodes
get no_episodes

# Iterate through episode-by-episode
for (no_episodes)
  # Select  $S_0 \in S, A_0 \in A$  randomly in such way that all pairs have a probability  $> 0$ 
   $S_0 = \text{random } s \in S$ 
   $A_0 = \text{random } a \in A$ 

  # Generate an episode that starts with A and S then follow the policy  $\pi: S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$ 
  episode append ( $S_0, A_0$ )
  episode append (policy_pit+1 - policy_piT)

  # Reset the final Reward value
  final_reward = 0

  # Loop through the last time step T to initial t = 0
  for (step = T down to step = 0)

    # Calculate current Reward for step = t
    final_reward =  $\gamma(\text{final\_reward}) + R_{\text{step}+1}$ 

    # Append or update the state  $S_{\text{step}}$  with the latest value
    # Note. Logic is part of the every visit update rule
    returns[ $S_{\text{step}}, A_{\text{step}}$ ].append = final_reward

    # average the rewards for the  $S_{\text{step}}$  and action  $A_{\text{step}}$ 
     $Q(S_{\text{step}}, A_{\text{step}}) = \text{average}(\text{returns}[S_{\text{step}}])$ 

    # Finally choose the greedy action-value
    policy_pi( $S_{\text{step}}$ ) =  $\text{argmax}_a Q(S_{\text{step}}, a)$ 
```

3.9 Temporal Difference Learning

Temporal Difference (TD) methods are a combination of the DP and Monte Carlo Methods, and this means that TD is also able to learn from experience (averaged rewards), but it has a hugely desired feature, TD can learn while iterating through from state to state, it does not need for the episode to finish.

This feature is extremely useful in continuous environments where there is not a proper ending state (a requirement of all methods so far). As all methods reviewed beforehand, this is an implementation of the Bellman equation with algorithms that sustain its optimality over time.

3.9.1 TD Evaluation (prediction)

Since Monte Carlo waits for the end state to be reached to average the rewards and backtrace all states, updating the value functions and Q values, all required data is at hand. In TD there is only the knowledge of the states that have been selected; therefore, one way to deal with this

III LITERATURE REVIEW

temporal data is to calculate the average reward on the fly, that is, consider the value of each state and increase or decrease its value right after the action reward is known.

The first approach to this method is called TD(0) where the zero means that each state will take in consideration the immediate prior reward for calculation. It is also possible to extend the propagation of the reward to more previous states by simply applying the γ but this dissertation is limited to the TD(0).

The Bellman equation requires a couple of modifications to take into consideration the current value function while calculating the new value as shown in equation 3.23:

$$V(S_t) = V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \quad (3.23)$$

The main idea is to take the current value and add or subtract a small value given the reward provided. In order to do so, the value of $V(S_t)$ is included in the calculation, and also, a new concept in the form of a step discount alpha (α), which represents how much each step can influence the current value function. This α is very similar to the discount factor γ used to determine the influence of future rewards in the current state value. Finally, the original amount is subtracted from the reward section. This is a way to control the *TD error*, an inherent value given the fact that the final reward is being “estimated.”, and therefore, prone to a certain amount of error.

TD Evaluation Algorithm

```
# Get the policy  $\pi$  to be evaluated
get policy_pi[]

# Get the step discount  $0 < \alpha \leq 1$ 
get alpha

# Get the factor discount gamma
get gamma

# Initialize  $V(s)$   $s \in S^+$  for all states except the terminal state which is assigned a zero value-state
V(s) = random
V(sT) = 0

# Get the number of episodes
get no_episodes
```

III LITERATURE REVIEW

```
# Get the number of steps
get no_steps

for each (no_episodes)
  #Initialize S
  initialize current_state
  for each (no_steps)
    # Take the action from the policy
    action = get_policy_pi(current_state)

    # Take action A and observe R, S'
    reward, next_state = environment.apply(action)

    # Apply Bellman equation
    V(current_state) = V(current_state) + alpha[reward + gamma(V(next_state)) - V(current_state)]

    # Move to the next state
    current_state = next_state

    # Break if current state is a final state (fail safe feature)
    if (current_state == V(s_T))
      break
```

3.9.2 SARSA

Now moving to the policy improvement for a TD approach, there is not much of a difference with previous works. This is because it is still true to the GPI theory; hence, the only constraint is the same as the evaluation, obtaining an averaged state-action value tuple for one state to another.

In this particular case, the main idea to follow the flow of actions required to move from one state-action to another:

$$\mathbf{State}_t \rightarrow \mathbf{Action}_t \rightarrow \mathbf{Reward}_{t+1} \rightarrow \mathbf{State}_{t+1} \rightarrow \mathbf{Action}_{t+1}$$

If the first letter of each segment is taken, the acronym SARSA becomes clear. This policy improvement requires the use of the new values as in the evaluation: the inclusion of the state-action current value, the *step discount* (α), and the TD error handler.

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \quad (3.24)$$

Having the Bellman equation (3.24), it becomes a relatively straightforward matter to implement an algorithm that fulfills the role of improvement.

III LITERATURE REVIEW

SARSA Algorithm

```
# Get the step discount  $0 < \alpha \leq 1$ 
get alpha

# Get the factor discount gamma
get gamma

# Get an epsilon  $\epsilon$  value
get epsilon

# Initialize all state-action values randomly, except the terminal state, which will contain a zero value-state
 $Q(s,a) = \text{random}(s,a)$ 
 $Q(s_T,a) = 0$ 

# Get the number of episodes
get no_episodes

# Get the number of steps
get no_steps

for each (no_episodes)
  #Initialize S
  initialize current_state

  # Choose A from S using policy derived from Q using  $\epsilon$ -greedy
  random_number = random()
  If (random_number < epsilon)
    action =  $\text{random}_a Q(\text{current\_state}, A)$ 
  else
    action =  $\text{argmax}_a Q(\text{current\_state}, A)$ 

  for each (no_steps)
    # Take action A and observe R, S'
    reward, next_state = environment.apply(action)

    # Choose A' from S' using policy derived from Q using  $\epsilon$ -greedy
    random_number = random()
    If (random_number < epsilon)
      next_action =  $\text{random}_a Q(\text{next\_state}, A)$ 
    else
      next_action =  $\text{argmax}_a Q(\text{next\_state}, A)$ 

    # Apply Bellman equation
     $Q(\text{current\_state}, \text{action}) = Q(\text{current\_state}, \text{action}) + \alpha(\text{reward} + \gamma(Q(\text{next\_state}, \text{next\_action})) - Q(\text{current\_state}, \text{action}))$ 

    # Move to the next state-action
    action = next_action
    current_state = next_state

  # Break if current state is a final state (fail safe feature)
  if (current_state ==  $Q(s_T, A)$ )
    break
```

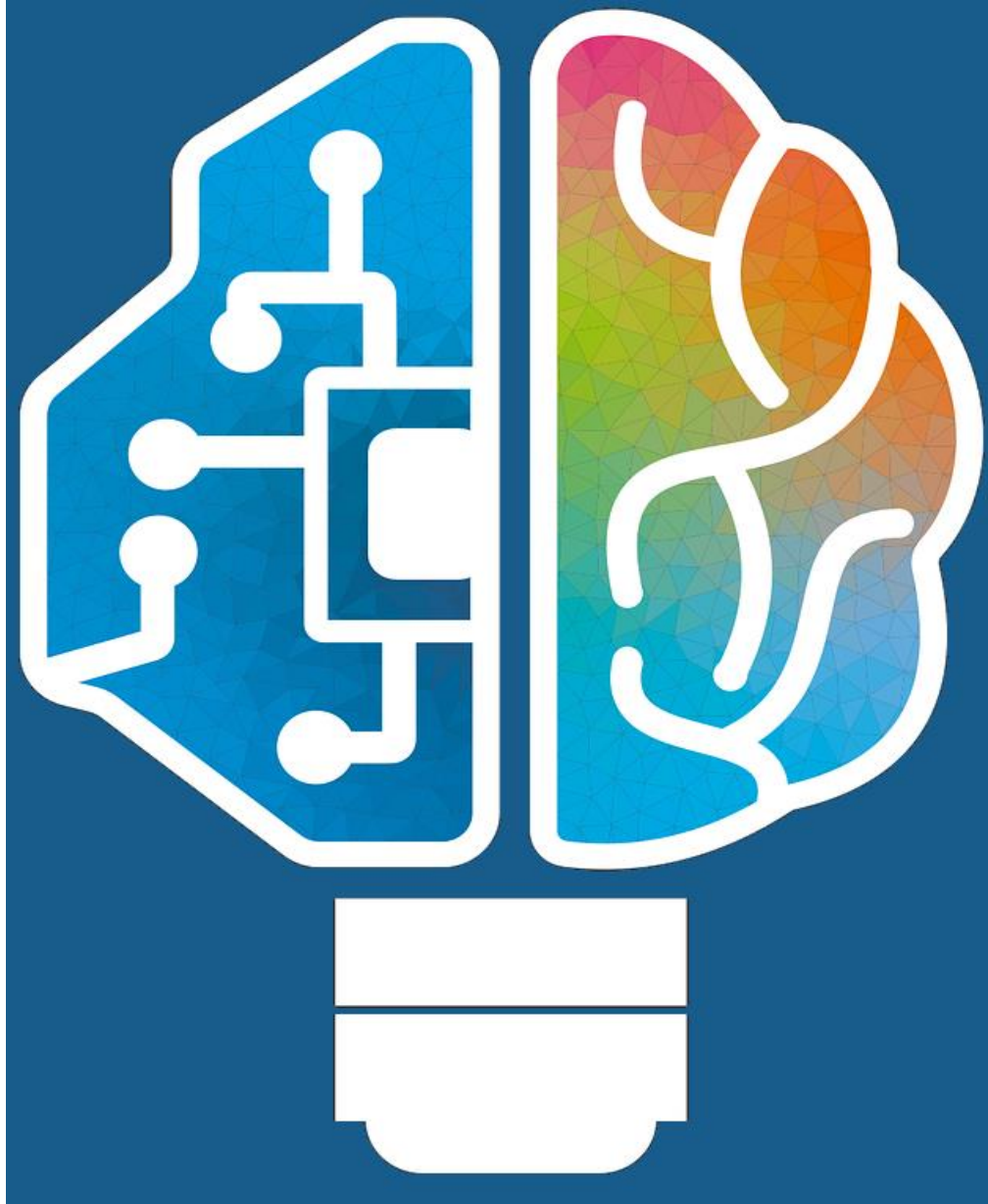
Now all the elements required for the RL AI within the scope of the project have been covered. TD(0) and SARSA provide the perfect framework for the constraints of the project:

III LITERATURE REVIEW

Continuous task with no “end state” and learning from one state at a time. The mathematical approach is already presented on pseudo code and will be expanded and fitted on proper Python code without losing the essence of the concept developed through this chapter.

IV DESIGN AND METHODOLOGY

IV DESIGN AND METHODOLOGY



IV DESIGN AND METHODOLOGY

4.1 Software Requirements Specification (SRS) for T-MegaloRide AI

4.1.1 Product Perspective

T-MegaloRide AI is an SW module part of the T-MegaloRide project. This module is intended to handle the project's AI RL requirements that contemplate the personalized automatic assistance control for a Pedelec vehicle.

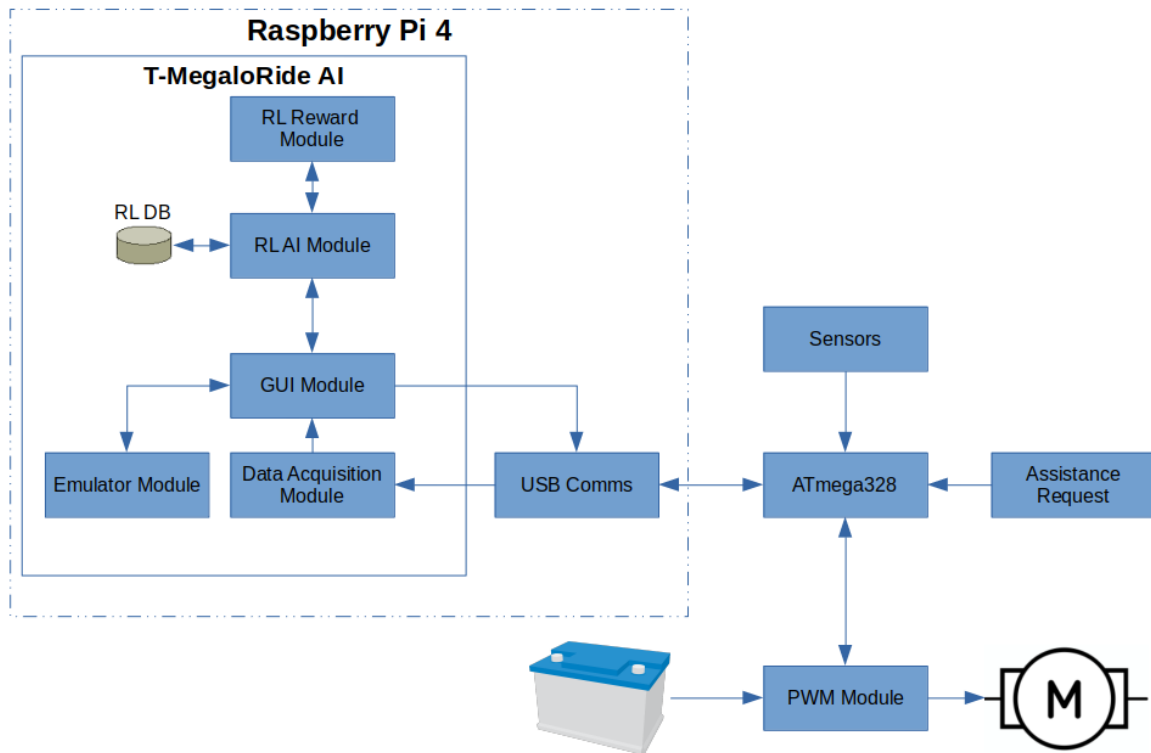


Figure 4.1. T-MegaloRide project.

As shown in figure 4.1, *T-MegaloRide AI* is one of many components of a larger project. Using a Serial-Over-USB interface, it will interact with the other modules, but as well it will have an internal emulator for purposes of testing and development. The target HW for the project is a Raspberry Pi version 4, running over a Linux OS Raspbian.

IV DESIGN AND METHODOLOGY

4.1.2 Product Functions

The T-MegaloRIde AI SW will provide the following functionality:

- RL module capable of learning and personalize pedaling assistance power on automatic operation. This module will be fed with the environment status once every second, and it will return the appropriate action for the external Pedelec controller.
- RL reward module with all the constraints required to provide the reward for any given state-action.
- Data acquisition module capable of handle USB communications, including data interpretation and subsequent storage.
- Emulator module capable of recreating a simulated human behavior during a Pedelec riding. This Emulator will provide feedback every second and have an interface to provide the RL module feedback and react accordingly.
- GUI module with all the necessary components for the interaction between the RL, emulator, and data acquisition modules. Also, it will provide all visual references to evaluate the functionality of the modules working together.

4.1.3 User Classes and Characteristics

The project will be developed on different files encapsulating the functionality in an independent approach to ensure parallel work and independence given a well-formed communications structure between each object. Each file will contain the Class definition and implementation to be easily imported by third-party modules:

IV DESIGN AND METHODOLOGY

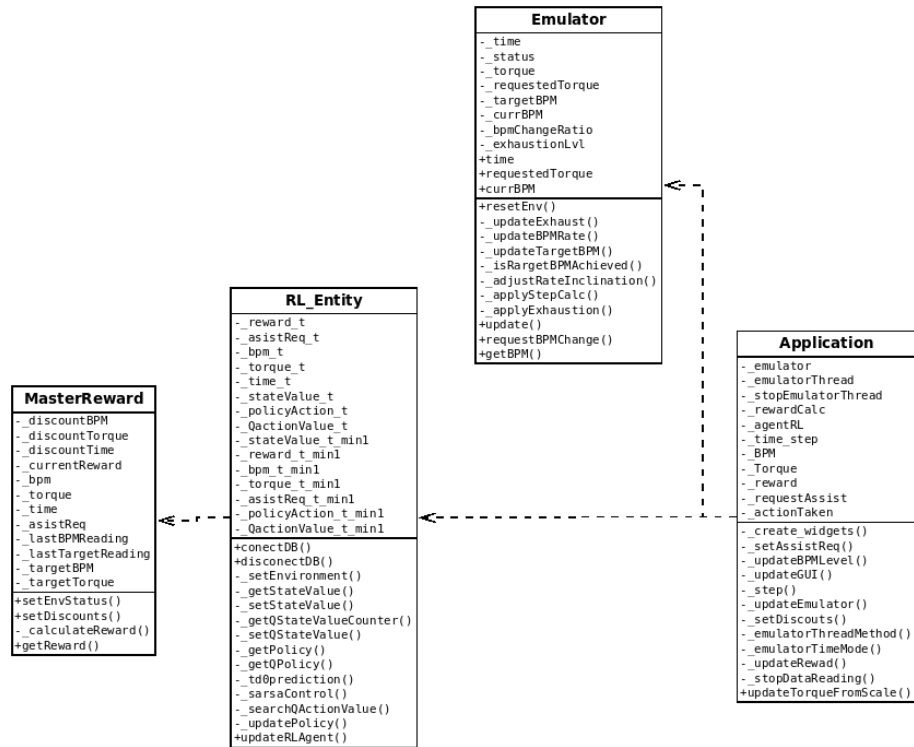


Figure 4.2 T-MegaloRide UML project.

Class RL_Entity

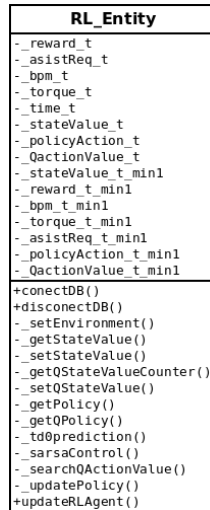


Figure 4.3. RL_entity class.

IV DESIGN AND METHODOLOGY

This class contains all the methods and logic for an RL entity (environment and agent), except the reward. Due to complexity constraints, this was separated into its own class.

Attributes

<code>_reward_t</code>	# Last calculated reward
<code>_assistReq_t</code>	# The current Assist Request value for calculations
<code>_bpm_t</code>	# The current BPM value for calculations
<code>_torque_t</code>	# The current Torque value for calculations
<code>_time_t</code>	# The current time value for calculations
<code>_stateValue_t</code>	# Calculated State value for current state
<code>_policyAction_t</code>	# Current action taken
<code>_QactionValue_t</code>	# Current state-action value
<code>_stateValue_t_min1</code>	# Previous state value
<code>_reward_t_min1</code>	# Previous Reward
<code>_bpm_t_min1</code>	# Previous state BPM
<code>_torque_t_min1</code>	# Previous state Torque
<code>_assistReq_t_min1</code>	# Previous assist request
<code>_policyAction_t_min1</code>	# Previous step action taken
<code>_QactionValue_t_min1</code>	# Previous step-action value

Methods

`connectDB()`

This method tries to connect to the tmealoride database on the local SQL server.

IV DESIGN AND METHODOLOGY

`disconnectDB()`

This method tries to disconnect to the tmegaloride database on the local SQL server.

`_setEnvironment()`

This method is a setter for current environment values provided either from the emulator or the USB input, plus the reward unit (which is independent so far from the RL entity).

`_getStateValue()`

This method searches for the state value of the given state (bpm, torque, assistReq) in the `state_value` table.

`_setStateValue()`

This method updates the state-value of the given state (bpm, torque, assistReq).

`_getQStateValueCounter()`

This method gets the $Q(s,a)$ counter for the given action.

`_setQStateValue()`

This method updates the $Q(s,a)$ of the given state-action pair (bpm, torque, assistReq, action).

`_getPolicy()`

This method searches for the Policy Table and select the action.

`_getQPolicy()`

This method gets the policy using the values from the $Q(s,a)$ values using greedy-epsilon.

`_td0prediction()`

This method will predict the new state value based on the theory of TD(0).

`_sarsaControl()`

This method will optimize the action taken using the SARSA control.

IV DESIGN AND METHODOLOGY

`_searchQActionValue()`

This method search for the state-action value on the Q Table for a specific action (it does not determine the best action, just search the value for the state and action given).

`_updatePolicy()`

This method chooses the max $Q(s,a)$ from t-1 and assign it to the policy of t-1 state.

`updateRLAgent()`

This method is the primary driver to implement the TD(0) and the SARSA techniques for the Reinforcement Learning entity of the TMegaloRide AI.

Class **MasterReward**

MasterReward
-_discountBPM
-_discountTorque
-_discountTime
-_currentReward
-_bpm
-_torque
-_time
-_asistReq
-_lastBPMReading
-_lastTargetReading
-_targetBPM
-_targetTorque
+setEnvStatus()
+setDiscounts()
-_calculateReward()
+getReward()

Figure 4.4. Emulator class.

This class contains all the logic required to run the human effort emulator for the training of T-MegaloRide_AI. This module intends to be used as standard, repeatable experiment data for evaluation of the different approaches for the AI RL network and provide data for debugging.

NOTE. For debugging purposes, the time/step will be provided externally.

IV DESIGN AND METHODOLOGY

Attributes

<code>_time</code>	# Current time in seconds (or step on stepped executions)
<code>_status</code>	# Current Emulator Status
<code>_torque</code>	# Current Torque Level (6 is the minimum allowed)
<code>_requestedTorque</code>	# Requested Torque level for calculations
<code>_targetBPM</code>	# Stabilization BPM value given the current torque
<code>_currBPM</code>	# Current calculated BPM
<code>_bpmChangeRatio</code>	# The slope on which the BPM approaches stabilization level
<code>_exhaustionLvl</code>	# Current exhaustion level
<code>time</code>	# Setter/Getter for <code>_time</code>
<code>requestedTorque</code>	# Setter/Getter for <code>_requestedTorque</code>
<code>currBPM</code>	# Getter for <code>_currBPM</code>

Methods

`resetEnv()`

This method resets all internal attributes and sets the status to STANDBY. It is a healthy approach to always reset the environment before use.

`_updateExhaust()`

This method updates the exhaustion level based on the time/step given.

`_updateBPMRate()`

This method calculates the stabilization RPM and determine if the current stabilization RPM matches the calculated stabilization RPM.

`_updateTargetBPM()`

IV DESIGN AND METHODOLOGY

This method checks if any change in the RPM is required. If so, it updates the attributes needed to change the current BPM into the targeted one.

`_isRargetBPMAchieved()`

This method determines if the target BPM has been achieved, it uses a range defined by $\text{currentBPM} \pm \text{TOLERANCE}$. This tolerance is intended to avoid "jump over" and then "fall short" infinite loops.

`_adjustRateInclination()`

This method determines whether the BPM change rate is going in the desired direction. This means that if a stabilization BPM is lower than the current BPM, we must warrant a decrease in the BPM even if the rate is a positive value. On the other hand, if the stabilization BPM is higher than the current BPM, the rate must be increased even if the rate is a negative value.

`_applyStepCalc()`

This method calculates the BPM increment/decrement given the current `bpmChangeRatio`

`_applyExhaustion()`

This method applies the accumulated exhaustion to the current BPM.

`update()`

This method updates the calculations given the current settings BPM. It also checks for BPM change requests.

NOTE. This method is intended to be executed once every second, but due to debugging concerns, it will require from an external request which can be done 1 per second or as a step by step for snap values

`requestBPMChange()`

IV DESIGN AND METHODOLOGY

This method is request for a BPM recalculation due a Torque level change.

getBPM()

This method is a getter of the current calculated BPM.

Class Application

Application
-_emulator -_emulatorThread -_stopEmulatorThread -_rewardCalc -_agentRL -_time_step -_BPM -_Torque -_reward -_requestAssist -_actionTaken
-_create_widgets() -_setAssistReq() -_updateBPMLevel() -_updateGUI() -_step() -_updateEmulator() -_setDiscouts() -_emulatorThreadMethod() -_emulatorTimeMode() -_updateRewad() -_stopDataReading() +updateTorqueFromScale()

Figure 4.5. Application class.

This class contains all the widget controls for TKinter and TTK. This implementation is intended as a modular section of the TMEgaloride_AI project, which means no functional code is to be included on this script other than the controls for the GUI itself.

Attributes

```
_emulator          # Emulator instance form tmegaloride_emulator
_emulatorThread    # Emulator thread used on time-based mode
_stopEmulatorThread # Signals the emulator thread to stop
_rewardCalc        # Reward instance from tmegaloride_ai
```

IV DESIGN AND METHODOLOGY

`_agentRL` # Artificial module for RL agent
`_time_step` # Current environment time/step
`_BPM` # Current environment BPM
`_Torque` # Current environment Torque
`_reward` # Last given reward
`_requestAssist` # Request for auxiliary assistance
`_actionTaken` # Last action taken

Methods

`_create_widgets()`

This method builds and places all the widgets used on the GUI. Usually, a method should not be this long (75 lines as the standard states). This method is the exception due to the proper nature of the placement of a large number of widgets. Dividing it into several methods do not improve actual reading and maintenance of codes.

`_setAssistReq()`

This method set Assist request to 1.

`_updateBPMLevel()`

This method updates the BPM label scale. It requires some work since the widget works from top to floor (why? just why?), meaning that the percentage must be calculated and reversed in order to provide proper visual context.

`_updateGUI()`

This method updates the GUI values according the data input and the emulator changes.

`_step()`

IV DESIGN AND METHODOLOGY

This method executes a single emulator step. This allows the RL agent debugging since it can be carefully exterminated while the simulator waits for a second step.

`_updateEmulator()`

This method updates the emulator, this includes applying a time step and requesting a torque change.

`_setDiscounts()`

This method tries to update the discount factors for the time, BPM, and torque values used on the reward calculations. Since the widget entries are not limited to numbers, some prevalidation will be required.

`_emulatorThreadMethod()`

This method is intended to integrate all the functionality required for the emulator independent thread. This is due to the multithread approach used encapsulates methods as a "child process."

`_emulatorTimeMode()`

This method executes a time-based (1 per second) execution of the emulator. Due to the independent nature of the emulator and the RL agent, this method requires its own thread to work with or without the RL agent being active.

`_updateReward()`

This method updates the reward given the current input values and discount values.

`_stopDataReading()`

This method stops data reading (whatever method might be in use).

`updateTorqueFromScale()`

This method updates the Torque value when the Torque scale is changed.

IV DESIGN AND METHODOLOGY

4.1.4 Operating Environment

- *Target HW.* Raspberry Pi 4 (4 Gb of RAM).
- *Operative System.* Linux Raspian (Debian 10 “Buster”, Kernel 4.19).
- *Language.* Python 3.8
- *Language Libraries dependencies.* Tkinter 8.6, sqlite3, datetime, math, random, and threading python libraries.
- *DB.* SQLite 3.

4.1.5 Assumptions and Dependencies

- The USB communications interface is intended to use Serial-Over-USB, which the OS will handle, and the T-MegaloRide AI will get the information from the assigned USB port assigned by the OS and be treated as a regular text file.
- The USB communications interface will work under 115200 baud transmission with readings available every 20 – 23 milliseconds.
- DB will be available and deployed with the required tables for both value functions and Q values.
- A Human heartbeat and Torque production analysis will be performed in order to provide the initial constrains for the RL reward system and the Emulator.

4.2 External Interface Requirements

4.2.1 User Interfaces

The main user interface will be provided with a GUI containing all the necessary controllers to train, test and debug the T-MegaloRide AI protect, as show in the figure 25:

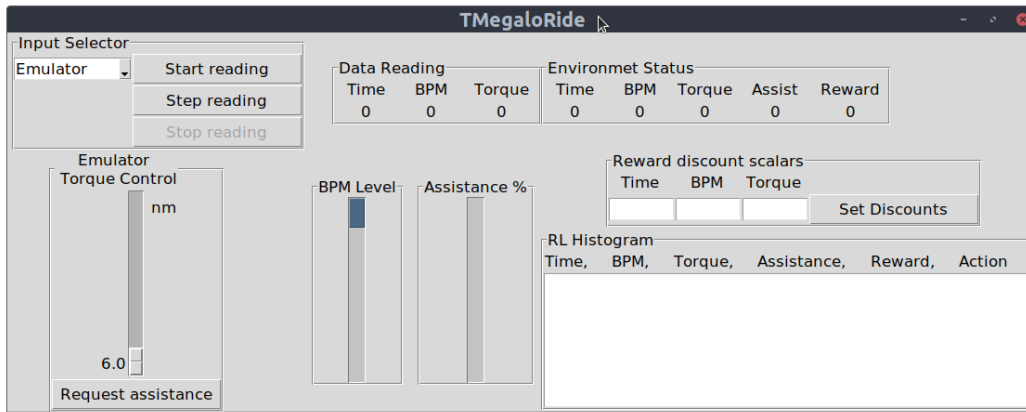


Figure 4.6. T-MegaloRide AI GUI.

Input selector

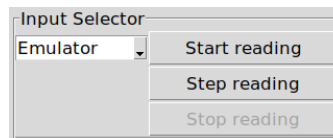


Figure 4.7. T-MegaloRide AI GUI Input Selector.

This widget is intended to select the emulator and the mode of operation:

- Start reading. This mode of operation is a time-based execution, where each step is taken at 1 second.
- Step reading. This is a manual step by step mode where the execution is provided manually, one click on the button means one step. This method is intended mostly for testing and debugging.

Emulator Torque Control

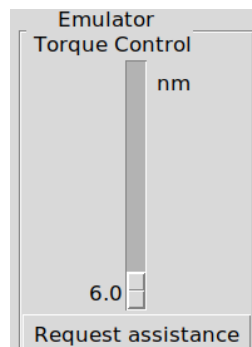
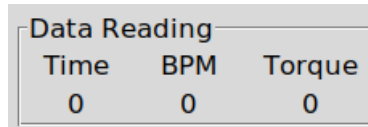


Figure 4.8. T-MegaloRide AI GUI Emulator Torque Control.

IV DESIGN AND METHODOLOGY

This scale widget has the means to control the torque requirements from the emulated rider. This function is essential for training and debugging the RL Agent. This section also provides an Assistance Request button to simulate the feedback from the emulated rider.

Data Reading

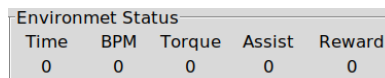


Data Reading		
Time	BPM	Torque
0	0	0

Figure 4.9. T-MegaloRide AI GUI Data Reading.

The data reading widget displays the current time, BPM reading, and torque produced by the rider.

Environment Status



Environment Status				
Time	BPM	Torque	Assist	Reward
0	0	0	0	0

Figure 4.10. T-MegaloRide AI GUI Environment Status.

The environment Status widget shows the RL environment values at the latest time step.

BPM Level

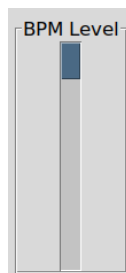


Figure 4.11. T-MegaloRide AI GUI BPM Level.

Visual display for the heartbeat monitor. The scale is used for rapid visualization of the level of effort of the cardiovascular system.

IV DESIGN AND METHODOLOGY

Reward Discount

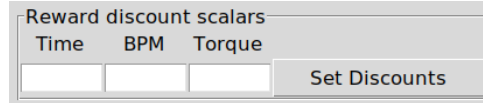


Figure 4.12. T-MegaloRide AI GUI Reward Discounts.

Widget to manipulate the Time, BPM, and Torque values at the time for reward calculations. Its function is mostly for research and testing.

RL Histogram

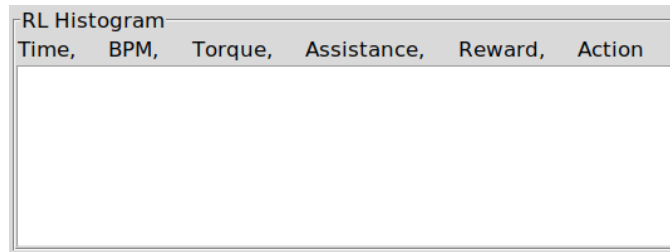


Figure 4.13. T-MegaloRide AI GUI RL histogram.

Widget to save a histogram of the environment, reward, and actions of the RL module. It can be used for debugging or RL evaluation. The contents of the widget can be copied for manipulation on a third-party SW.

4.2.2 Hardware Interfaces

ATmega328

USB connection using Serial-Over-USB will be used to communicate with an external ATmega328 development board. This board will have all SW requirements (out of the current scope) to acquire external sensor data and provide it at a 20-23 milliseconds rate.

IV DESIGN AND METHODOLOGY

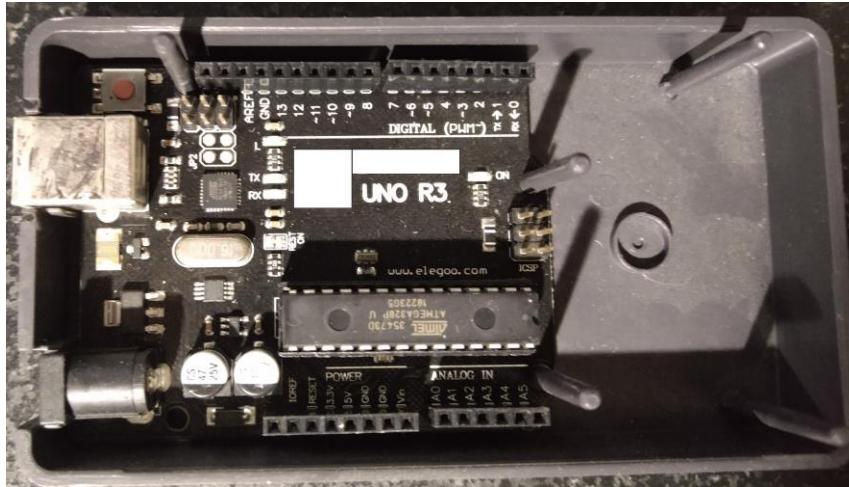


Figure 4.14 ATmega328 development board.

HDMI Port

The Raspberry Pi 4 uses a micro HDMI port required for an external monitor. This port is required for local use of the T-MegaloRide AI SW.

Touch screen monitors are supported but are required to be on the list of supported Raspbian touchscreen devices (not in the scope of this project).

USB 2.0 Port

The USB communications will be handled directly by the OS Raspbian. The data lecture will be handled as a file using the typical Linux text file behavior for USB devices.

Mouse and Keyboard

A mouse and keyboard are required for local operation on the Raspberry Pi, yet, the OS is expected to handle these devices, and the Tkinter library will handle the visual interaction.

4.2.3 Software Interfaces

SQLite 3

A local instance of SQLite 3 will be running on the same working folder as the T-MegaloRide AI project. The communications will be handled by Python 3.8 library sqlite3. The DB

IV DESIGN AND METHODOLOGY

protection is handled at the OS level, meaning that only root and creator group users can access the DB.

Python libraries

- *sqlite3*. Library to handle the communications with the local SQLite 3 DB.
- *math*. This module provides access to the mathematical functions defined by the C standard.
- *random*. This module implements pseudo-random number generators for various distributions.
- *csv*. The csv module implements classes to read and write tabular data in CSV format.
- *datetime*. The datetime module supplies classes for manipulating dates and times.
- *tkinter*. The tkinter package (“Tk interface”) is the standard Python interface to the Tk GUI toolkit.
- *time*. This module provides various time-related functions.

4.3 System Features

4.3.1 Data acquisition.

- *Description and Priority.*

This module will handle the data acquisition and storage from the USB port. Therefore, the priority stands as highest (9) due to the need for data to work on all other modules. The risk is low (2) because most of the communication details are constrained in the OS (Linux). The benefit of early development is the parallel work with other modules.

- *Stimulus/Response Sequences.*

Execute the python script “usb_reader.py”.

- *Functional Requirements:*

IV DESIGN AND METHODOLOGY

- **REQ-1:** Set up the Serial-Over-USB overall parameters:
 - baudrate = 115200 # Arduino writing speed
 - bytesize = EIGHTBITS # Number of bits per bytes
 - parity = PARITY_NONE # set parity check: no parity
 - stopbits = STOPBITS_ONE # number of stop bits
 - timeout = 2 # 2. 0: non-blocking mode, return immediately
 - rtscts = False # disable hardware (RTS/CTS) flow control
 - dsrdtr = False # disable hardware (DSR/DTR) flow control
 - readStatus = True # Global flag for multi-trading communication on USB read
- **REQ-2:** Open the serial port “/dev/ttyACM0” (Arduino default port). If port unavailable, send the message [error open serial port: /dev/ttyACM0] to the stdout and exit script.
- **REQ-3:** Create CSV file using the following name constrains [day + month + Year + Hour + Minute + Second].csv
- **REQ-4:** Continuous read of serial port “/dev/ttyACM0” and redirect the data to the CSV file buffer. If a message fails, send warning through stdout. If message is incomplete, ignore it.
- **REQ-5:** Every second worth of data insert a special line into the CSV file with the following format ['T','T','T','T','T','T','T','T']. Then send a stdout notification message.
- **REQ-6:** Use a special thread to monitor the keyboard, in any stance where a key is pressed stop the serial read and save the CSV file.

4.3.2 Emulator

- *Description and Priority*

IV DESIGN AND METHODOLOGY

This module will recreate the BPM reactions from a test subject given a torque requirement provided. The priority is high (8) due to the proper functionality is a requirement by the RL module for testing and training. The benefit from this module is the unlimited capabilities to test and train the RL module creating all scenarios that are deemed necessary. The risk is in the medium range (5) due to the lack of diverse test subjects. This means that the emulator will contain the bias of the single test subject.

- *Stimulus/Response Sequences.*
 - Execute the python script “tmegaloride_gui.py”.
 - Select the emulator from the “input selector”.
 - Press the [start reading] button to start emulation.
 - Press the [stop reading] button to stop emulation.
 - From the “emulator torque section” move the slide bar to control the torque generation requirement.
- *Functional Requirements*
 - REQ-1: Set up the Emulator constrains:
 - EXHAUSTION = 0.00000031 # Rate of exhaustion per second
 - MINIMUM_BPM = 90.0 # Minimum BPM the emulator will provide
 - BPM_AVERA_SLOPE = 4.4107 # the slope to calculate the bpm change rate m=
 - BPM_AVERA_BIAS = 55.5755 # The bias to calculate the bpm change rate b=
 - BPM_RATE_SLOPE = 0.0441 # The slope to calculate the speed ratio for BPM
change m=
 - BPM_RATE_BIAS = 0 # The bias to calculate the speed ratio for BPM
Change b=
 - TOLERANCE = 0.02 # Tolerance for the approximation of the stabilization
BPM needed to avoid jump over and then fall

IV DESIGN AND METHODOLOGY

Short loops

- MIN_TORQUE = 6.0 # Minimum allowed Torque
- MAX_Torque = 60 # Maximum allowed Torque
- DEFAULT_BPM = 90 # The default BPM when not provided

- **REQ-2:** Method to update the exhaustion value. This method is intended to apply a relative exhaustion to the current BPM value, based on the exhaustion level from REQ-1 and the current elapsed time and applying the following equation:
 - $\text{exhaustion} = 1 + (\text{math.exp}(\text{EXHAUSTION} * \text{time}) - 1)$
- **REQ-3:** Create a method to calculate the stabilization²¹ BPM and determine if the current stabilization BPM matches the calculated stabilization BPM.
- **REQ-4:** A method checks if any change on the RPM is needed, if so it updates the attributes needed to change the current BPM into the targeted one and checks if actually a change is needed.
- **REQ-5:** A method determines if the target BPM has been achieved, it uses a range defined by $\text{current BPM} \pm \text{TOLERANCE}$. This tolerance is intended to avoid "jump over" and then "fall short" infinite loops.
- **REQ-6:** A method to determine whether the BPM change rate is going in the desired direction. This means that if a stabilization BPM is lower than the current BPM, we must grant a decrease in the BPM even if the rate is a positive value. On the other hand, if the stabilization BPM is higher than the current BPM, the rate must be increased even if the rate is a negative value.
- **REQ-7:** A method that calculates the BPM increment/decrement given the current BPM Change Ratio.

²¹ Stabilization BPM is considered the BPM average value that is expected at a given Torque production, this is needed to determine how much the BPM will be increased/decreased when Torque generation get higher/lower.

IV DESIGN AND METHODOLOGY

- **REQ-8:** A method that applies the accumulated exhaustion to the current BPM.
- **REQ-9:** A method that updates the calculations given the current requested BPM. It also checks for BPM change requests. NOTE. This method is intended to be executed once every second, but due to debugging concerns, it will require from an external request which can be done 1 per second or as a step by step for snap values.
- **REQ-10:** A method that provides the current calculated BPM.

4.3.3 Reward Module.

- *Description and Priority*

This module will contain all the reward logic required by the agent to learn through the different actions taken. This module will take the current BPM, Torque produced, and the “assistance requested” status to calculate the proper reward. Since the agent cannot learn without this section, the priority is high (7), the risk is medium (4) since there is intended to use simple second-grade equations for both the BPM and Torque inputs, and a static behavior when the “assistance requested” status is triggered.

- *Stimulus/Response Sequences.*

- Execute the python script “tmegaloride_gui.py”.
- Select the emulator from the “input selector”.
- Press the [start reading] button to start the reward evaluation.
- Press the [stop reading] button to stop reward evaluation.
- Current reward will be displayed on the “Environment Status” section.
- The “RL histogram” section will contain all rewards from the latest run.

- *Functional Requirements.*

- **REQ-1:** Set up the Reward module constrains:

IV DESIGN AND METHODOLOGY

- $MAX_TORQUE = 29.8$ # Maximum torque allowed before punishment
 - $MAX_BPM = 175$ # Maximum BPM Allowed before punishment
 - $targetBPM = 135$ # Current target BPM, to get from the DB
 - $targetTorque = 17$ # Current target Torque, to get from DB
- **REQ-2:** Determine if the current BPM has changed since the last reading. If it has changed, proceed with the BPM reward calculations requirement (REQ-6). If not, then evaluate if BPM is on target and provide a reward of [20]. If not on target, provide a punishment of [-1].
- **REQ-3:** Determine if the current Torque production has changed since the last reading. If it has changed, proceed with the Torque reward calculations requirement (REQ-5). If not, then evaluate if the Torque production is on target and provide a reward of [10]. If not on target, provide a punishment of [-1].
- **REQ-4:** Determine if the current Torque generation is converging into the target Torque. If not, change the calculated reward into a punishment (change the value into a negative number).
- **REQ-5:** Determine if the current BPM is converging into the target BPM. If not, change the calculated reward into a punishment (change the value into a negative number).
- **REQ-6:** Calculate the Torque reward using the following equation:
- $TorqueReward = (-1.0 * ((torque - targetTorque)^2)/200) + 10$
- **REQ-7:** Calculate the BPM reward using the following equation:
- $BPMReward = (-1 * ((bpm - targetBPM)^2)/303) + 20$
- **REQ-8:** If the Torque production is higher than the target Torque production, divide the BPM reward by 4 as a bias to diminish too high initial Torque production requirements.

IV DESIGN AND METHODOLOGY

➤ **REQ-9:** Calculate the Total Reward using the following equation:

$$\blacksquare \text{ TotalReward} = \text{BPMReward} + \text{TorqueReward}$$

4.3.4 Reinforcement Learning module.

- *Description and Priority*

This module will contain all logic to implement a Reinforcement Learning AI, and this includes the interaction with the environment, the reward module (imported), and the agent segment. The priority is high (7) since it is the project's main focus, yet it cannot be developed until previous requirements are completed. Therefore, the priority is on par with the previous requirements. Risk is medium (5) since the environment is well defined, the reward system is reliable, and the control (TD-0) and prediction(SARSA) sections have a robust theory background.

- *Stimulus/Response Sequences*

- Execute the python script “tmegaloride_gui.py”.
- Select the emulator from the “input selector”.
- Press the [start reading] button to start the RL agent.
- Press the [stop reading] button to stop RL agent.
- The “RL histogram” section will contain all actions taken from the latest run.

- *Functional Requirements*

➤ **REQ-1:** Set up the RL module constrains:

- STATE_ALPHA = 0.2 # Alpha for Bellman equation
- STATE_GAMMA = 0.8 # Gamma for Bellman equation
- EPSILON = 0.1 # Epsilon for MDP exploration

➤ **REQ-2:** Method to connect and disconnect with local sqlite3 DB named “tmegaloride_rl.db”.

IV DESIGN AND METHODOLOGY

- **REQ-3:** Method to set the environment, this includes time, bpm, torque, assisting, and stop signal.
- **REQ-4:** Method to get the current state value from DB.
- **REQ-5:** Method to set the current state value from DB.
- **REQ-5:** Method to set the current state value from DB.

4.3.5 GUI.

- Description and Priority

The GUI contains the complete project and is intended to be the main user interface while training the RL Agent with the emulator data. It serves a joint point of all previous modules, therefore the priority is medium (5) since it cannot be fully developed until all other modules are completed or at least in a debugging state. The risk related to this task relies in the tkinter library used to create the GUI through python, it is not user friendly and does not use any tool to visually set and place the different visual components, therefore a medium risk (6) is expected.

- Stimulus/Response Sequences

- Execute the python script “tmegaloride_gui.py”.

- Functional Requirements

➤ **REQ-1:** Set up the GUI Sections:

- Frames
- Combo Box
- Scale
- Progress bar
- Text
- Labels

IV DESIGN AND METHODOLOGY

- Buttons
- **REQ-2:** Import and create an instance of the emulator.
- **REQ-3:** Import and create an instance of the RL module.
- **REQ-4:** Create a method to request assistance.
- **REQ-5:** Create a method to update the BPM level on the designated scale.
- **REQ-6:** Create a method to update the complete GUI, this includes:
 - Data Reading.
 - Tome
 - BPM
 - Torque
 - Environment status.
 - Time
 - BPM
 - Torque
 - Assistance Requested
 - Last Reward
 - Emulator torque control scale.
 - BPM Level scale.
 - RL Histogram:
 - Time.
 - BPM.
 - Torque.
 - Assistance Requested.
 - Last Reward.
 - Action Taken.

IV DESIGN AND METHODOLOGY

- **REQ-7:** Step mode method to execute the training on steps instead of time-based.
- **REQ-8:** Emulator update Method to provide the latest changes on the Torque generation requirements.
- **REQ-9:** Emulator separate thread method for time based training. This is a requirement so the emulator can keep working in its own tread and do not require wait for the GUI process (if any)
- **REQ-10:** Emulator time-based mode Method. This method will setup and execute an instance of the separated thread for the emulator.
- **REQ-11:** Method to stop the data reading and the agent training.
- **REQ-12:** Method to update Torque generation from the designated scale.

4.3.6 Modified Bicycle for data acquisition

- *Description and Priority*

Although the project is focused on the RL IA and SW in general, there is an intrinsic need for data acquisition to create the emulator and the initial policies. In order to fulfill this need, this section will consider such project requirements, although it will not be fully detailed on the HW components, just the functional requirement. Therefore, the priority is high (10) since it is essential for data sampling and the risk level is moderately high (8) since it requires the work with sensors and raw data preprocessing, not part of the current project, but necessary nerveless.

- *Stimulus/Response Sequences.*

- Energize the sensors and the ATmega328P development board.
- Connect the ATmega328P development board to the host Raspberry Pi though the USB.

- *Functional Requirements*

IV DESIGN AND METHODOLOGY

- REQ-1: Adapt a generic mountain bicycle for sampling. A generic mountain bike will be equipped with the necessary sensors for the compilation of statistical data from a test subject:
 - REQ-1: Adapt a generic mountain bicycle for sampling. A generic mountain bike will be equipped with the necessary sensors for the compilation of statistical data from a test subject:
 - Torque sensor (N.m) placed into the pedals bottom bracket.
 - Photoplethysmography sensor with a fixture to enable it to be placed in the left ear lobe.

4.4 Implementation

This section covers the code developed to fulfill all SRS for the T-MegaloRide AI project. As a note, the code will be summarized to capture only the critical sections as the requirements state.

4.4.1 Data acquisition implementation.

REQ-1: Set up the Serial-Over-USB overall parameters

```
baudrate = 115200      # Arduino writing speed
bytesize = EIGHTBITS  # Number of bits per bytes
parity = PARITY_NONE   # set parity check: no parity
stopbits = STOPBITS_ONE # number of stop bits
timeout = 2            # 2. 0: non-blocking mode, return immediately
rtscts = False         # disable hardware (RTS/CTS) flow control
dsrdtr = False         # disable hardware (DSR/DTR) flow control
readStatus = True      # Global flag for multi-trading communication on USB read
```

REQ-2: Open the serial port “/dev/ttyACM0”

```
ser.port = "/dev/ttyACM0" # Default Arduino terminal
```

REQ-3: Create CSV file

```
now = datetime.now()
dtString = now.strftime("%d_%m_%Y_%H_%M_%S")
fileName=dtString + ".csv"
...
```

IV DESIGN AND METHODOLOGY

```
def saveCSV(fileName, value):
    with open(fileName, "a", newline="") as f:
        writer = csv.writer(f)
        writer.writerow(value)
```

REQ-4: Continuous read of serial port.

```
# Clean whatever might be in the port for a clean start
ser.flushInput()      #flush input buffer, discarding all its contents
ser.flushOutput()     #flush output buffer, aborting current output
                      #and discard all that is in buffer

time.sleep(1) #give the serial port sometime to receive the data

while True:
    # OK, the serial.readline() actually returns bytes, not string SO it is filled with
    # garbage, since the Arduino is set to work on ascii, we need to decode it as such
    # fortunately the decode() method comes really handy (it works with utf-8 but it
    # crash and burns on incomplete packets so ascii it is)
    try:
        response = ser.readline().decode("ascii")
    except:
        print("bad serial message")
        continue

    # Well since we are dealing with ascii we need to manually remove the carriage ret-
    # urn '\r' and the new line '\n' from the line
    response = response.rstrip("\r\n")

    # now let's add the read value to a list for validation
    if (len(response) > 1 ):
        # Assign the values to a list
        message=[e for e in response.split(',')]

    if (readStatus == False):
        break

    if (len(message) != 8):
        # Incomplete message, just ignore it
        continue
```

REQ-5: Every second worth of data insert a special line into the CSV file

```
secondsCounterCurrent = perf_counter()
if (secondsCounterCurrent - secondsCounterStart >= 60):
```

IV DESIGN AND METHODOLOGY

```
minutesElapsed += 1
print("Minutes Elapsed [{}]" .format(minutesElapsed))
saveCSV(fileName, minuteCSVDivisor)
secondsCounterStart = secondsCounterCurrent

# Save the value on a CSV
saveCSV(fileName, message)
```

REQ-6: Use a special thread to monitor the keyboard.

```
def get_input():
    global readStatus
    keystrk=input('Press enter to stop\n')

    # thread doesn't continue until key is pressed
    print('You pressed: ', keystrk)
    readStatus=False
    print('readStatus is now:', readStatus)

i=threading.Thread(target=get_input)
i.start()
```

4.4.2 Emulator implementation

REQ-1: Set up the Emulator constrains:

```
EXHAUSTION = 0.00000031      # Rate of exhaustion per second
MINIMUM_BPM = 90.0          # Minimum BPM the emulator will provide
BPM_AVERA_SLOPE = 4.4107    # the slope to calculate the bpm change rate m=
BPM_AVERA_BIAS = 55.5755    # The bias to calculate the bpm change rate b=
BPM_RATE_SLOPE = 0.0441    # The slope to calculate the speed ratio for BPM
                             # change m=
BPM_RATE_BIAS = 0          # The bias to calculate the speed ratio for BPM
                             # Change b=
TOLERANCE = 0.02           # Tolerance for the approximation of the stabilization
                             # BPM needed to avoid jump over and then fall
                             # short loops
MIN_TORQUE = 6.0           # Minimum allowed Torque
MAX_Torque = 60            # Maximum allowed Torque
DEFAULT_BPM = 90           # The default BPM when not provided
```

REQ-2: Method to update the exhaustion value.

```
def _updateExhaust(self):
    # Exhaust level is the EXHAUSTION rate times the timestep given
    self._exhaustionLvl = 1 + (math.exp(self.EXHAUSTION * self._time) - 1)
```

IV DESIGN AND METHODOLOGY

```
return 0
```

REQ-3: Create a method to calculate the stabilization BPM.

```
def _updateBPMRate(self):
    #Calculate the stabilization BPM for the requested torque
    self._localChangeRatio = (self.BPM_RATE_SLOPE * self._requestedTorque) + self.BPM_RATE_BIAS

    # If stabilization BPM is equal to current BPM
    if (self._localChangeRatio == self._bpmChangeRatio):
        # Return False
        return False
    else:
        self._bpmChangeRatio = self._localChangeRatio

    # Return True
    return True
```

REQ-4: A method checks if any change on the RPM is needed.

```
def _updateTargetBPM(self):
    #Calculate the stabilization BPM for the requested torque
    self._targetBPM = (self.BPM_AVERA_SLOPE * self._requestedTorque) + self.BPM_AVERA_BIAS

    # If current BPM is greater or equal than MAXIMUM_BPM
    if (self._targetBPM >= self.MAXIMUM_BPM):
        # Make current BPM equal to MAXIMUM BPM
        self._targetBPM = self.MAXIMUM_BPM
        return 0

    # If current BPM is smaller or equal than MINIMUM_BPM
    if (self._targetBPM <= self.MINIMUM_BPM):
        # Make current BPM equal to MINIMUM_BPM
        self._targetBPM = self.MINIMUM_BPM

    # If stabilization BPM is equal to current BPM
    if (self._targetBPM == self._currBPM):
        # Return False
        return False
    else:
        # Return True
        return True
```

REQ-5: A method determines if the target BPM has been achieved.

IV DESIGN AND METHODOLOGY

```
def _isTargetBPMAchieved(self):
    # Local attributes
    self._upperTolerance = None # Upper tolerance for currentBPM to be good enough (eg. +5%)
    self._lowerTolerance = None # Lower tolerance for currentBPM to be good enog (eg. -5*)

    #Get the tolerances
    self._upperTolerance = self._targetBPM * (1.0 + self.TOLERANCE)
    self._lowerTolerance = self._targetBPM * (1.0 - self.TOLERANCE)

    # If currentBPM is between the tolerance range
    if ((self._currBPM < self._upperTolerance) and (self._currBPM > self._lowerTolerance)):
        # Adjust the current BPM
        self._currBPM = self._targetBPM
        self._expStepTrack = 0
        return True

    # Return False
    return False
```

REQ-6: A method to determine whether the BPM change rate is going in the desired direction.

```
def _adjustRateInclination(self):
    # Get then BPM change rate slope
    self._bpmChangeRatio = (self.BPM_RATE_SLOPE * (self._requestedTorque) + self.BPM_RATE_BIAS)

    # If the currentBPM is higher than the targetBPM
    if (self._currBPM > self._targetBPM):
        # If the change BPM ratio is positive
        if (self._bpmChangeRatio > 0.0):
            # Invert the change BPM rate slope
            self._increaseRatio *= -1
    else:
        # If the change BPM ratio is negative
        if (self._bpmChangeRatio < 0.0):
            # Invert the change BPM rate slope
            self._increaseRatio *= -1
    return 0
```

REQ-7: A method that calculates the BPM increment/decrement.

```
def _applyStepCalc(self):
    self._localRandomBPMChance = None # change to add or remove some BPM from the calculations

    # Calculate the current BPM using the bpmChangeRatio
    self._expStepTrack = self._expStepTrack + 1
```

IV DESIGN AND METHODOLOGY

```
# Calculate step increase
self._exponential= (math.exp(self._bpmChangeRatio * 0.0025)) - 1
self._increaseRatio = self._exponential * self._expStepTrack
self._adjustRateInclination()
self._currBPM = self._currBPM + (self._currBPM * self._increaseRatio)

# If current BPM is greater or equal than MAXIMUM_BPM
#print("_currBPM = [{}], MAXIMUM_BPM = [{}]" .format(self._currBPM , self.MAXIMUM_BPM))
if (self._currBPM >= self.MAXIMUM_BPM):
    # Make current BPM equal to MAXIMUM BPM
    self._currBPM = self.MAXIMUM_BPM
    return 0

# If current BPM is smaller or equal than MINIMUM_BPM
if (self._currBPM <= self.MINIMUM_BPM):
    # Make current BPM equal to MINIMUM_BPM
    self._currBPM = self.MINIMUM_BPM

# Apply current exhaustion
self._applyExhaustion()
print("_applyStepCalc: after exhaustion self._currBPM: [{}]" .format(self._currBPM))
return(0)
```

REQ-8: A method that applies the accumulated exhaustion to the current BPM.

```
def _applyExhaustion(self):
    self._currBPM = self._currBPM * self._exhaustionLvl
    if (self._currBPM >= self.MAXIMUM_BPM):

        # Make current BPM equal to MAXIMUM BPM
        self._currBPM = self.MAXIMUM_BPM
        return 0
    return 0
```

REQ-9: A method that updates the calculations given the current requested BPM.

```
def update(self, time_step):
    self.bpmRateChange = False

    # Increase the time/step
    self._time = time_step

    # If status is STANDBY
    if (self._status == self.STANDBY):
```

IV DESIGN AND METHODOLOGY

```
self.resetEnv()
return 0

# Update exhaustion level
# We know something is going on so exhaustion must be applied nerveless of the next states
self._updateExhaust()#

# If status is BPM_CHANGE
if (self._status == self.BPM_CHANGE):
    # Get the BPM change rate
    self._bpmRateChange = self._updateBPMRate()
    self._updateTargetBPM()

    # If rate not changed
    if ( False == self._bpmRateChange):
        # Change the status to STEADY
        self._status = self.STEADY
    else:
        # Change the status to BPM_CHANGING
        self._status = self.BPM_CHANGING
        self._expStepTrack = 0

# If status is BPM_CHANGING
if (self._status == self.BPM_CHANGING):
    # If stabilization BPM has been reached
    if (True == self._isTargetBPMAchieved()):
        #change status to STEADY
        self._status = self.STEADY
    else:
        # Adjust the BPM Change rate inclination/declination
        # Apply time/step calculations
        self._applyStepCalc()

# If status is STEADY
if (self._status == self.STEADY):
    # Just apply exhaustion and return Success
    self._applyExhaustion()
    return 0
```

REQ-10: A method that provides the current calculated BPM.

```
def getBPM(self):
    return round(self._currBPM, 3)
```

IV DESIGN AND METHODOLOGY

4.4.3 Reward Module Implementation

REQ-1: Set up the Reward module constrains

```
MAX_TORQUE = 29.8      # Maximum torque allowed before punishment
MAX_BPM = 175         # Maximum BPM Allowed before punishment
targetBPM = 135       # Current target BPM, to get from the DB
targetTorque = 17     # Current target Torque, to get from DB
```

REQ-2: Determine if the current BPM has changed since the last reading.

```
# If torque is the same as in last reading
if (self._torque == self._lastTargetReading):
    # If torque is the same as in last reading
    self._localTorqueRewardEnabled = False
```

REQ-3: Determine if the current Torque production has changed since the last reading.

```
self._localBPMRewardEnabled = True
if (self._bpm == self._lastBPMReading):
    print("_calculateReward(): Same BPM")
    self._localBPMRewardEnabled = False
```

REQ-4: Determine if the current Torque production is converging into the target Torque.

```
self._localToTorque = True
if(abs((self._torque - self._targetTorque)) > abs((self._lastTargetReading - self._targetTorque))):
    print("_calculateReward(): moving away from target Torque")
    self._localToTorque = False
...
# Change to negative if is going in opposite direction from target
if(False == self._localToTorque):
    self._localTorqueReward *= -1.0
```

REQ-5: Determine if the current BPM is converging into the target BPM.

```
self._localToBPM = True
if(abs((self._bpm - self._targetBPM)) > abs((self._lastBPMReading - self._targetBPM ))):
    print("_calculateReward(): moving away from target BPM")
    self._localToBPM = False
...
if(False == self._localToBPM):
    self._localBPMReward *= -1.0
```

REQ-6: Calculate the Torque reward.

```
self._localTorqueReward = (-1.0 * ((self._torque - self._targetTorque)**2)/200) + 10
```

REQ-7: Calculate the BPM reward.

IV DESIGN AND METHODOLOGY

```
self._localBPMReward = (-1 * ((self._bpm - self._targetBPM)**2)/303) + 20
```

REQ-8: If the Torque production is higher than the target Torque, divide the BPM reward by 4.

```
if (self._torque > self._targetTorque):  
    self._localBPMReward = self._localBPMReward / 4
```

REQ-9: Calculate the Total Reward.

```
self._localReward = (self._localBPMReward + self._localTorqueReward)
```

4.4.4 Reinforcement Learning Implementation

REQ-1: Set up the RL module constrains.

```
STATE_ALPHA = 0.2           # Alpha for Bellman equation  
STATE_GAMMA = 0.8          # Gamma for Bellman equation  
EPSILON = 0.1              # Epsilon for MDP exploration
```

REQ-2: Method to connect and disconnect with local sqlite3 DB.

```
def connectDB(self):  
    try:  
        # Open a connection with the server  
        self._conn = sqlite3.connect('tmegaloride_rl.db')  
  
        # Create a cursor  
        self._cursor = self._conn.cursor()  
    except sqlite3.Error as e:  
        # Print error  
        print(e)  
  
        # Return FAILURE  
        return(1)  
  
    # Return SUCCESS  
    return(0)  
  
def disconnectDB(self):  
    try:  
        # Close the cursor  
        self._cursor.close()  
  
        # Close the connection  
        self._conn.close()  
  
    except sqlite3.Error as e:  
        # Print error  
        print(e)  
  
        # Return FAILURE  
        return(1)
```

IV DESIGN AND METHODOLOGY

```
# Return SUCCESS  
return(0)
```

REQ-3: Method to set the environment.

```
def _setEnvironment(self, time, bpm, torque, assisting, stopSignal):  
    # Set environment values  
    self._time_t = time  
    self._bpm_t = bpm  
    self._torque_t = torque  
    self._stopSignal = stopSignal  
  
    if (1 == assisting):  
        if (False == self._onAssistTime):  
            self._criticalAssist = True  
            self._onAssistTime = True  
            self._assistCooldown = 0  
  
    return(0)
```

REQ-4: Method to get the current state value from DB.

```
def _getStateValue(self, bpm, torque):  
    self._localStateValueParams = list()    # List of parameters to locate the state value  
    self._localRecord = None                # Local query result buffer  
  
    print("_getStateValue(): Entering\n")  
  
    # Get the required search parameters  
    self._localStateValueParams = [bpm, torque]  
  
    # Execute the query  
    try:  
        self._cursor.execute(self.SQL_SEARCH_STATE_VAL, self._localStateValueParams)  
  
    except sqlite3.Error as e:  
        print(e)  
  
    # Fetch the result  
    self._localRecord = self._cursor.fetchone()  
  
    # Return the state-value  
    return (self._localRecord)
```

REQ-5: Method to set the current state value on the DB.

```
def _setStateValue(self, bpm, torque, value, counter):  
    self._localStateValueParams = list()    # List of parameters to locate the state value  
    self._localRecord = None                # Local query result buffer
```

IV DESIGN AND METHODOLOGY

```
# Get the required search parameters
self._localStateValueParams = [value, counter, bpm, torque]

# Execute the query
try:
    self._cursor.execute(self.SQL_UPDATE_STATE_VAL, self._localStateValueParams)
    self._conn.commit()
    return(0)
except sqlite3.Error as e:
    print(e)
    return(1)
```

REQ-6: Method to get the current Q value counter from DB.

```
def _getQStateValueCounter(self, bpm, torque, action):
    self._localActionName = "" # Buffer for the name of the action on the DB
    self._localStateValueParams = list() # List of parameters to locate the state value
    self._localRecord = None # Local query result buffer

    # Get the required search parameters
    try:
        # If action is ACTION_REDUCE
        if (self.ACTION_REDUCE == action):
            # Execute SQL_SEARCH_REDUCE_COUNTER query
            self._cursor.execute(self.SQL_SEARCH_REDUCE_COUNTER, self._localStateValueParams)

            #-----
            # Else if action is ACTION_SUSTAIN
            elif (self.ACTION_SUSTAIN == action):
                # Execute SQL_SEARCH_SUSTAIN_COUNTER query
                self._cursor.execute(self.SQL_SEARCH_SUSTAIN_COUNTER, self._localStateValueParams)
            #-----

            elif (self.ACTION_ASSIST == action):
                self._cursor.execute(self.SQL_SEARCH_ASSIST_COUNTER, self._localStateValueParams)
            #-----

            elif (self.HIGH_REDUCTION == action):
                self._cursor.execute(self.SQL_SEARCH_H_REDU_COUNTER, self._localStateValueParams)

            #-----
            elif (self.HIGH_ASSISTANCE == action):
                self._cursor.execute(self.SQL_SEARCH_H_ASSI_COUNTER, self._localStateValueParams)

        self._localRecord = self._cursor.fetchone()
        return(self._localRecord[0])
    except sqlite3.Error as e:
        print(e)
        print("_getQStateValueCounter(): Exiting with FAILURE\n")
        return("Error")
```

IV DESIGN AND METHODOLOGY

REQ-7: Method to set the current Q value from DB.

```
def _setQStateValue(self, bpm, torque, action, value, counter):
    self._localActionName = ""           # Buffer for the name of the action on the DB
    self._localStateValueParams = list() # List of parameters to locate the state value
    self._localRecord = None            # Local query result buffer

    # Get the required search parameters
    self._localStateValueParams = [value, counter, bpm, torque]
    try:
        if (self.ACTION_REDUCE == action):
            self._cursor.execute(self.SQL_UPDATE_Q_REDUCE_VAL, self._localStateValueParams)

        elif (self.ACTION_SUSTAIN == action):
            self._cursor.execute(self.SQL_UPDATE_Q_SUSTAIN_VAL, self._localStateValueParams)

        elif (self.ACTION_ASSIST == action):
            self._cursor.execute(self.SQL_UPDATE_Q_ASSIST_VAL, self._localStateValueParams)

        elif (self.HIGH_REDUCTION == action):
            self._cursor.execute(self.SQL_UPDATE_Q_H_REDU_VAL, self._localStateValueParams)

        elif (self.HIGH_ASSISTANCE == action):
            self._cursor.execute(self.SQL_UPDATE_Q_H_ASSI_VAL, self._localStateValueParams)

        self._conn.commit()
        print("_setQStateValue(): Exiting\n")
        return(0)
    except sqlite3.Error as e:
        print(e)
        print("_setQStateValue(): Exiting with FAILURE\n")
        return(1)
```

REQ-8: Method to get policy given the state (bpm and torque).

```
def _getPolicyAction(self, bpm, torque):
    # Local Attributes
    self._localStateValueParams = list() # List of parameters to locate the state value
    self._localPolicyRecord = None      # Local query result buffer
    self._localAction = None            # Best action available
    self._greedyApsilonProb = None      # Greedy epsilon probability roll
    self._localRandomAction = None      # Random action taken

    print("\n_getPolicyAction(): Entering\n")
```

IV DESIGN AND METHODOLOGY

```
# If assistance request, the action will be to assist
if (True == self._onAssistTime):
    # Return assist
    return(self.ACTION_ASSIST)

if (True == self.POLICY_OPTIM):
    self._epsilonGreedeProb = random.uniform(0, 1)
else:
    self._epsilonGreedeProb = 0.9

if (self.EPSILON >= self._epsilonGreedeProb ):
    self._localAction = random.randrange(self.ACTION_ACCOUNT)
    return(self._localAction)

# Get the required search parameters
self._localStateValueParams = [bpm, torque]

# Execute the query
try:
    self._cursor.execute(self.SQL_SEARCH_POLICY, self._localStateValueParams)
except sqlite3.Error as e:
    print(e)

# Fetch the result
self._localPolicyRecord = self._cursor.fetchone()

# Get the Action
self._localAction = self._localPolicyRecord[0]

print("\n_getPolicyAction(): Exiting\n")

# Return the policy Action
return (self._localAction)
```

REQ-9: Method to get Q policy given the state (bpm and torque) for greedy-epsilon approach.

```
def _getQPolicyAction(self, bpm, torque):
    # Local Attributes
    self._localStateValueParams = list()      # List of parameters to locate the state value
    self._localQRecord = list()              # List of Q State Values record
    self._localQStateValues = list()         # List of q State Values
    self._localMaxValues = list ()           # List of repeated Max values
    self._localQPolicyRecord = None          # Local query result buffer
```

IV DESIGN AND METHODOLOGY

```
self._localAction = None           # Best action available
self._greedyApsilonProb = None     # Greedy epsilon probability roll
self._localRandomAction = None     # Random action taken
self._maxActionValue = None        # Selected Max value
self._localCount = 0               # Iteration counter
self._localMaxBuffer = None        # Temporal buffer for max values

print("_getQPolicyAction(): Entering\n")

# If assistance required, the action will be to assist
if (True == self._onAssistTime):
    # Return assist
    return(self.HIGH_ASSISTANCE)

self._epsilonGreedeProb = random.uniform(0, 1)

if (self.EPSILON >= self._epsilonGreedeProb ):
    self._localAction = random.randrange(self.ACTION_ACCOUNT)
    print("_getQPolicyAction(): Exiting\n")
    return(self._localAction)

self._localQRecord = self._searchQActionValue( bpm, torque )

self._localQStateValues.append([self.ACTION_REDUCE, self._localQRecord[0]])
self._localQStateValues.append([self.ACTION_SUSTAIN, self._localQRecord[1]])
self._localQStateValues.append([self.ACTION_ASSIST, self._localQRecord[2]])
self._localQStateValues.append([self.HIGH_REDUCTION, self._localQRecord[3]])
self._localQStateValues.append([self.HIGH_ASSISTANCE, self._localQRecord[4]])

# Get the Maxarg from the Q(St-1, At-1)
self._maxActionValue = max(self._localQStateValues, key=lambda item:item[1])
self._localMaxValues.clear()

# Check if there are additional records with the same value (yes, it happens a lot)
for self._localMaxBuffer in (self._localQStateValues):
    if (self._localMaxBuffer[1] == self._maxActionValue[1]):
        self._localMaxValues.append(self._localMaxBuffer)

# If there are more than one values, choose it randomly
if (len(self._localMaxValues) > 1):
    self._localRandomAction = random.randrange(len(self._localMaxValues))
    self._maxActionValue = self._localMaxValues[self._localRandomAction]
```

IV DESIGN AND METHODOLOGY

```
self._localQPolicyRecord = self._maxActionValue[0]
print("_getPolicyAction(): Exiting\n")
```

```
# Return the policy Action
return (self._localQPolicyRecord)
```

REQ-10: Method with TD-0 prediction implementation given the current state.

```
def _td0prediction(self):
    # Local Attributes
    self._localCSVbuffer = None      # Local buffer for the CSV list
    self._localTorqueReward = True   # The torque has changed
    self._localBPMReward = True
    self._localToBPM = True          # The BPM changed towards target BPM
    self._localToTorque = True       # The Torque changed towards target Torque
    self._stateValueCounter = None   # The number of times the value has been calculated
    self._stateValueCopy = None      # Copy of the original State Value used later on for reward
                                        # Accumulation average

    # Update Value Function on time t-1
    # Since an action has been taken previously now, we can calculate the reward we got, also we
    # can search the value of the current state
    # Get the current state value and counter from the state_value table
    # Note, the counter is needed for the calculation of the reward accumulation average
    self._stateValue_t, self._stateValueCounter = self._getStateValue(self._bpm_t,
                                                                    self._torque_t)

    self._stateValueCopy = self._stateValue_t
    self._localTorqueReward = True   # The torque has changed
    self._localBPMReward = True
    self._localToBPM = True          # The BPM changed towards target BPM
    self._localToTorque = True       # The Torque changed towards target Torque

    # Get the reward for being at this state due have taken action_t-1
    if (True == self._criticalAssist):
        self._rewardCalc.setEnvStatus(self._bpm_t,
                                      self._torque_t,
                                      self._time_t,
                                      self.CRITICAL_REWARD)

        print("_td0prediction(): Critical assistance requested")
        self._criticalAssist = False

    elif ((self._assistCooldown < self.ASSIST_REQUEST_TIME) and (True == self._onAssistTime)):
        self._rewardCalc.setEnvStatus(self._bpm_t,
```

IV DESIGN AND METHODOLOGY

```
        self._torque_t,
        self._time_t,
        self.ASSITACE_REWARD)
    self._assistCooldown += 1

else:
    print("_td0prediction(): Regular reward requested")
    self._onAssistTime = False
    self._rewardCalc.setEnvStatus(self._bpm_t,
                                   self._torque_t,
                                   self._time_t,
                                   self.REGULAR_REWARD)
    self._reward_t = self._rewardCalc.getReward()

# With all the pieces we can calculate the state value for t-1 (Bellman Equation)
print ("self._stateValue_t_min1 = self._stateValue_t_min1 + alpha(self._reward_t + " +
        "gamma*(self._stateValue_t)- self._stateValue_t_min1)")

self._stateValue_t_min1 =( self._stateValue_t_min1 +
                             self.STATE_ALPHA *
                             (self._reward_t +
                              (self.STATE_GAMMA *
                               self._stateValue_t) -
                               self._stateValue_t_min1))
self._stateValue_t_min1 = round(self._stateValue_t_min1, 4)

# If the state we t-1 is exactly the same as t, we need to update the stateValue t to be the same as t-1
if((self._bpm_t == self._bpm_t_min1) and (self._torque_t == self._torque_t_min1) and
    (self._asistReq_t == self._asistReq_t_min1)):
    self._stateValue_t = self._stateValue_t_min1

# Use the reward accumulation average to normalize (sort off) the value
# Increase the State Value counter
self._stateValueCounter += 1
self._stateValue_t_min1 = (self._stateValueCopy +
                            (self._stateValue_t_min1 - self._stateValueCopy)/self._stateValueCounter)

self._stateValue_t_min1 = round(self._stateValue_t_min1,4)

# Now let's update the state value on the state_value Table
self._setStateValue(self._bpm_t_min1,
                    self._torque_t_min1,
                    self._stateValue_t_min1,
```


IV DESIGN AND METHODOLOGY

```
self._stateValueCounter)

# Get the _policy to follow
if (True == self.SARSA_Q_POLICY):
    self._policyAction_t = self._getQPolicyAction(self._bpm_t, self._torque_t)

else:
    self._policyAction_t = self._getPolicyAction(self._bpm_t, self._torque_t)

# t-1 is done for the function value, now let's call SARSA to update the policy
if (True == self.POLICY_OPTIM):
    self._sarsaControl()

# Finally, move all values from t to t-1
self._stateValue_t_min1 = self._stateValue_t
self._reward_t_min1 = self._reward_t
self._bpm_t_min1 = self._bpm_t
self._torque_t_min1 = self._torque_t
self._asistReq_t_min1 = self._asistReq_t
self._time_t_min1 = self._time_t
self._policyAction_t_min1 = self._policyAction_t
self._QActionValue_t_min1 = self._QActionValue_t

if (True == self.SAVE_CSV):
    self._localCSVbuffer= [self._time_t_min1,
                           self._bpm_t_min1,
                           self._torque_t_min1,
                           self._stateValue_t_min1,
                           self._QActionValue_t_min1,
                           self._policyAction_t_min1,
                           self._actionToName(self._policyAction_t_min1),
                           self._reward_t_min1]
    self._csvRecords.append(self._localCSVbuffer)

return([self._policyAction_t,self._reward_t])
```

REQ-11: Method with SARSA control implementation given the current state.

```
def _sarsaControl(self):
    # Local attributes
    self._localQValues = []
    self._localQCounter = 0          # Local buffer for action counter needed for reward accumu-
                                     # lation
```

IV DESIGN AND METHODOLOGY

```
self._qActionValCopy = None    # Local copy of the _QActionValue_t_min1 for reward accumu-
                              # lation

print("\n -----_sarsaControl(): Entering time" +
      "[{}] -----\n".format(self._time_t))

# Get the current Q(S,A) Value
self._localQValues = self._searchQActionValue(self._bpm_t, self._torque_t)
self._QActionValue_t = self._localQValues[ self._policyAction_t]

# Make a copy of _QActionValue_t_min1 for reward accumulation calculation
self._qActionValCopy = self._QActionValue_t_min1

# By this point we should have enough data to update the state-value from t-1 and then select
# a new policy for it
# Calculate the t-1 state-action value (Bellman Equation)
print ("self._QActionValue_t_min1 = self._QActionValue_t_min1 + alpha(self._reward_t + " +
      "gamma*(self._QActionValue_t)- self._QActionValue_t_min1)")

self._QActionValue_t_min1 = (self._QActionValue_t_min1 +
                             self.Q_ALPHA*(self._reward_t +
                                             (self.Q_GAMMA*self._QActionValue_t) -
                                             self._QActionValue_t_min1))
self._QActionValue_t_min1 = round(self._QActionValue_t_min1, 4)

# If the state we t-1 is exactly the same as t, we need to update the stateValue t to
# be the same as t-1
if((self._bpm_t == self._bpm_t_min1) and (self._torque_t == self._torque_t_min1) and
   (self._asistReq_t == self._asistReq_t_min1)):
    self._QActionValue_t = self._QActionValue_t_min1

# Use the reward accumulation average to normalize (sort off) the value
# Get the Action counter from t-1
self._localQCounter = self._getQStateValueCounter( self._bpm_t_min1,
                                                    self._torque_t_min1,
                                                    self._policyAction_t_min1)

# Increase the State Value counter
self._localQCounter += 1

# Apply the reward accumulation average
self._QActionValue_t_min1 = (self._qActionValCopy +
                             (self._QActionValue_t_min1 - self._qActionValCopy)/self._localQCounter)
```

IV DESIGN AND METHODOLOGY

```
self._QActionValue_t_min1 = round(self._QActionValue_t_min1, 4)

# Update the Q table for t-1
self._setQStateValue(self._bpm_t_min1,
                    self._torque_t_min1,
                    self._policyAction_t_min1,
                    self._QActionValue_t_min1,
                    self._localQCounter )

# Update Policy from t-1 with the new Q(S,A) Values
if (False == self.SARSA_Q_POLICY):
    self._updatePolicy( self._bpm_t_min1, self._torque_t_min1, self._asistReq_t_min1)

print("\n -----_sarsaControl(): Exiting ----- \n")
return(0)
```

REQ-12: Method to update policy, given a state and action taken.

```
def _updatePolicy(self, bpm, torque):
    # Local attributes
    self._qLocalRecord = None          # Local Buffer to keep the Action values from the Q table
    self._maxActionValue = None        # Holds the maximum value Action
    self._localActionValues = list()   # List of action-values tuples
    self._localPolicyParams = list()   # List of parameters for Policy update SQL

    print("_updatePolicy(): Entering \n")

    # Get the Action Values for the given State Actions
    self._qLocalRecord = self._searchQActionValue(bpm, torque)
    self._localActionValues.append([self.ACTION_REDUCE, self._qLocalRecord[0]])
    self._localActionValues.append([self.ACTION_SUSTAIN, self._qLocalRecord[1]])
    self._localActionValues.append([self.ACTION_ASSIST, self._qLocalRecord[2]])
    self._localActionValues.append([self.ACTION_ASSIST, self._qLocalRecord[3]])
    self._localActionValues.append([self.ACTION_ASSIST, self._qLocalRecord[4]])

    # Get the Maxarg from the Q(St-1, At-1)
    self._maxActionValue = max(self._localActionValues, key=lambda item:item[1])
    self._localPolicyParams = [self._maxActionValue[0], bpm, torque]

    # Execute the query
    try:
        self._cursor.execute(self.SQL_UPDATE_POLICY, self._localPolicyParams)
        print("_updatePolicy(): sql succesfull")
```

IV DESIGN AND METHODOLOGY

```
        self._conn.commit()
    except sqlite3.Error as e:
        print(e)
    return(0)
```

REQ-13: Method to update the state and provide the action to take.

```
def updateRLAgent(self, time, bpm, torque, assistReq, stopSignal):
    # Local variables
    self._localActionReward = None

    # Get the required parameters
    self._setEnvironment(time, bpm, torque, assistReq, stopSignal)

    # If the run has stopped, save if needed, and return no reward and no action
    if (True == stopSignal):
        return(self._stopRL())

    # If first run just determine the action and reward, this is to be considered t0
    if (True == self._firstRun):
        self._firstRun = False
        return (self._firstRunInit(time, bpm, torque, assistReq))

    #Execute TD(0) prediction
    self._localActionReward = self._td0prediction()

    # Return action
    print("updateRLAgent(): Exiting with self._localActionReward: " +
          "{}".format(self._localActionReward))
    return(self._localActionReward)
```

4.4.5 GUI Implementation

REQ-1: Set up the GUI Sections.

```
self._dataInputFrame = LabelFrame(self.master,
                                  text="Input Selector")

self._currentDataFrame = LabelFrame(self.master,
                                    text="Data Reading")

self._environmentFrame = LabelFrame(self.master,
                                    text="Environmet Status")
```

IV DESIGN AND METHODOLOGY

```
self._discountFrame = LabelFrame(self.master,
                                text="Reward discount scalars")

self._emulTorCtrlFrame = LabelFrame(self.master,
                                    text="Emulator \n Torque Control")

self._bpmLvlFrame = LabelFrame(self.master,
                                text="BPM Level")

self._assistLblFrame = LabelFrame(self.master,
                                   text="Assistance %")

self._histoFrame = LabelFrame(self.master,
                               text="RL Histogram")

# Labels
self.winfo_toplevel().title("TMegaloRide")

# Data reading section
self._currentTimeLbl = Label(self._currentDataFrame,
                             text="Time",
                             width=6,
                             anchor=CENTER)

self._currentTime = Label(self._currentDataFrame,
                           text="0",
                           width=6,
                           anchor=CENTER)

self._currentBPMLbl = Label(self._currentDataFrame,
                             text="BPM",
                             width=6,
                             anchor=CENTER)

self._currentTorqueLbl = Label(self._currentDataFrame,
                                text="Torque",
                                width=7,
                                anchor=CENTER)

self._currentBPM = Label(self._currentDataFrame,
                          text="0",
                          width=6,
                          anchor=CENTER)

self._currentTorque = Label(self._currentDataFrame,
                             text="0",
                             width=7,
                             anchor=CENTER)

# Environment section
self._timeEnvLbl = Label(self._environmentFrame,
                         text="Time",
                         width=6,
```

IV DESIGN AND METHODOLOGY

```
        anchor=CENTER)

self._timeEnv = Label(self._environmentFrame,
                      text="0",
                      width=6,
                      anchor=CENTER)

self._bpmEnvLbl = Label(self._environmentFrame,
                        text="BPM",
                        width=6,
                        anchor=CENTER)

self._bpmEnv = Label(self._environmentFrame,
                     text="0",
                     width=6,
                     anchor=CENTER)

self._torqueEnvLbl = Label(self._environmentFrame,
                            text="Torque",
                            width=6,
                            anchor=CENTER)

self._torqueEnv = Label(self._environmentFrame,
                        text="0",
                        width=6,
                        anchor=CENTER)

self._assistReqEnvLbl = Label(self._environmentFrame,
                               text="Assist",
                               width=7,
                               anchor=CENTER)

self._assistReqEnv = Label(self._environmentFrame,
                            text="0",
                            width=7,
                            anchor=CENTER)

self._rewardEnvLbl = Label(self._environmentFrame,
                            text="Reward",
                            width=7,
                            anchor=CENTER)

self._rewardLbl = Label(self._environmentFrame,
                        text="0",
                        width=7,
                        anchor=CENTER)

# Discount section
self._discTimeLbl = Label(self._discountFrame,
                           text="Time",
                           width=6,
                           anchor=CENTER)

self._discBPMLbl = Label(self._discountFrame,
```

IV DESIGN AND METHODOLOGY

```
        text="BPM",
        width=6,
        anchor=CENTER)

self._discTorqueLbl = Label(self._discountFrame,
                            text="Torque",
                            width=6,
                            anchor=CENTER)

self._discTimeEntry = Entry(self._discountFrame,
                             width=6)

self._discBPMEEntry = Entry(self._discountFrame,
                             width=6)

self._discTorqueEntry = Entry(self._discountFrame,
                               width=6)

# Histogram section
self._histoTime = Label(self._histoFrame,
                        text="Time,",
                        anchor=CENTER)

self._histoBpm = Label(self._histoFrame,
                       text="BPM,",
                       anchor=CENTER)

self._histoTorque = Label(self._histoFrame,
                          text="Torque,",
                          anchor=CENTER)

self._histoAssistReq = Label(self._histoFrame,
                             text="Assistance,",
                             anchor=CENTER)

self._histoReward = Label(self._histoFrame,
                           text="Reward,",
                           anchor=CENTER)

self._histoAction = Label(self._histoFrame,
                           text="Action ",
                           anchor=CENTER)

# Combo box
self._dataInputCBox = ttk.Combobox(self._dataInputFrame,
                                   values=["Emulator", "USB"],
                                   width=10)

self._dataInputCBox.current(0)

# Scale
self.scaleTorqueCtrl = Scale(self._emulTorCtrlFrame,
                             variable=self.emulTorCtrlLvl,
                             resolution=.1,
                             label="nm",
```

IV DESIGN AND METHODOLOGY

```
        from_=30,
        to=6,
        orient=VERTICAL,
        length=200)

# Progress Bar
self.bpmProgBar = ttk.Progressbar(self._bpmLvlFrame,
                                  orient=VERTICAL,
                                  length=200,
                                  mode='indeterminate'
                                  )

self.assistProgBar = ttk.Progressbar(self._assistLvlFrame,
                                     orient=VERTICAL,
                                     length=200,
                                     mode='determinate'
                                     )

# Text
# The width was set in order to match the size of the label headers
self.histoTxt = Text(self._histoFrame,
                     width=35,
                     height=5,
                     font=("Helvetica", 16))

# Buttons
self._timeReadBtn = Button(self._dataInputFrame,
                            text="Start reading",
                            width=14,
                            command=lambda:self._emulatorTimeMode())

self._stepReadBtn = Button(self._dataInputFrame,
                            text="Step reading",
                            width=14,
                            command=lambda:self.step())

self._stopReadBtn = Button(self._dataInputFrame,
                            text="Stop reading",
                            width=14,
                            command=lambda:self._stopDataReading(),
                            state=DISABLED)

self._setDiscBtn = Button(self._discountFrame,
                           text="Set Discounts",
                           width=14,
                           command=lambda:self._setDiscouts(),
                           state=NORMAL)

self._assistReqBtn = Button(self._emulTorCtrlFrame,
                             text="Request assistance",
                             width=14,
                             command=lambda:self._setAssistReq(),
                             state=NORMAL)
```


IV DESIGN AND METHODOLOGY

REQ-2: Import and create an instance of the emulator.

```
# Reward Module
self._rewardCalc = rewardModule.MasterReward() # Reward instance from tmegaloride_ai
```

REQ-3: Import and create an instance of the RL module.

```
# RL agent
self._agentRL = aiMaster.RL_Entity() # Artificial module for RL agent
```

REQ-4: Create a method to request assistance.

```
def _setAssistReq(self):
    # Set the Request assistance value to 1
    self._requestAssist = 1
    self._assistanceCount = self.ASISTANCE_TIME
    self._assisting = True

    # Disable the Request Assistance button
    self._assistReqBtn['state'] = DISABLED
    return(0)
```

REQ-5: Create a method to update the BPM level on the designated scale.

```
def _updateBPMLevel(self):
    # Local attributes
    self._localAdjBPM = 0 # Adjusted BPM value (CURRENT - MINIMUM)
    self._localMaximum = 0 # Maximum percentage (MAXIMUM - MINIMUM)
    self._localBPMPercent = 0 # Calculated percentage

    # Get the Maximum BPM adjusted
    self._localMaximum = self.MAXIMUM_BPM - self.MINIMUM_BPM

    # Adjust current BPM value
    self._localAdjBPM = self._BPM - self.MINIMUM_BPM

    # Calculate the BPM percent
    self._localBPMPercent = (self._localAdjBPM * 100) / self._localMaximum

    # Inver the value in order to use it on the progress bar
    self._localBPMPercent = (self._localBPMPercent - 100) * -1

    # Set the percentage on the progress bar
    self.bpmProgBar['value'] = int(self._localBPMPercent)
    return
```

IV DESIGN AND METHODOLOGY

REQ-6: Create a method to update the complete GUI.

```
def _updateGUI(self):
    # Local attributes
    self._localTextLine = None

    # Update Data Reading Section
    self._currentTime['text'] = "{}".format(self._time_step)
    self._currentBPM['text'] = "{}".format(self._BPM)
    self._currentTorque['text'] = "{}".format(round(self._Torque,2))
    self._rewardLbl['text'] = "{}".format(self._reward)

    # Update Histogram section
    self._histoTime = "{}".format(self._time_step)
    self._histoBpm = "{}".format(self._BPM)
    self._histoTorque = "{}".format(round(self._Torque,2))
    self._histoAssistReq = "{}".format(self._requestAssist)
    self._histoReward = "{}".format(self._reward)
    if (self.ACTION_REDUCE == self._actionTaken):
        self._histoAction = "Decrease"
    elif (self.ACTION_SUSTAIN == self._actionTaken):
        self._histoAction = "Sustain"
    elif (self.ACTION_ASSIST == self._actionTaken):
        self._histoAction = "Assist"
    elif (self.HIGH_REDUCTION == self._actionTaken):
        self._histoAction = "High Decrease"
    elif (self.HIGH_ASSISTANCE == self._actionTaken):
        self._histoAction = "High assistance"
    else:
        self._histoAction = "N/A"

    # Create a single line for the text widget
    self._localTextLine = "{}, {}, {}, {}, {}, {}, {} \n".format(self._histoTime,
                                                                    self._histoBpm,
                                                                    self._histoTorque,
                                                                    self._histoAssistReq,
                                                                    self._histoReward,
                                                                    self._actionTaken,
                                                                    self._histoAction)

    self.histoTxt.insert(END, self._localTextLine)
    self.histoTxt.see(END)
    self.histoTxt.update()
```

IV DESIGN AND METHODOLOGY

```
# Update the Environment Status section
self._timeEnv['text'] = "{}".format(self._time_step)
self._bpmEnv['text'] = "{}".format(self._BPM)
self._torqueEnv['text'] = "{}".format(round(self._Torque,2))
self._assistReqEnv['text'] = "{}".format(self._requestAssist)

self._updateBPMLevel()
return
```

REQ-7 Step mode method to execute the training on steps instead of time-based.

```
def step(self):
if (False == self._dbConected):
    self._agentRL.conectDB()

# requestBPMChange
# Make sure the time start is disabled
self._timeReadBtn['state'] = DISABLED

# Update the data input mode to step
self._dataInputMode = self.EMULATOR_STEP_MODE

# Make sure the stop button is enabled
self._stopReadBtn['state'] = NORMAL

# Increase current time/step value
self._updateEmulator()
return(0)
```

REQ-8: Emulator update Method to provide the latest changes on the Torque generation requirements.

```
def _updateEmulator(self):
    # Local attributes
    self._scaleTorqueTmp = self.MINIMUM_TORQUE
    self._lastScaleTorque = 0.0 # Buffer for last torque reading
    self._aiActionReward = None # Reward for the action taken by the AI

    # Increase current time/step value
    self._time_step += 1

    # If torque requirement changes
    # NOTE. this was expected to be executed directly on the scale widget, but it resulted in a
    # waste of time since it won't execute but once and that's it, useless so far...
```

IV DESIGN AND METHODOLOGY

```
self._scaleTorqueTmp = self.scaleTorqueCtrl.get()
if (self._lastScaleTorque != self._scaleTorqueTmp):
    # Update the torque value and request a BPM change to the emulator
    self._lastScaleTorque = self._scaleTorqueTmp
    self._Torque = self._scaleTorqueTmp
    self._emulator.requestBPMChange(self._scaleTorqueTmp )

# Execute the current timestep on the emulator
self._emulator.update(self._time_step)
self._BPM = int(self._emulator.getBPM())

# Update the data values and GUI
# Feed the IA agent
self._aiActionReward = self._agentRL.updateRLAgent(self._time_step,
                                                    self._BPM, round(self._Torque,1),
                                                    self._requestAssist,
                                                    False )
self._actionTaken = self._aiActionReward[0]
self._reward = self._aiActionReward[1]
self._updateGUI()

if (self.ACTION_ASSIST == self._actionTaken):
    if ((self._Torque >= 6.1)):
        print("updateEmulator(): Asisting")
        self._Torque = round(self._Torque - 0.1,2)

elif (self.ACTION_SUSTAIN == self._actionTaken):
    if (self._Torque <= 29.8):
        print("updateEmulator(): Sustain")
        self._Torque = self._Torque

elif (self.ACTION_REDUCE == self._actionTaken):
    if (self._Torque <= 29.8):
        print("updateEmulator(): Reducing")
        self._Torque = round(self._Torque + 0.1, 2)

elif (self.HIGH_REDUCTION == self._actionTaken):
    if (self._Torque <= 29.7):
        print("updateEmulator(): High Reduction")
        self._Torque = round(self._Torque + 0.2, 2)

elif (self.HIGH_ASSISTANCE == self._actionTaken):
```

IV DESIGN AND METHODOLOGY

```
if (self._Torque >= 6.2):
    print("updateEmulator(): High assistance")
    self._Torque = round(self._Torque - 0.2, 2)
else:
    print("\nupdateEmulator(): UNKNOWN ACTION TAKEN!!!![{}]\n".format(self._actionTaken))

self.scaleTorqueCtrl.set(round(self._Torque,1))

if (True == self.TIMER):
    self._timerCounter += 1

    # If timer goal completed
    if (self.TIMER_GOAL <= self._timerCounter):
        self._timerCounter = 0

        # Send the stop signal to the RL agent
        self._aiActionReward = self._agentRL.updateRLAgent(self._time_step,
                                                            self._BPM,
                                                            round(self._Torque,1),
                                                            self._requestAssist,
                                                            True )

        self._stopDataReading()
        print("_updateEmulator(): TIMED RUN COMPLETED!!!")

    # Reset any Assistance request
    if ((True == self._assisting)):
        self._assistanceCount += 1

    if ((self._assistanceCount >= self.ASISTANCE_TIME)):
        self._requestAssist = 0
        self._assisting = False
        self._assistReqBtn['state'] = NORMAL

return(0)
```

REQ-9: Emulator separate thread method for time based training.

```
def _emulatorThreadMethod(self, stopSignal):
    if (False == self._dbConected):
        self._agentRL.conectDB()
    while True:

        # Request a modulator update
```

IV DESIGN AND METHODOLOGY

```
self._updateEmulator()

# If a stop signal has been issued
if (True == self._stopEmulatorThread):
    self._time_step = 0
    self._BPM = 0
    self._Torque = 6
    self._agentRL.disconnectDB()
    break

# Wait before continue
time.sleep(0.3)
```

REQ-10: Emulator time-based mode Method.

```
def _emulatorTimeMode(self):
    if (False == self._dbConected):
        self._agentRL.conectDB()

    # Make sure the time start is disabled
    # We don't want multiple threads messing around
    self._timeReadBtn['state'] = DISABLED

    # Make sure the step start is disabled
    self._stepReadBtn['state'] = DISABLED

    # Update the data input mode to time mode
    self._dataInputMode = self.EMULATOR_TIME_MODE

    # Make sure the stop button is enabled
    self._stopReadBtn['state'] = NORMAL

    # Create a new Thread
    self._stopEmulatorThread = False
    self._emulatorThread = threading.Thread(target=self._emulatorThreadMethod,
                                           args=(lambda:self._stopEmulatorThread,),
                                           daemon=True)

    # Execute thread contained calculations and updates
    self._emulatorThread.start()
    return(0)
```

IV DESIGN AND METHODOLOGY

REQ-11: Method to stop the data reading and the agent training.

```
def _stopDataReading(self):
    if(self._dataInputMode == self.EMULATOR_TIME_MODE):
        # Set stop signal to True
        self._stopEmulatorThread = True

    # Join the thread so the signal its catch
    # Reset attributes
    # Reset emulator
    self._emulator.resetEnv()

    # Enable start buttons
    self._timeReadBtn['state'] = NORMAL

    # Make sure the step start is disabled
    self._stepReadBtn['state'] = NORMAL

    # Make sure the stop button is disabled
    self._stopReadBtn['state'] = DISABLED

    # Update the GUI
    self._time_step = 0
    self._BPM = 0
    self._Torque = 6
    self._agentRL.disconnectDB()
    self._updateGUI()
    return(0)
```

REQ-12: Method to update Torque generation from the designated scale.

```
def updateTorqueFromScale(self):
    self._Torque = self.scaleTorqueCtrl.get()
    self._emulator.requestBPMChange(self._Torque)
    self._updateGUI()
    return(0)
```

4.4.6 Modified Bicycle for data acquisition

REQ-1: Adapt a generic mountain bicycle for sampling, image 33.

IV DESIGN AND METHODOLOGY



Figure 4.15. Test bicycle with the photoplethysmography sensor (green box) Torque sensor (yellow box) and magnetic resistance trainer (red box) to create different torque requirements.

4.5 Testing

The project will be divided into two different areas for testing. The first area will be the data acquisition-related tests to perform the data analysis required for the emulator and initial policy creation. The second area will be the RL Agent performance on different levels of learning stages.

4.5.1. Data acquisition

Test subject supervised sampling. A sampling session will be performed on an individual volunteer who must perform a minimum of 3 sets of data sample collection. On each sampling, an aerobic fitness test will be performed, each sample will have a minimum of 3 minutes, and this will stop at the discretion of the test individual when he considers that the physical effort leaves his particular comfort area.

- Stages (It should be noted that the time and torque values described here are part of the research itself, this is in absentia²² of reliable bibliographic sources that provide reference values):
 - 100 seconds minimum of exercise with a torque of 13.5 N.m. (speed 7).
 - 25-minute break.

²² Latin for absence.

IV DESIGN AND METHODOLOGY

- 100 seconds minimum of exercise with a torque of 15.5 N.m. (speed 6).
- 25-minute break.
- 100 seconds minimum of exercise with a torque of 18 N.m. (speed 5).
- Analyze the data. The sampled data will be exported and analyzed on a spreadsheet program. BPM and Torque graphs will be created as reference.
- Initial policy creation. Once the samples are obtained under different orders of effort, the prototype policy will be created. This policy will represent a human picked set of actions that would be considered good behavior. This policy will take into consideration the test subject feedback and the results of the sampling analysis. Furthermore, it will be considered the baseline to compare the results of the RL performance.

4.5.2 RL Agent performance

Human-made policy evaluation. The policy created with the data acquired will be put to the test to use as a reference for the RL Agent performance. This test aims to determine if the agent is learning by converging on this Human-made policy.

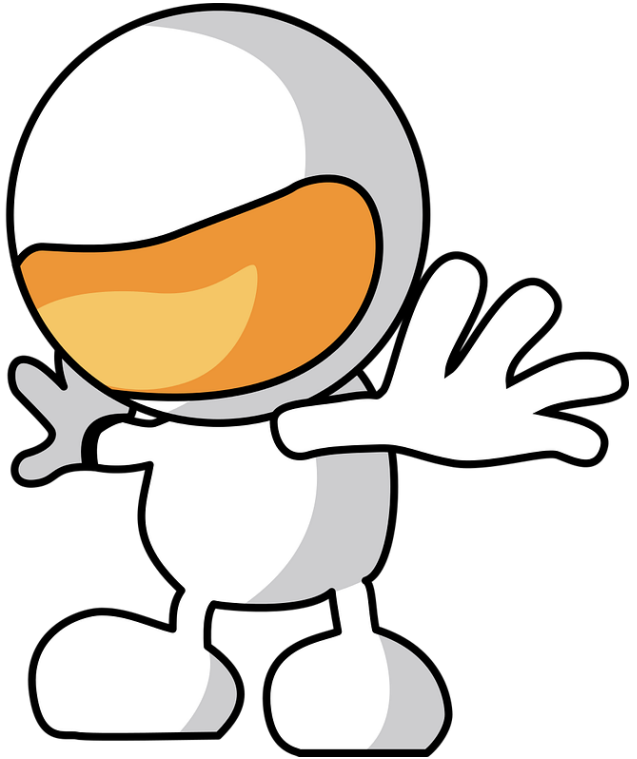
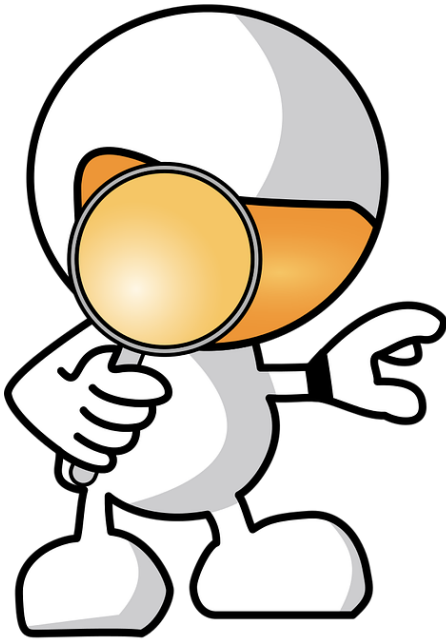
First training phase. Experiment with an initial requisition of 6.0N.m and no experience. It is expected to see how the AI increases or decreases the torque requirement based on trial and error until it stabilizes the test subject at 135 BPM. The total run will be of 2400 seconds. The entire run will consist of 2400 seconds/steps.

Second training phase. Experiment with an initial requisition of 6.0N.m and a previous experience of 2400 seconds (first phase). It is expected to see a rapid convergence towards the ideal values.

Third training phase. Experiment with an initial requisition of 6.0N.m and an experience of 4800 seconds (first and second phase). It is expected to see an even faster convergence towards the ideal values.

V FINDINGS AND DISCUSSION.

V FINDINGS AND DISCUSSION.



V FINDINGS AND DISCUSSION.

5.1 Data acquisition Results

Test subject supervised sampling.

100 seconds minimum of exercise with a torque of 13.5 N.m. (speed 7).

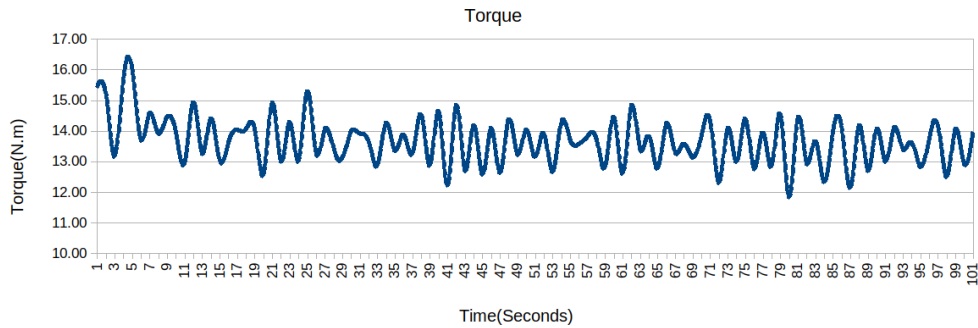


Figure 5.1. Torque generation at 13.5 N.m average.

Figure 5.1 contains a graph with the results of the supervised sampling taken from the torque sensor at an average of 13.5 N.m. The signal has a quasi-sinusoidal shape due to the fact that the pedals used for the test were the classic models. The configuration of these pedals means that only about 50% of the torque is produced continuously. This due to the force applied to the pedal is limited to (approx.) 160° (clockwise) forward motion of the bicycle crank. There is a gap of time where both legs transfer the momentum between each other, which leads to a fast decay in torque production.

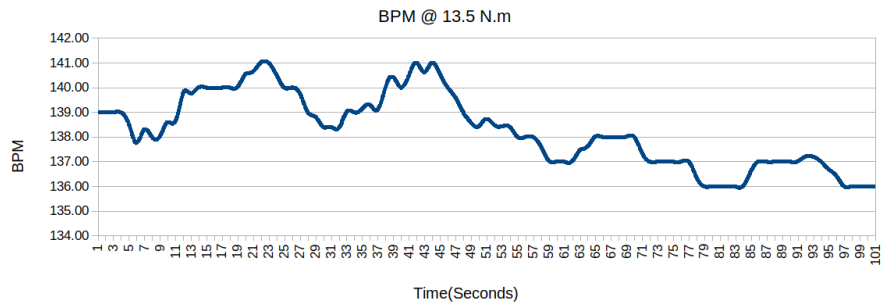


Figure 5.2. BPM response at 13.5 N.m

V FINDINGS AND DISCUSSION.

Figure 5.2 Shows the BPM response for the 13.5 N.m requirement. Although this is the initial (and lowest) torque requirement, it showed an unexpectedly (relative) high BPM initial value. This behavior was due to the torque control required from the bicycle mechanical gearing system, and this means that in order to achieve the torque requirement, it was required to change to shift number 7, which is the “easy/first” gear (the torque requirement from motion is inverse to the number of the selected shift). The principle of the gear system is to trade crank motion speed for torque, therefore, the pedals have to be moved faster (with a relatively small force applied) to achieve the desired torque on the back wheel. This trade-off took a slight cardio increase until the torque production was stabilized.

100 seconds minimum of exercise with a torque of 15.5 N.m. (speed 6).

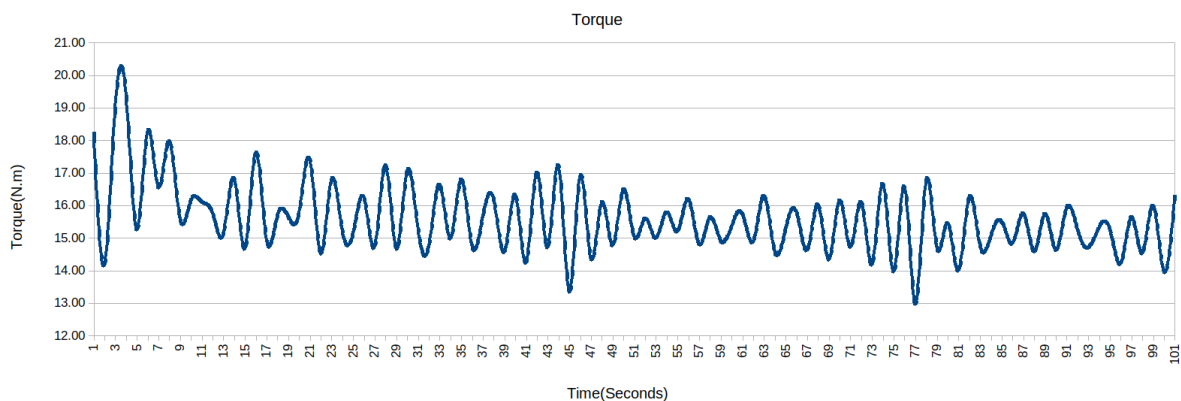


Figure 5.3. Torque generation at 15.5 N.m average.

Figure 5.3 shows the graph of the 15.5 N.m data acquisition. As before, the data shows the quasi-sinusoidal behavior. The initial readings (first 7 seconds) show a transient that is to be expected due to the need to break the inertia. This extra torque will introduce some bias, yet, it is a desired behavior since it will provide a more realistic emulation and a posterior benefit to the RL Agent in transient changes.

V FINDINGS AND DISCUSSION.

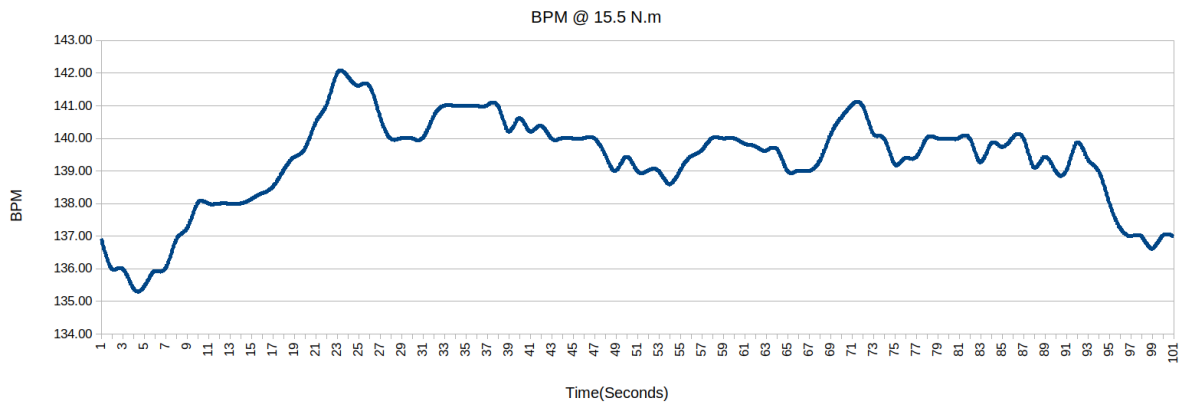


Figure 5.4. BPM response at 13.5 N.m

In figure 5.4, the heartbeat response can be seen, and as expected, the initial inertia breathing has some consequence on the readings but again, beneficial further on. The BPM shows a steady increase until the torque production stabilizes, then it shows some steadiness. This data already proves helpful in a quick view since the heartbeat shows clear patterns so far.

100 seconds minimum of exercise with a torque of 18 N.m. (speed 5).

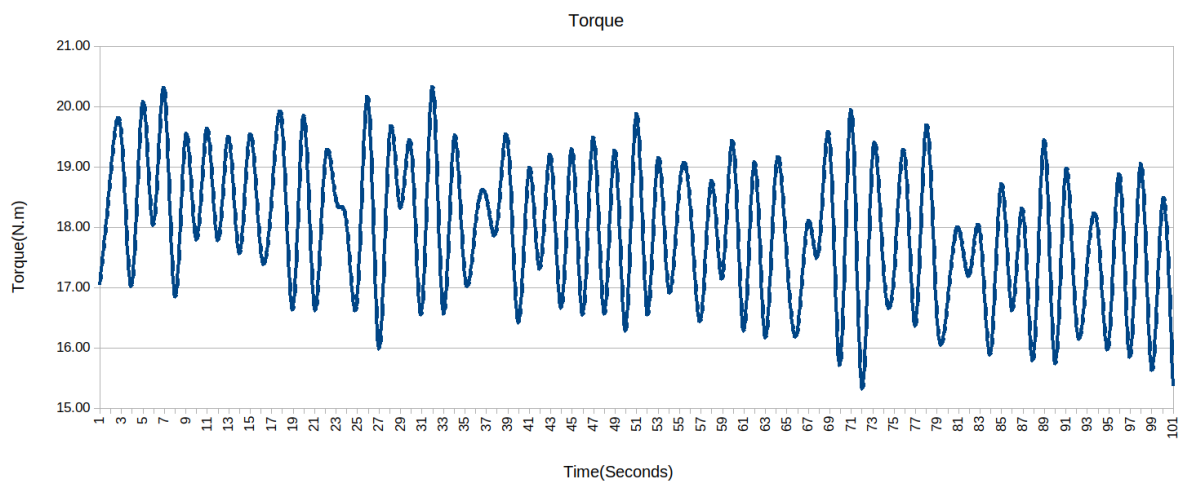


Figure 5.5. Torque generation at 18 N.m average.

Figure 5.5 shows the graph of the data acquisition for the 18 N.m Torque production. As stated before, the need for the bicycle mechanical gear system that trades speed for torque and vice-

V FINDINGS AND DISCUSSION.

versa, in this case, it is easy to see a clear even trade-off since the quasi-sinusoidal shape almost match each push of the bicycle crank, each valley shows the transition between the pushing leg.

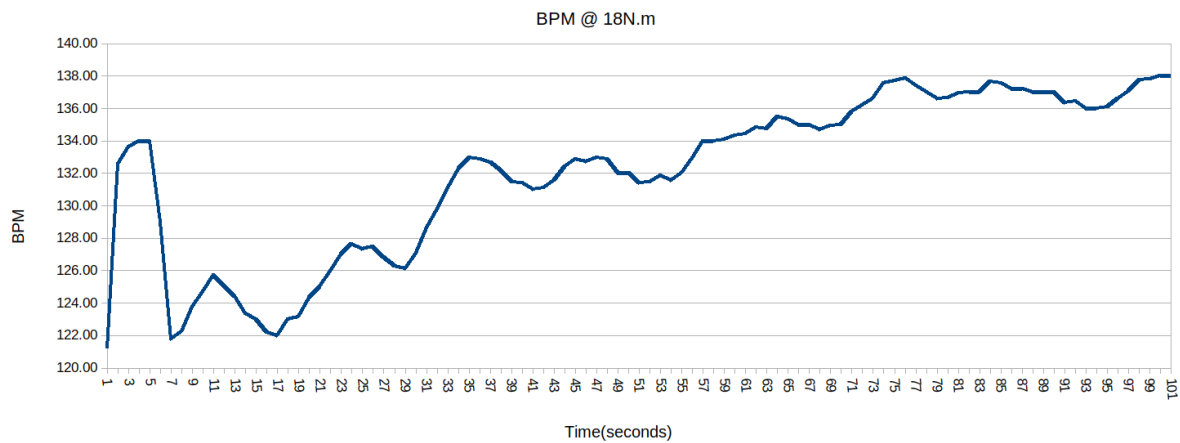


Figure 5.6. BPM response at 18 N.m

Finally, in figure 5.6 the heartbeat response can be appreciated. As before, an initial high BPM effort is required to break the initial inertia. After that, an increase of BPM is observed going steadily and barely stabilizing at the last 30 seconds, meaning that this Torque production requirement is on the edge of the test subject fiscal capabilities.

Initial policy creation

A test subject was used on the modified bicycle, and the "comfortable" torque output for that subject was determined to be 17 N.m, with a heart rate of 135 BPM (± 5 BPM). It should be noted that interviewing was required to arrive at these particular values, which rules out the values as "universal". However, these metrics are of vital importance to be able to compare later the results of the RL Agent against something considered "fitting" (at least for the test subject).

V FINDINGS AND DISCUSSION.

5.2 RL Agent performance results

Human-made policy evaluation

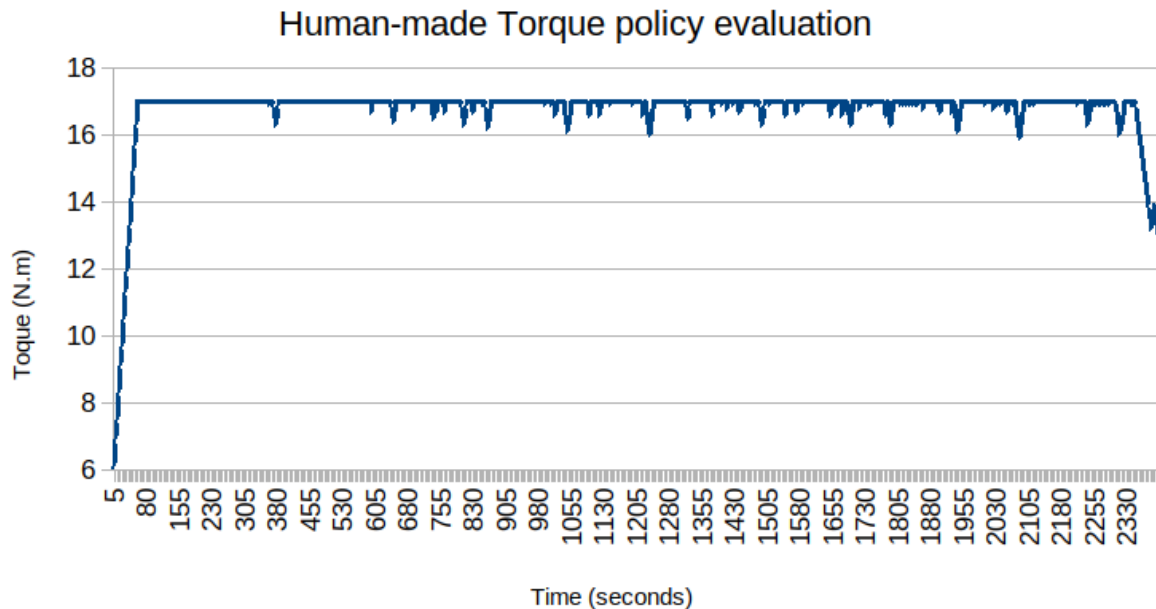


Figure 5.7. Human-made Torque policy evaluation.

Figure 5.7 shows the performance of the Human-Made Policy, since it is premade to keep the Torque production and BPM within specific preconceived ranges (17 N.m, with a heart rate of 135 ± 5 BPM), it has a pretty linear response since the policy is simple and there is no need to figure it out anything. It's worth noting how static behavior can prove not to be 100% bulletproof.

At the end of the run, there is an unexpected transient showing that the policy encounters great difficulties maintaining the ideal values for torque production. This will be expanded further on the BPM response graph.

V FINDINGS AND DISCUSSION.

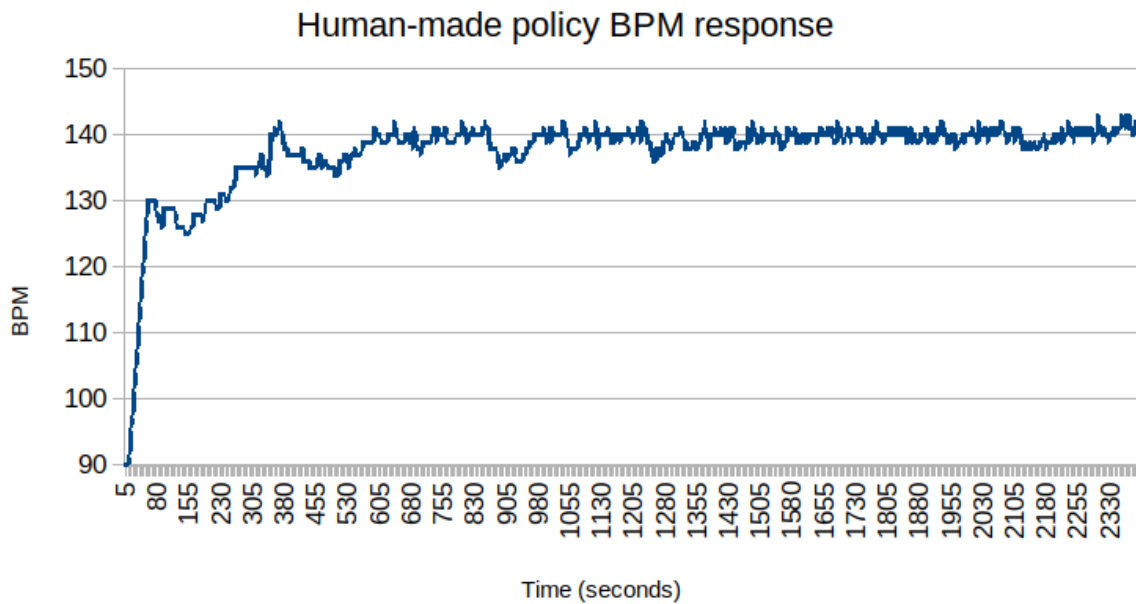


Figure 5.8. Human-made policy BPM response.

Figure 5.8 shows the BPM response following the Human-made policy. At first the behavior goes as expected, a steady BPM increase due the policy increases to until matching the desired Torque production level and then sustain as long the BPM keeps in the desired range of 135 ± 5 BPM. This approach leads to a behavior where the policy only assists when the 140 BPM threshold is exceeded.

This approach leads to a fight to quickly assist in order to return to the wanted BPM range, but it has a hidden risk, the exhaustion plays a vital role in this scenario, keeping the BPM on the upper hand all the time means that the exhaustion will eventually overcome the policy, and this means that the policy cannot keep with the exhaustion rate even if it applies the maximum available assist per second. In other words, what starts as a well-defined policy, becomes a greedy one with time until it is too greedy and becomes overwhelmed.

V FINDINGS AND DISCUSSION.

First training phase

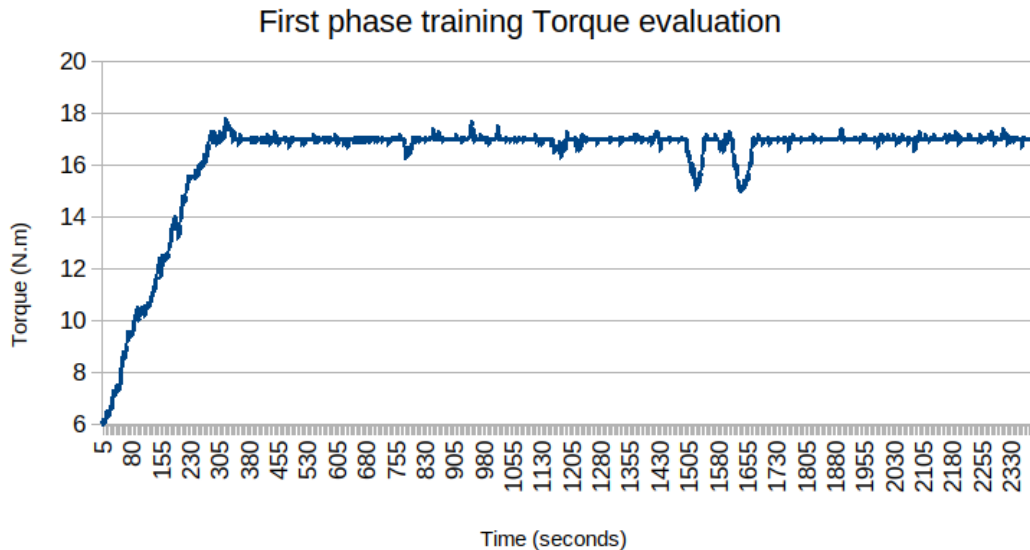


Figure 5.9. First training phase Torque evaluation.

Figure 5.9 shows the first training set of the RL Agent. As expected, the first 300 seconds show the trial-and-error trade-off of the RL approach. This trade-off is from the lack of experience from the Agent. In other words, the Agent does not know what to do at each different step/state and relies on the randomness entirely.

The downside of the randomness is represented on all the “assist” and “maintain” actions during the first 300 steps. This is clearly not expected since the torque requisition is far from the (unknown to the Agent) 17 N.m. Here, the reward is a crucial section that gives the proper direction to the Agent trying to maximize the reward. After the 300 steps mark, it becomes clear that the maximum reward has been reached and the Agent (with some errors) learns to stay there for the rest of the run.

V FINDINGS AND DISCUSSION.

First training phase BPM response

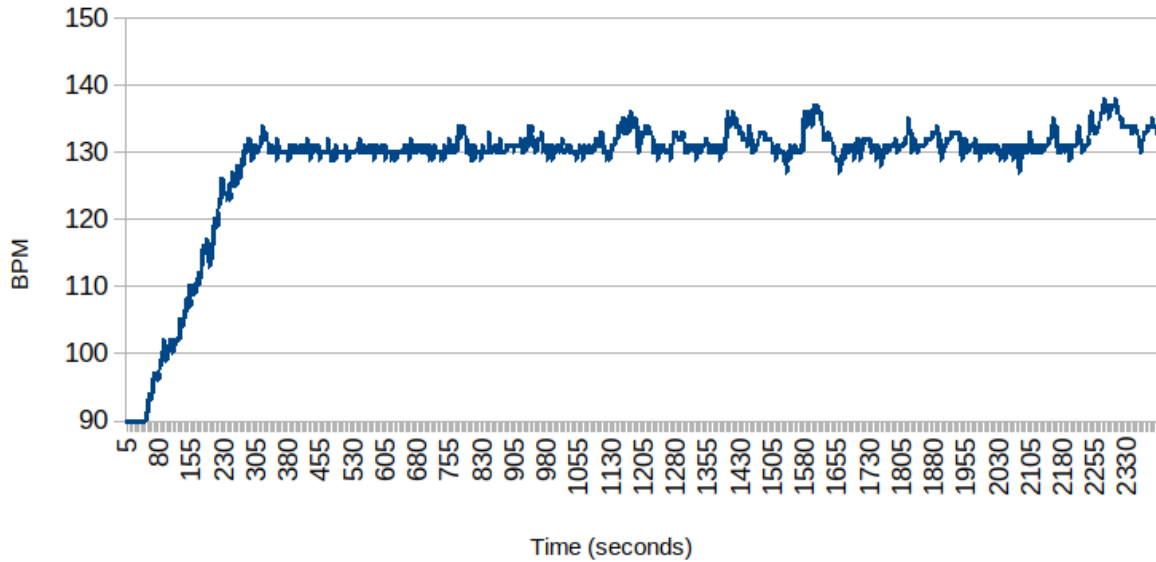


Figure 5.10. First training phase BPM response.

The BPM response (figure 5.10) to the first Agent learning run does matches the behavior of the policy constructed on the fly. One important aspect is that the several “errors” and the unknown optimal values keep the BPM on the lower expected range, the opposite from the Human-made policy. As stated before, the Reward module is the responsible at this state for this behavior since the “increase Torque production” and the “Sustain” actions have somehow similar values.

V FINDINGS AND DISCUSSION.
Second training phase

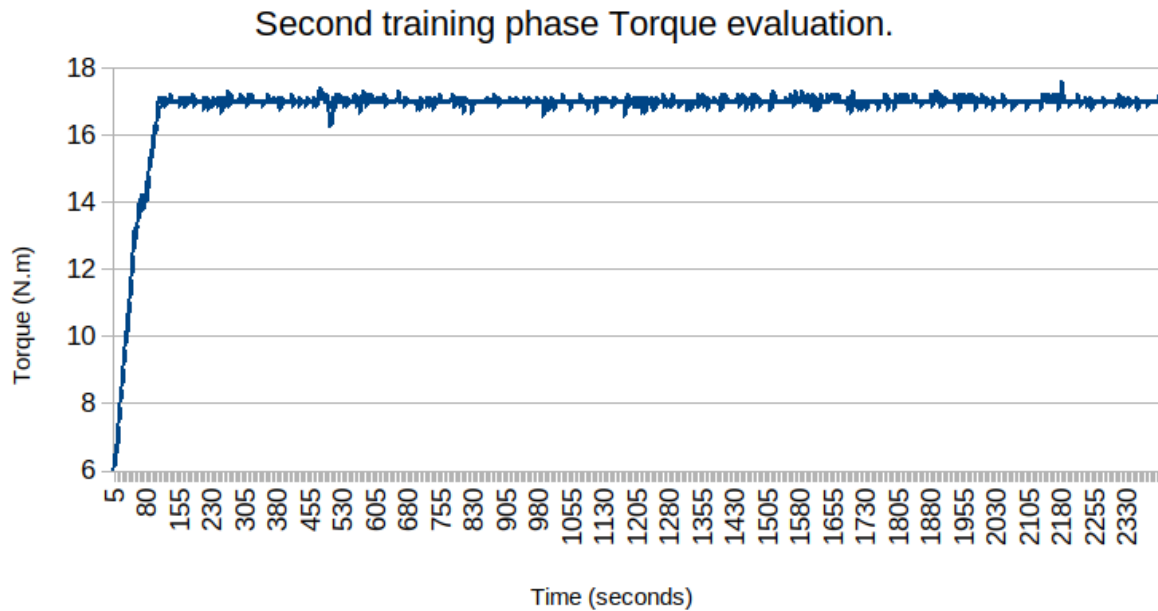


Figure 5.11. Second training phase Torque evaluation.

For the second training run, the performance (figure 5.11) has a remarkable performance increment. This is because there is a basic experience acquired, which means that the first policy generated at least tells the Agent what actions were bad, so it tries some other action the following ϵ -greedy method instead of pure randomness. It still has room for improvement since eliminating one of the actions per state does not grant that a better option can be taken next time.

V FINDINGS AND DISCUSSION.

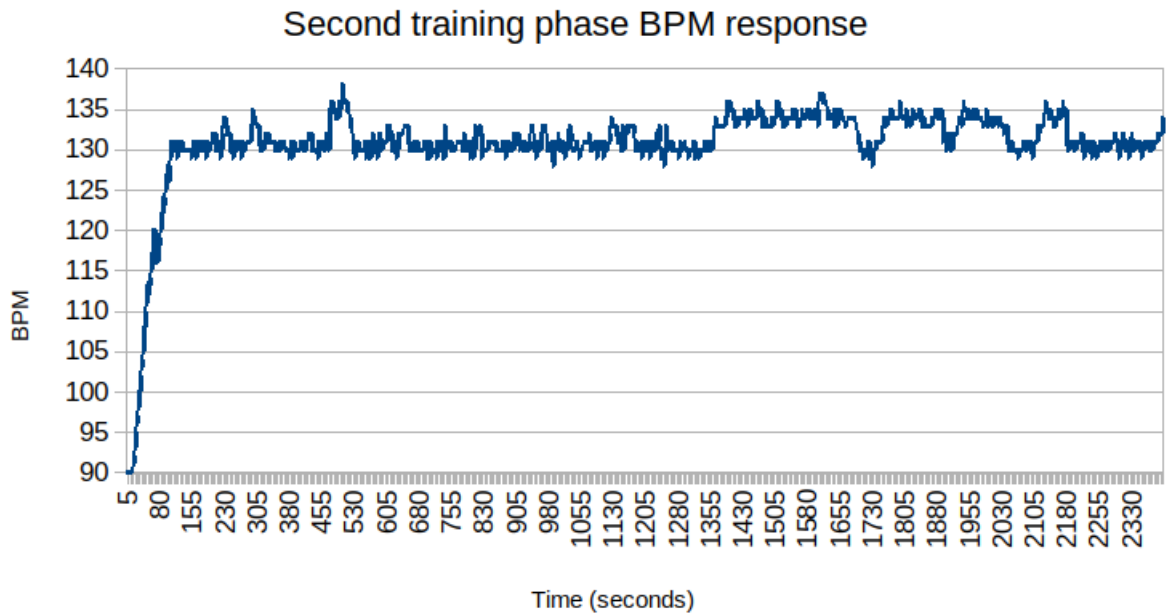


Figure 5.12. Second training phase BPM response.

The BPM response for the second training room for the Agent (figure 45) also shows quite some improvement. The BPM does not stay on the lower threshold; it actually increases to 135 BPM, especially at the end of the run. This behavior is again gained thanks to the ϵ -greedy method, that from time to time ($<\epsilon$), tries a random action that proved (with a bit of luck) to be beneficial.

V FINDINGS AND DISCUSSION.

Third training phase

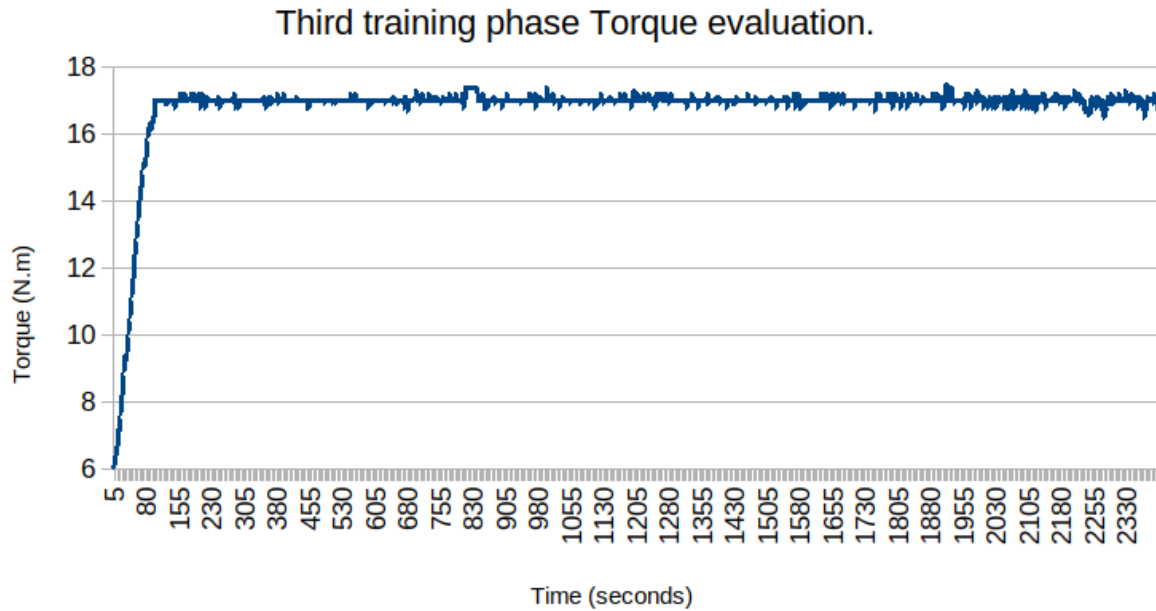


Figure 5.13. Third training phase Torque evaluation.

The third learning run (figure 5.13) shows a less dramatic improvement, yet, it has some interesting features in the small details. Mostly due ϵ -greedy method trying new actions from time to time, there is an interesting response on the stability on the 17 N.m Torque requirement. Instead of staying as close to the goal, it shows much variance, almost noise at the end of the run. This causes a fascinating result on the BPM response shown below.

V FINDINGS AND DISCUSSION.

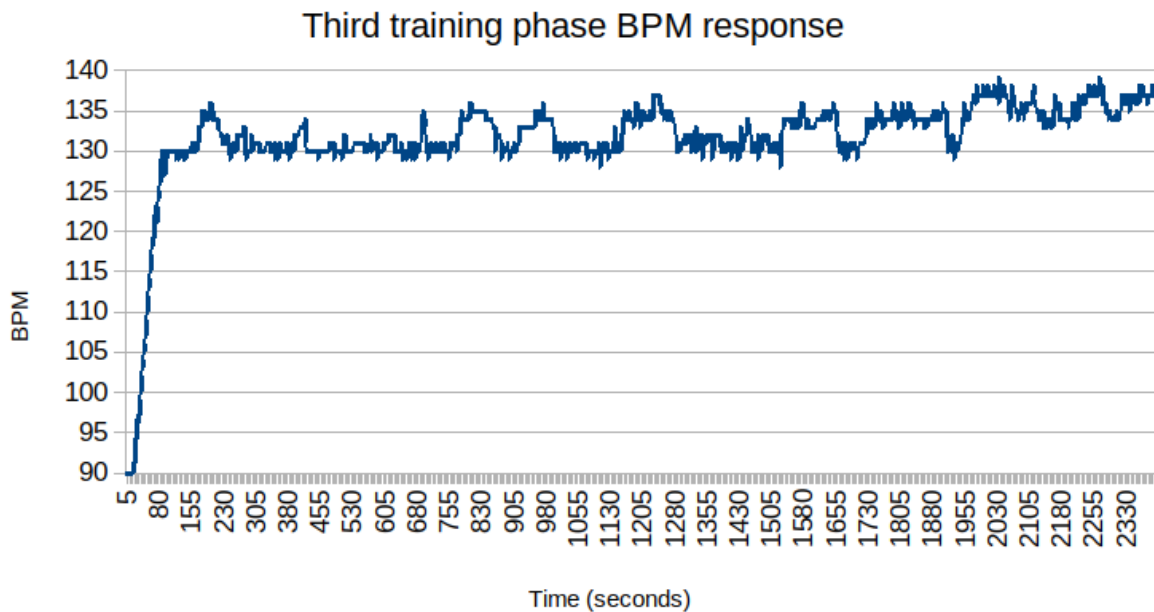


Figure 5.14. Third training phase BPM response.

Finally, the BPM response (figure 5.14) for the third training run shows the results of the noisy Torque policy. Instead of keeping the Torque at the desired level, it changes it to increase and decrease the BPM closer to 135 BPM for short periods, yet it brings it back right away, avoiding the issue of the Human-Made Policy. This Agent policy learns to keep the BPM as close to the target while avoiding the upper threshold.

V FINDINGS AND DISCUSSION. Comparative

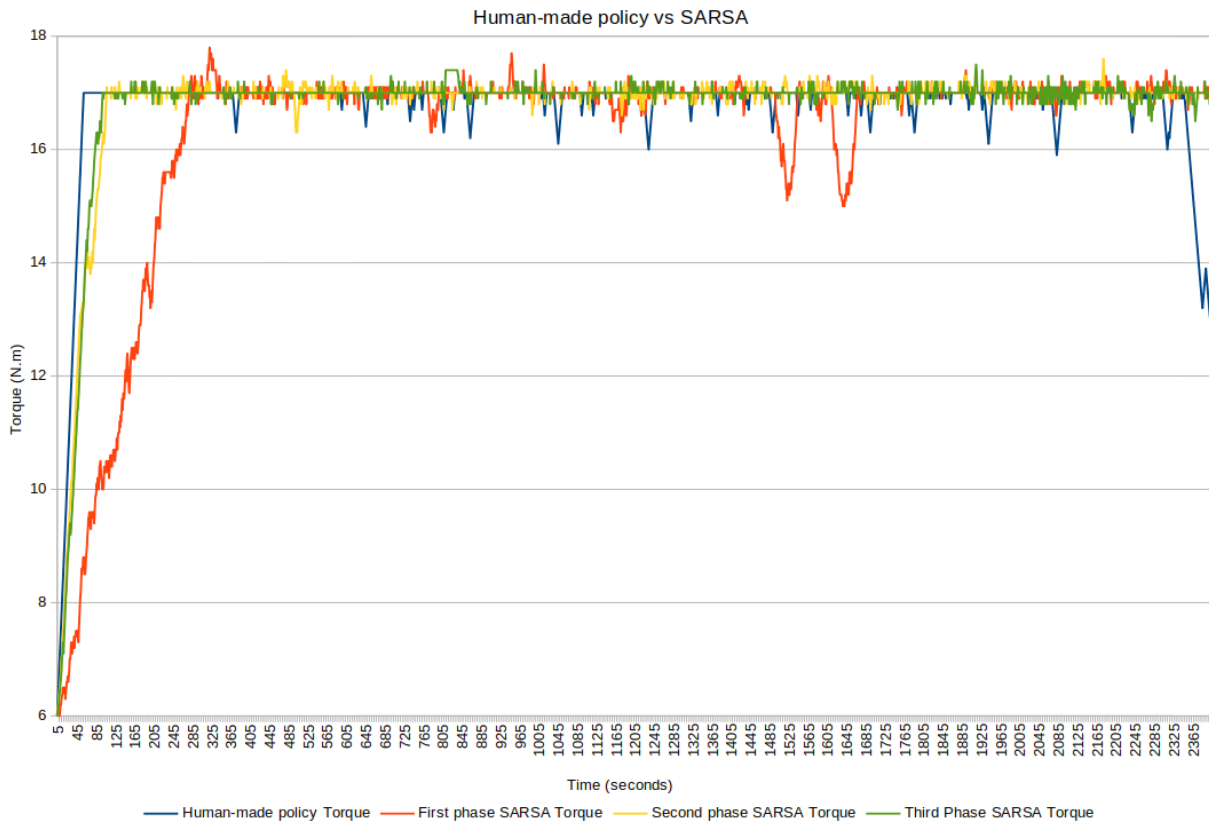


Figure 5.15. Human-made policy vs SARSA performance comparative.

The results in the torque control are shown in figure 5.15. The Human-made policy is represented in blue, a fairly linear response given that the desired values are clearly stated. The red line represents the behavior of the artificial agent while learning from scratch. It is observed that it required a much longer time than the static policy to achieve the objective (70 vs. 342 seconds). This performance is explained by the fact that, in the beginning, the agent is making "trial and error" decisions.

Then we have the second experiment in yellow. Here an exceptional improvement can be seen. Previous experience is reflected with a faster convergence (70 vs. 106 seconds).

V FINDINGS AND DISCUSSION.

Finally, the third run is displayed in green. In this case, the speed in convergence does not differ much from the previous one (106 vs. 99 seconds). However, it can be seen that throughout the experiment, this is the one that presents the most outstanding flexibility in the target area (17 N.m), even better than static policy.

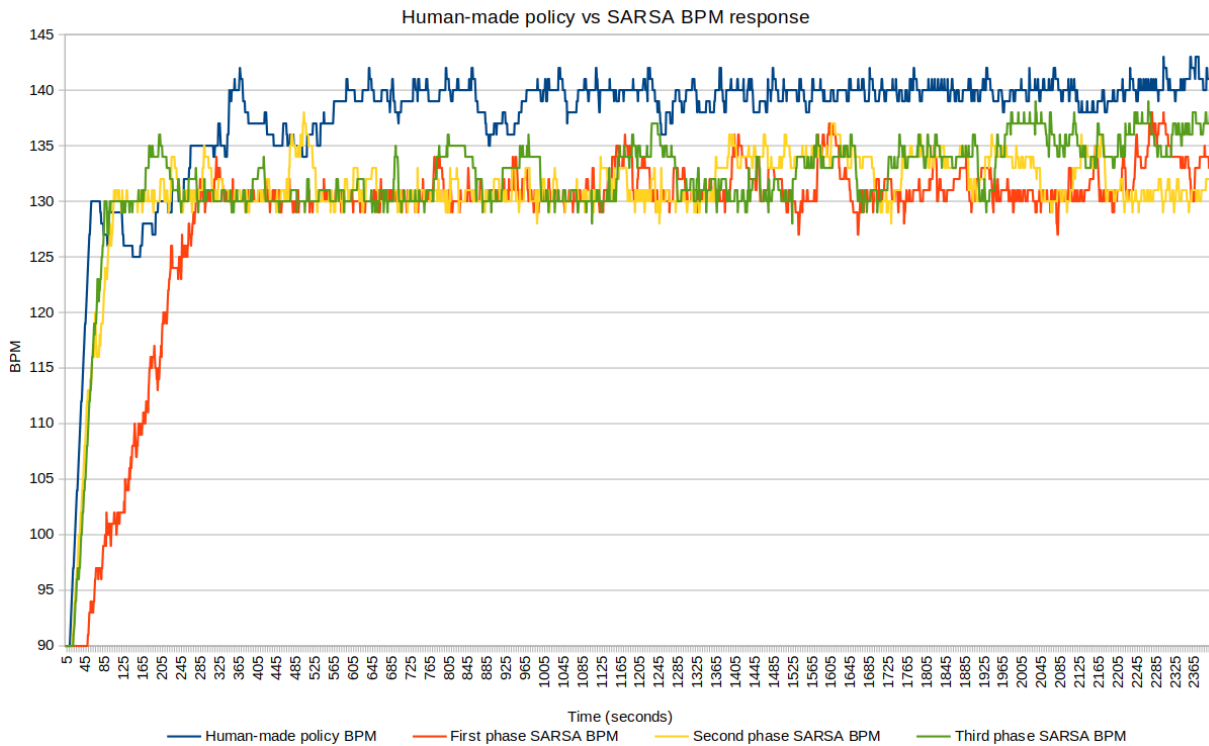


Figure 5.16. Human-made policy vs SARSA BPM response.

Finally, figure 5.16 shows the most important results from the experiments, comparing the RL Agent vs. the Human-made policy. As stated previously, the third run proved to be the most successful due to the consistent approach to the target 135 BPM.

This last run proved that the bias from the test subject indeed influences the decision making. The reason for this is the different times needed to stabilize the BPM when a Torque production has been reached.

The third policy shows some faint resemblance to the quasi-sinusoidal behavior from the data acquisition sections, affecting the BPM stabilization values.

V FINDINGS AND DISCUSSION.

5.3 Results Discussion

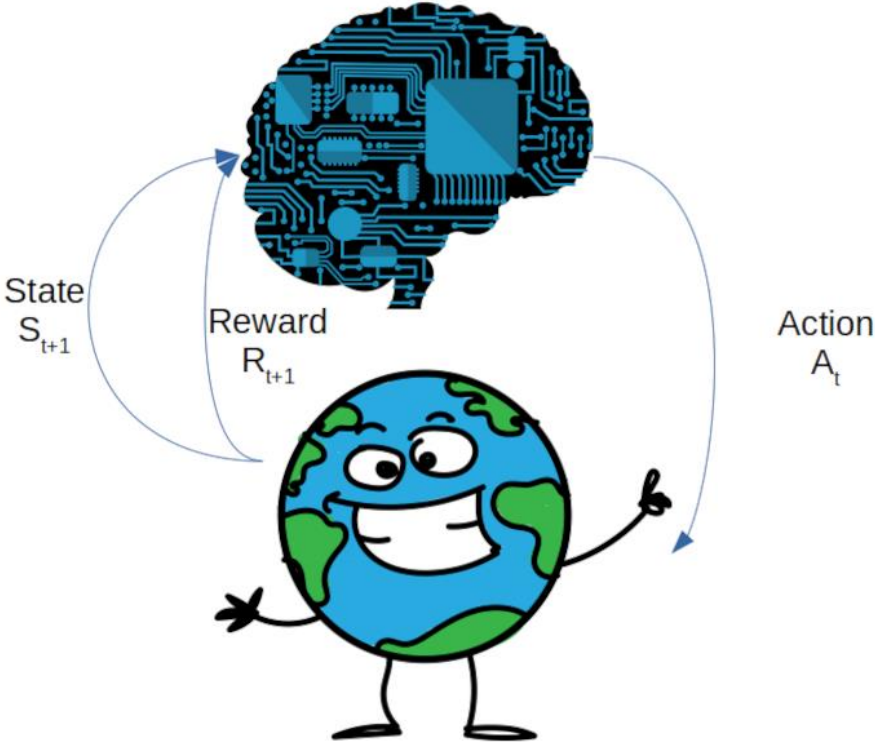
The results corroborate the effectiveness of the combination of gadgets and RL AI. Analyzing the data obtained in more detail, we can notice some unique characteristics: RL is capable of learning as it is used, regardless of the lack of specific instruction. Little by little, the desired convergence is reached.

The results are consistent with a laborious and meticulously fabricated policy by contrast. If the test subject were changed at any given time, it would take about 30-40 minutes to re-train, assuming that the physical capabilities of the new subject are entirely different (for example, a 20-year-old and a 60-year-old adult). This feature can be considered a massive advantage over other AI systems where a total re-training is necessary for each test subject, requiring hours or even days.

The use of two inexpensive commercial gadgets was enough to achieve the desired objective. AI can be implemented in a practical way without having to resort to unaffordable cost projects.

CONCLUSION

CONCLUSION



CONCLUSION

Thanks to technological advances, the availability on the market of various easily integrated sensors/gadgets, development boards, and micro-portable computers, it has been possible to demonstrate how the RL AI can improve the quality, flexibility, and advantages offered by Pedelec vehicles by including a novel autonomous assistance selection system governed by the cyclist's physical condition. Among the most significant advantages are the following:

The RL will always be looking for greater rewards, and nothing is set in stone. If the physical capacity of the cyclist changes at any given time, either due to constant use of the vehicle, gradually improving his physical qualities, or some adverse event such as illness or injury, the RL may be adjusted as many times as necessary on an autonomous behavior, this thanks to the “request assistance” user interface that provides an insight to the RL IA that something bad is going on with the cyclist, and therefore, take actions to improve assistance. While in the other hand the epsilon-greedy will be continuously “challenging” the cyclist (in a very subtly manner) to keep going as much as possible to the target Torque production level, as long this does not triggers a BPM increase.

Recognizing the physical condition of each cyclist will no longer imply a colossal task. Instead, it will only be continuous learning for each case (around 40 minutes in extreme cases).

Future work

Although the results are satisfactory, there are still several areas for future research and improvements to the established concept, thus consolidating a more robust and flexible system. In the first instance, incorporating additional sensors that allow the AI to have a broader vision of the environment will be considered. Some commercial sensors that will be studied are:

Cadence sensor to obtain the number of revolutions per minute on the pedals. During the data-gathering section, there was a hint of a relationship between the cadence (RPM) and the BPM

CONCLUSION

(especially during the first seconds). This value has the potential to increase the initial state's policy required to break the bicycle inertia.

Inclination sensor. The Inclination value will enhance the environment by letting the Agent know about the slope of the route's terrain. The hypothesis is that the torque production levels should diminish or halt on downhills since there is a risk of accelerating and achieving a velocity above the allowed forcing the need for unnecessary use of the brake system.

Speed sensor. Although the speed limit is ruled by an HW system, including this value in the RL environment can (hypothetically) prevent over-speed on a smooth behavior. As proven in the Human-made policy vs. Sarsa section, a static value does not grant the optimal performance.

Approximation functions addition. One of the downsides of the RL is the overwhelming increase of states while adding additional environment elements. This issue is a particular "deal-breaker" when it comes to environments that require time/step as one of the elements on a continuous non-ending task since it would mean an infinite number of states. Approximation functions allow creating a neural network to store the states, even if they tend to infinite, by providing an "approximation" of the state value functions and Q values, then the Agent works on the new values and it "backpropagate" them into the Neural Network, thus, overcoming the infinite state issue.

Tiling. The final consideration for the project is to change the second-by-second analysis and clustering the data into packages using the Tiling theories. This tiling has the potential of saving computing time, which is always a desired feature.

BIBLIOGRAPHY

BIBLIOGRAPHY

Allen J., 2007 Photoplethysmography and its application in clinical physiological measurement *Physiol. Meas.* 28 R1–39

Altini M., Penders J. and Amft O. Estimating oxygen uptake during nonsteady-state activities and transitions using wearable sensors. *IEEE J Biomed Health Inform* 20: 469 –475, 2016. doi:10.1109/JBHI.2015.2390493.

Ballantine, R., (2001). *Richard's 21st Century Bicycle Book*. New York: Overlook Press. pp. 28–39. ISBN 1-58567-112-6.

Beltrame T., Amelard R., Villar R, Shafiee MJ, Wong A. and Hughson R. Estimating oxygen uptake and energy expenditure during treadmill walking by neural network analysis of easy-to-obtain inputs. *J Appl Physiol* (1985) 121: 1226–1233, 2016. doi:10.1152/jappphysiol.00600.2016.

Beltrame T., Amelard R., Wong A. and Hughson R. Prediction of oxygen uptake dynamics by machine learning analysis of wearable sensors during activities of daily living. *Sci Rep* 7:45738, 2017. doi:10.1038/srep45738.

Beltrame T., Amelard R., Wong A. and Hughson R., 2017. Emerging Wearable Physiological Monitoring Technologies & Decision Aids for Health & Performance. *J Appl Physiol* doi:10.1152/jappphysiol.00299.2017

Beltrame T., Hughson R. Linear and non-linear contributions to oxygen transport and utilization during moderate random exercise in humans. *Exp Physiol* 102: 563–577, 2017. doi:10.1113/EP086145.

Breiman L. Random forests. *Mach Learn* 45: 5–32, 2001. doi:10.1023/A:1010933404324.

CFE, Contratación de Servicios Fotovoltaicos -
<https://www.cfe.mx/Casa/InformacionCliente/Pages/Contrataci%C3%B3n-de-servicios-Fotovoltaicos.aspx> Retrieved 4 June 2019.

BIBLIOGRAPHY

Cozzi L., 2019. Commentary: Growing preference for SUVs challenges emissions reductions in passenger car market, recovered in 2019 from: <https://www.iea.org/newsroom/news/2019/october/growing-preference-for-suvs-challenges-emissions-reductions-in-passenger-car-mark.html>

Heitmann J. The Automobile and American Life. McFarland, 2009, ISBN 0-7864-4013-9, pp. 11Ff.

INEGI, INEGI. XII Censo General de Población y Vivienda 2000, INEGI. II Conteo de Población y Vivienda 2005 y INEGI. Censo de Población y Vivienda 2010. 2019 in November 2019 from:

https://www.inegi.org.mx/app/tabulados/interactivos/default?px=Poblacion_01&bd=Poblacion

INEGI, Módulo de Práctica Deportiva y Ejercicio Físico (MOPRADEF). Extraído el 18 de Noviembre del 2019 de la pagina: <https://www.inegi.org.mx/programas/moprade/>

Herlihy, David V. (2004). Bicycle: The History. Yale University Press. ISBN 978-0-300-12047-9.

Kang S. and Parameswaran R. (1994). "Real-time computing: a new discipline of computer science and engineering". Proceedings of the IEEE. 82 (1): 6–24. CiteSeerX 10.1.1.252.3947. doi:10.1109/5.259423. ISSN 0018-9219.

Kerlin T. Frequency Response Testing in Nuclear Reactors. New York: Academic Press, 1974.

Moré J., 1978 The Levenberg-Marquardt algorithm: Implementation and theory Numerical Analysis (Lecture Notes in Mathematics vol. 630) ed G A Watson (Springer Verlag) pp 105–16

Monasterio V., Burgess F. and Clifford G., 2012 Robust neonatal apnoea-related desaturation classification *Physiol. Meas.* Accepted for publication, 2012.

Sutton R. and Barto A., 2019. Reinforcement Learning: An Introduction. ISBN 9780262039246.

BIBLIOGRAPHY

Staudenmayer J., Poher D., Crouter S., Bassett D. and Freedson P. An artificial neural network to estimate physical activity energy expenditure and identify physical activity type from an accelerometer. *J Appl Physiol* (1985) 107: 1300–1307, 2009. doi:10.1152/jappphysiol.00465.2009.

Villar R., Beltrame T. and Hughson R. Validation of the Hexoskin wearable vest during lying, sitting, standing, and walking activities. *Appl Physiol Nutr Metab* 40: 1019 –1024, 2015. doi:10.1139/apnm-2015-0140.

Waller D. 1887. A Demonstration on Man of Electromotive Changes accompanying the Heart's Beat. *The Physiological Society* <https://doi.org/10.1113/jphysiol.1887.sp000257>.

Shoham Y., Perrault R., Brynjolfsson E., Clark J., Manyika J., Niebles J., Lyons T., Etchemendy J., Grosz B. and Bauer Z., "The AI Index 2018 Annual Report", AI Index Steering Committee, Human-Centered AI Initiative, Stanford University, Stanford, CA, December 2018.