Application of a genetic algorithm to optimise parameter values of a PID controller

2047284 1910584 2048520

***Abstract*—PID controllers are used extensively in the field of robotics, enabling the user to achieve precise and smooth motion control, which has led to their extensive application over assembly lines. This report proposes an efficient and automated method for the tuning of a PID speed controller on the Pololu 3pi+ robot through the implementation of a genetic algorithm (GA). Experiments with a constant speed demand show that the GA is able to tune a PID controller in under 10 generations. Following this, the same algorithm is applied to a system with a changing demand where it is also found that a PID controller can be tuned to a sufficient threshold in under 10 generations.**

I. Introduction

In the field of robotics, achieving optimal control system performance is crucial, particularly in applications involving complex, dynamic environments. The Pololu 3pi+ robot [1] serves as an excellent case study for such challenges due to its agility and versatility in tasks ranging from line following to advanced maze navigation. The Proportional-Integral-Derivative (PID) controller [2], a staple in robotic control, is prized for its straightforward design and effective handling of various robotic behaviors. However, the task of tuning PID parameters is fraught with difficulties as these parameters must be meticulously adjusted to suit the nonlinear dynamics and changing conditions typical of robotic operations. Traditional manual tuning methods are not only labor-intensive but often fall short of achieving optimal performance due to the intricate interplay between a robot's control system and its operational environment. The necessity for precision and adaptability in robotic control underscores the importance of advanced tuning methodologies for PID controllers. For the Pololu 3pi+ robot, the controller's effectiveness in real-time adjustments of steering and speed is paramount to its performance.

Genetic algorithm's (GA's) were first theorised in the 1960's by John Holland and his students at the University of Michigan [3]. Contrasting work at the time, which focused on evolution strategies and evolutionary programming, the goal of their research was to study the phenomenon of adaptation in nature, and it's potential applications to computer systems. In essence genetic algorithms are a search and optimization algorithm, based off mechanisms observed in biology. The algorithms take advantage of genetic make-up and natural selection processes to improve population performance across large state-spaces. Furthermore, through the implementation of a mutation aspect, the GA becomes an incredibly effective method of finding optimal solutions in spaces containing many local maxima/minima. Although effective, GA's don't guarantee finding optimum solutions, relying on multiple searches and trials, and the tuning of its own parameters, including population size, number of generations, mutation rates etc.

In the field of robotics, GA's have been utilized in several areas. One example includes autonomous navigation, as seen in the paper by Manikas, Ashenayi and Wainwright [4]. Researches were able to construct chromosomal representations of a robots' possible paths in uncertain environments. Using population evolution and an appropriate fitness function, the agent is able to form optimal traversal of the area, including object avoidance and distance minimisation. Simulated results suggest genetic algorithms as a viable option for solving large state space problems such as path finding. Another example is seen in the report by Levitin, Rubinovitz and Shnits on robotic assembly line balancing (RALB) [5]. The paper introduces the problems involved with RALB, including optimal assignment for robots in line stations in order to maximise production rate. The researchers were able to solve these issues using genetic algorithms, testing said procedures on a range of randomly generated problems.

This report proposes an alternative method for the tuning of parameters in a PID controller, used for wheel torque modulation on the 3pi+ robot. Previous methods rely on human trial and error for the tuning of parameters; this is computationally inefficient and requires pre-requisite knowledge of appropriate parameter requirements for the context of the task. The proposed alternative relies on the implementation of a genetic algorithm, used to search the solution space for the optimum values of each parameter.

A. Hypothesis Statement

During preliminary studies, it was found that PID's took a large amount of time and prior knowledge of the system to tune the parameters to acceptable values. Additionally, all three components have to be adjusted in sync, due to the implicit affect of each parameter on one another. Manual methods of trial and error begin with the tuning of a single parameter, followed by sequential calibration of the remaining variables, occasionally revisiting previous parameters for further adjustment. This presents significant inefficiency and complexity to the user when searching for optimal solutions. Even with expert knowledge of compensating known errors, such as overshooting and settling times, this process is still time consuming.

Issues surrounding manual PID tuning in such contexts can be described as an optimisation problem. Although acceptable solutions can be reached, this report aims to investigate whether the introduction of a genetic algorithm allows for a universally applicable and efficient method for finding optimal parameter values for PID controllers.

A GA will allow us to search a wide solution space, along with the potential detection of multiple local maxima. From this, the following hypotheses are proposed:

***H1**: Given a reasonable range of values that the initial population can be drawn from, a PID controller for straight line wheel speed for the Pololu 3pi+ robot can be tuned in 10 generations.*

***H2**: H1 holds when the robot begins from rest to achieve its demand and when the direction shifts to the immediate reverse.*

It should be noted that the preliminary hypothesis, H1, is based on a robot that is lifted off the ground, e,g on a friction-less surface.

The hypotheses are investigated through a variety of experiments carried out on the 3Pi+ robot, including constant and switching demands, as well as the translation of an optimal solution to a PID controller.

The report proceeds as follows: Section II discusses the implementation of the genetic algorithm, along with how controller performance evaluation is defined. Section III outlines the experiments used to test the GA, including the definition of parameter values. In section IV, the findings of the experiments are displayed, followed by section V where the outcomes and hypothesis assessment are discussed concerning the implementation and testing of the GA.

II. Implementation

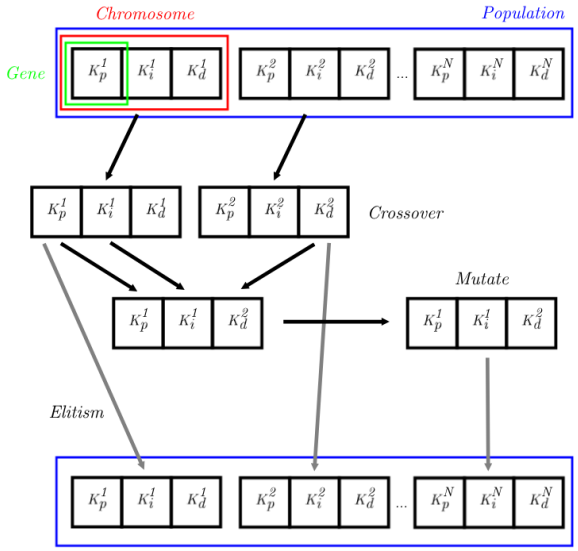

Fig. 1: Individual, population and evolution formation, including crossover, mutation and elitism. The figure highlights the process of generating a new population from the previous using the outlined techniques.

A. Genetic Algorithm

The genetic algorithm is constructed from a series of individuals forming a population (or generation), as seen in figure 1. Each individual has its own chromosomal make-up, where genes are the equivalent PID terms, K_p, K_i and K_d. Upon initialisation, individual genes are randomly assigned based on a uniform distribution between the desired state space ranges being explored.

The next stage involves iteratively testing each individual in the generation for a specified run time, then evaluating performance against a pre-defined fitness function. This report defines its fitness functions as a combination of overshoot and settling time against the required demand. Overshoot calculates the maximum absolute error captured above the settle tolerance threshold. Settling time refers to the final time the individual maintains its performance within the desired threshold tolerance for the remainder of the run. For example, if an individual achieves a performance within the demand threshold (i.e. $\pm 10\%$) in 5 seconds, its settling time will be 5 seconds. However, if the performance deteriorates outside of this range, the settling time resets to the total run time, unless the performance returns within the threshold, giving a new settling time. If the individual produces any values over 10% above the demand, the largest absolute difference is recorded as the overshoot error. A weighted average of both metrics creates the final fitness score, further normalised between 0-1, with smaller fitnesses showing greater performances.

Algorithm 1 Genetic Algorithm one trial pseudocode for automatic PID tuning on the Pololu 3pi+ robot.

Input: PopSize, PID value ranges, numTop, maxGen, rateMutate, RunTime
Output: Fitness evaluation, PID parameters

$P \leftarrow$ PopSize
$K_p \leftarrow$ range(p_{low}, p_{high})
$K_i \leftarrow$ range(i_{low}, i_{high})
$K_d \leftarrow$ range(d_{low}, d_{high})
for $i \leftarrow 0$ to P **do** ▷ initialise random population
    pop[i].PID = [rand(K_p), rand(K_i), rand(K_d)]
end for
curGen $\leftarrow 0$
while curGen $<$ maxGen **do**
    for $i \leftarrow 0$ to P **do** ▷ Evaluate individual fitness
        start = millis()
        while time $<$ RunTime **do**
            pop[i].run ▷ Run PID on motor demand
            time = millis() - start
        end while
        pop[i].fitness = F(pop[i]) ▷ F: Fitness function
    end for
    pop = sort(pop **by** fitness **from** low **to** high)
    for $j \leftarrow 0$ to numTop **do** ▷ Elitism
        newGen[j] = pop[j]
    end for
    for $k \leftarrow$ numTop to P **do**
        P_1, P_2 = rand(range(0, NumTop - 1))
    ▷ Choose two random parents from top performers
        newGen[k] = crossover(pop[P_1], pop[P_2])
        newGen[k] = mutate(newGen[k], rateMutate)
    end for
    pop = newGen
end while

Once a population has been fully evaluated, the individuals are sorted from best to worst. A sub-set of the top performing individuals is then selected as the basis for the next generation, known as elitism. Elitism ensures the top performers are not lost by copying the best individuals directly into the the new population. The remaining individuals are formed through random crossover

and mutation on this sub-set. Each new individual requires two randomly selected parents and crossover point. This point determines which PID values are passed on to the child from the parents. Values up to the crossover are taken from parent 1, whilst the remaining are taken from parent 2. Once the new chromosome is formed, each value has a probability of mutating to a randomised value within their original respective state space boundaries. This allows for the potential location of multiple local optima, rather than focusing purely on the local, current best performers. Figure 1 illustrates a high-level overview of the genetic algorithm sub-processes.

Finally, once the new population has been initialised, the testing process restarts, repeating for the desired number of generation. The genetic algorithm should be run for multiple trials, where a single trial represents the evaluation of all individuals for all desired generation.

III. Experiment Methodology

Algorithm 1 illustrates the genetic algorithm procedure to be implemented on the Pololu 3pi+ robot for the tuning of its PID controller. The robot is turned upside-down such that the experiment may be ran on a single floating wheel, i.e. without surface resistance, repeated for two different scenarios; constant and changing demand. Constant demand allocates a set, fixed wheel speed for the controller to maintain for the full duration of the run. Changing demand will use the same wheel speed, however, halfway through will change the desired direction of rotation.

Each experiment is ran with the following parameters:

- Population Size, P = 20
- Run Time, t_{run} = 5000ms
- Demand, D, = 1.5ec/s (encoder counts per second)
- Number of elites = 10
- Mutation rate for each parameter = 10%
- Settle Tolerance = 10%
- Number of generations = 9 (excluding initial population)
- Number of trials = 13

The parameter values are justified through the following assumptions: a population size of 20 is sufficiently large to explore state space effectively and induce mutations whilst reducing overall trial time; Run time of 5000ms is sufficient time to show proof of settling but short enough to promote low fitness scores; Demand of 1.5ec/s is an appropriate use-case speed for the POLO3pi+ robot moving in a straight line; Number of elites of 10 ensures retention of best performers whilst maintaining potential to variation of future generations; Mutation rate of 10% is low enough to not overwhelm other GA procedures yet high enough to allow mutations within the population size; Settle tolerance of 10% is used to converge towards a reasonable yet obtainable result within a physical system; Number of generations of 9 is used to test the validity of the outlined hypothesis in section I-A; 13 trials is sufficient to reduce the effect of randomness to evaluate the GAs average performance.

The implemented PID ranges are outlined in table I. These are set based on preliminary PID testing, ensuring motors are not burned out and are able to display a wide array of possible controller behaviours.

PID Values \ Range	Low	High
K_p	30.0	200.0
K_i	0.0	2.0
K_d	-200.0	0.0

TABLE I: Table showing the K_p, K_i, K_d range of values found through previous testing to ensure no motor damage whilst training GAs

The fitness function, however, differs between the two experiments. Constant demand is evaluated using a weighted combination of the forward settling time, t_{sf}, and overshoot, δ, normalised between 0 and 1, as seen in equation 1. The weighted ratio is set to 4:1 in favour of the settling time. This is to ensure that individuals who produce overshoot, yet quick settling times, are generally regarded higher than those with less overshoot and longer settling times. Short settling times are regarded as the favourable outcome for this task, as this mean the system responds quicker and more accurately to the desired wheel speed.

$$\gamma_{const} = 0.8\left(\frac{t_{sf}}{t_{run}}\right) + 0.2\left(\frac{\delta}{D}\right) \tag{1}$$

For changing demand, an identical ratio is chosen between settle and overshoot, however, an additional backward settling time, t_{sb} is included. Equation 2 demonstrates the modified function, containing an additional 1:4 ratio between forwards and backwards settling times. Upon flipping the desired direction of rotation, the absolute difference in demand is double that of when stationary. Hence, it is deemed more important for an individual to react and settle quicker during this phase than at the beginning.

$$\gamma_{change} = 0.8\left(0.2\left(\frac{2t_{sf}}{t_{run}}\right) + 0.8\left(\frac{2t_{sb}}{t_{run}}\right)\right) + 0.2\left(\frac{\delta}{D}\right) \tag{2}$$

At the end of each generation, the robot reports all performances, printing their generation number, individual ID, parameter values and fitness score in ascending order. Data is copied to a csv file to be plotted and analysed.

To validate the constructed fitness function, a high performing controller will be extracted from the dataset and analysed to investigate its average wheel speed and error over the course of an example run.

IV. Results

A. Constant demand for straight line speed

From the initial set of experiments, focusing on a constant demand for the PID, the results of how the fitness scores evolve with generations of the genetic algorithm are shown in figure 2.

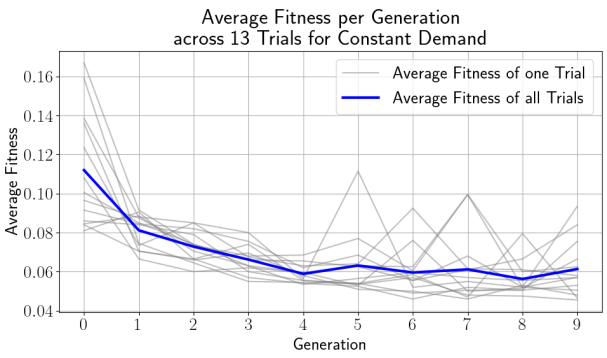

Fig. 2: Average fitness per generation over 10 runs (blue) over each individual trial (grey) for a constant demand.

Figure 2 evidences the performance of the genetic algorithm, displaying a clear convergence of the average fitness towards approximately 0.06. The effects of mutations on the population can also be visualised in the plot. Although the average fitness score shows a general converging trend, individual trials can be seen to display spikes in fitness score. Yet due to elitism these plots quickly converge back to the average as the mutations with poor fitness scores are filtered out.

Once the average score converges, smaller fluctuations above and below this value are observed. This is due to the similarities passed on from elitism and crossover. If no mutations occur, solutions will vary only slightly amongst already top performing individuals. Although an appropriate solution may have been reached, this reduces the rate of improvement due to the discovery of a minima. Without mutation, the ability to escape a potential local minima is not possible.

Examining the data at an individual level reveals an additional cause of minor fluctuations in fitness scores ($\pm \approx 0.01$). Identical PID values can produce slightly different fitness scores in subsequent generations. This is due to variations in the physical system, such as motor response, encoder counts, battery levels etc, the effects of which cannot be controlled.

B. Changing demand for straight line speed

Continuing with experimentation, alternating to a changing demand, the versatility of the method over more areas of speed control is studied.

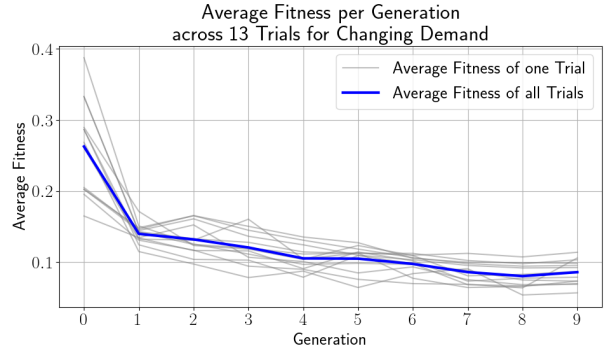

Fig. 3: Average fitness per generation over 13 trials (blue) over each individual trial (grey) for a switching demand.

Figure 3 evidences the performance of the genetic algorithm applied to a more complex experiment, displaying a clear convergence towards an average fitness of approximately 0.08. Although from an overview this experiment seems to show fewer fluctuations than the constant demand experiment, it must be noted that the displayed axis differ in ranges. In fact, changing demand has more occurrences of larger spikes as the generations progress. This, however, was to be expected as the settling times are normalised by only half the run time, producing larger variation in scores for equal perturbations. For reference, constant demand fitness scores are normalised over the total run time. Furthermore, the maximum absolute difference in demand is double that of constant demand when changing direction, requiring more effort to meet the new demand threshold. Due to this it is expected that the changing demand experiment will generally more variation and worse performances in the results of the average fitness score.

As seen in the previous experiment, it is also observed that mutations may cause spikes away from the average fitness for singular trials. Despite this, the algorithm shows a strong ability to filter out these scores and converge back towards the average fitness score.

C. Evaluating Mutations

Variation across individual trials are imminent due to the random processes contributing to the genetic algorithm sequencing. A key factor in determining the average fitness of a single generation is outliers. Figure 4 shows the average fitness per generation of a highly variational trial. Although average fitness does maintain a general downward trend, many deteriorations in overall performance are noted in subsequent generations. However, comparing to it's equivalent box plot, it is seen how the cause of rising fitness is due to outliers. Mutations can alter any of the 3 parameter values within the entire searchable state space ranges. This means that individuals may get thrown far from the local minima, producing insufficient results, such as in generations 4, 6 and 7. The opposite is also true, where individual mutations may improve their respective performance. An example is seen in generations 8 and 9, showing outliers below the interquartile ranges. The general convergence of the GA can be noted by the decrease in median values and smaller interquartile ranges as the algorithm proceeds into later generations.

D. Evaluation of high performing fitness scores

In light of the results detailed in IV-A and IV-B, it is imperative to have a fundamental understanding of what constitutes a 'good' fitness score and how it translates to motor performance. Figure 5 displays the performance of an individual PID controller produced by the GA, with a fitness score of 0.04.

Several features from figure 5 indicate that low fitness scores translate to good performances. The first feature, making up the largest weighting of the fitness function, is the settling time, for which this PID has an incredibly low settling time with respect to overall run duration. Furthermore, this example shows no overshoot, a characteristic that can cause large initial errors. In addition to this, once the average speed and error have converged to within

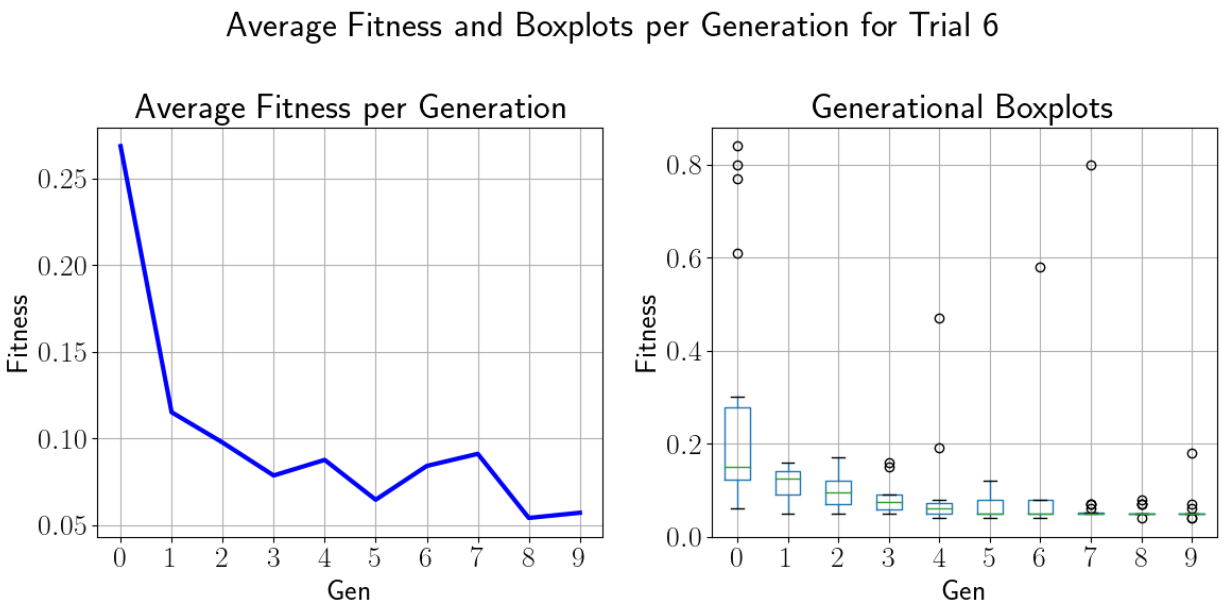

Fig. 4: Average fitness per generation (left) and boxplot per generation (right) for trial 6, chosen as a highly variable trial evaluating effect of mutations.

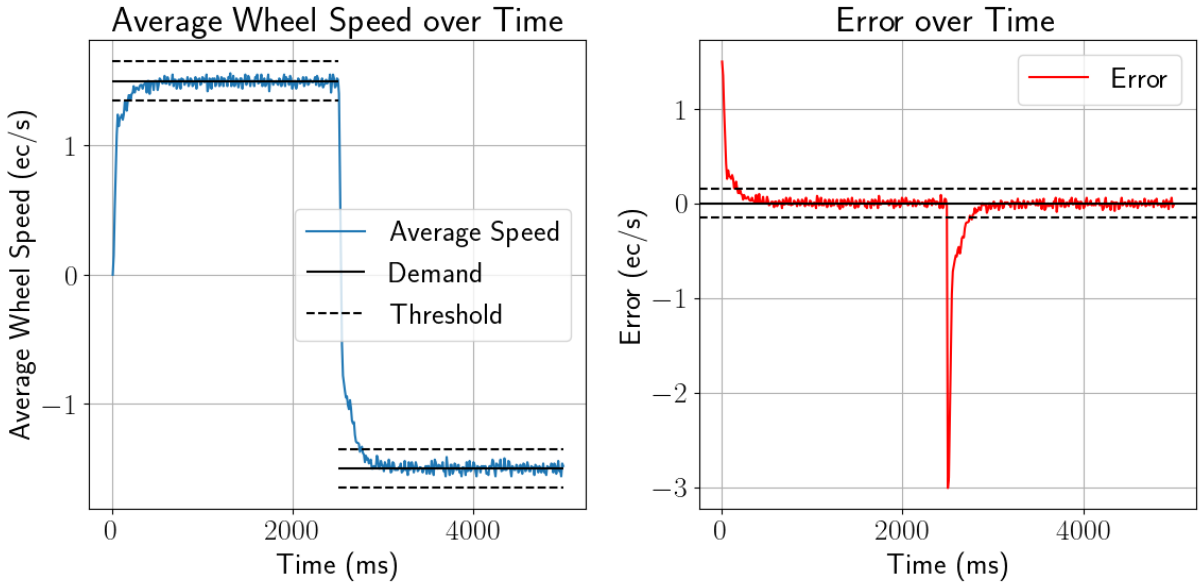

Fig. 5: Average speed (left) and the error with regards to the demand (right) over time, for a PID with a related fitness score of 0.04. Threshold values can also be visualised (dotted black) which are 10% either side of the demand for speed and error.

the threshold, there are only small fluctuations from the required value. The final aspect, which is specific to the changing demand case, is how the PID adapts to a switch of demand. It is observed in figure 5 that this case has an incredibly fast reaction to a severe switch in demand, maximising performance. All of these factors constitute what makes a good PID speed controller and for that reason form the base of evaluation for the functions.

V. Discussion

***H1**:Given a reasonable range of values that the initial population can be drawn from, a PID controller for straight line wheel speed for the Pololu 3pi+ robot can be tuned in 10 generations.*

It is clear from figure 2 that a PID controller can be tuned for constant demand using genetic algorithms within less than 10 generations with an average convergence towards a low fitness level shown. This proves the stated hypothesis, tuning for constant demand through a fitness function which incorporates settling time and overshoot optimising for straight line speed. Furthermore, the translation of a low fitness score to the physical system (figure 5) presents low settling times and minimal overshoot, indicating the Pololu 3pi+ robot can effectively traverse in a straight line.

***H2**: H1 holds when the robot begins from rest to achieve its demand and when the direction shifts to the immediate reverse.*

From figure 3, it is shown that the GA can additionally tune a PID controller for changing demand within 10 generations as the average fitness scores are seen to converge. The performance of a produced controller for changing demand is also evaluated in figure 5, illustrating rapid response to sudden demand changes.

***H3**: H1 and H2 will hold when introduced to a frictional surface.*

The genetic algorithm is shown to be versatile to multiple demand conditions. It is therefore hypothesised that the GA can be applied to the Pololu 3pi+ robot when placed on a frictional surface. Given a frictional surface that allows normal motor function, the effects of the opposing forces should reduce settling times by a small factor, but insufficient as to limit convergence to the demand within the run time. Damping caused by friction may also improve certain controllers by reducing overshoot. Conversely, some current solutions may no longer be viable due to overdamping. Therefore, using only the evidence given in this report, it is not sufficient to prove this hypothesis at this stage.

A. Limitations

The application of the GA for further motor control is limited by higher demands due to battery limitations. Higher speeds require more power, draining battery levels quicker over the course of a trial. Trials were often required to be re-ran due to this over the course of the experiment. It is important therefore that the demand is set to a value which allows sufficient generations to be ran to achieve convergence.

The GA was designed based on prior knowledge of the system. This knowledge was leveraged to reduce the size of the state space being searched. In turn, the genetic algorithm is able to converge more quickly than for other potential applications, allowing us to achieve the desired hypothesis effectively. There is insufficient evidence to support that this genetic algorithm would perform as effectively on larger scopes. However, given enough generations, it is likely that the formulated algorithm structure would converge to an appropriate solution.

Similar tasks involving PID control for the 3pi+ robot may also rely on short settling times and minimal overshoot, such as line following and rotation. However, the fitness function formulation and weightings presented may be insufficient for such tasks. Accurate heading alignment may require slower yet more precise adjustments, indicating a preference for reduced overshoot over shorter settling times. Before implementing the GA on such challenges, it is important therefore that the fitness functions are adjusted appropriately for the required task.

Genetic algorithms present their own need for optimisation, tuning parameters of the system for more effective results. The effect of changing the algorithm parameters has not been explored in this report. Although justified and effective for the specified task, it is not shown that the chosen parameters values are optimal for the GA formulation. It is possible therefore that the genetic algorithms used in this experiment may be improved to provide convergence to lower average fitness scores in fewer generations.

VI. Conclusion

In conclusion, it has been found that genetic algorithms are able to effectively tune a PID speed controller for straight line travel, within 10 generations to a fitness score of less than 0.1. Furthermore, this conclusion is extended to the case of changing direction for which an identical controller is tuned in under 10 generations to the same threshold. It is further theorised that the hypotheses will hold for straight line speed on a surface with friction. The reasoning behind this is that the addition of friction to the system applies slight damping over the PID curve. The extended hypothesis states that this would result in an increased settling time yet decreased overshoot of the PID, resulting in similar fitness scores. Despite this, the extent of the effect of friction is unknown, and therefore it can only be hypothesised that this method would be successful on frictional surfaces. Although more efficient formulations and parameter values may be possible, the genetic algorithm is shown to be robust to varying scenarios. As a result, the structure of the genetic algorithm is likely to be transferable to similar tasks involving PID motor control, such as line and heading following, with appropriately altered fitness functions. The current GA is, however, a quick, simple and effective alternative to manual tuning methods, especially for those new to these systems.

VII. Future work

While genetic algorithms have been proven to be effective for the situation where the robot is lifted off the ground, future work would aim to test the viability of the GA when the robot is introduced to a frictional surface. This presents further challenges not experienced in this experiment. Generation and trial results would require to be stored onboard during run time. Limited volatile memory storage may limit the number of individuals and generation that may be run within any single trial. Furthermore, the robot will now move during run time, hence sufficient space is required to traverse the specified speed for the given run time.

Further work can be found in the exploration of the optimisation of the genetic algorithm itself. Alternative artificial intelligence/machine learning frameworks may be utilised to investigate this challenge. Also, computer vision models may be deployed to refine the desired fitness functions. Deviations in straight line traversal may be captured by these to incorporate simultaneous tuning of both wheel PID controllers, aimed at minimising the difference in speeds between the two motors.

References

[1] Pololu Robotics and Electronics, "3pi+ 32U4 OLED Robot - Standard Edition (30:1 MP Motors)."

[2] D. Sellers, "An overview of proportional plus integral plus derivative control and suggestions for its successful application and implementation," 2001.

[3] M. Mitchell, *An introduction to genetic algorithms*. MIT press, 1998.

[4] T. W. Manikas, K. Ashenayi, and R. L. Wainwright, "Genetic algorithms for autonomous robot navigation," *IEEE Instrumentation & Measurement Magazine*, vol. 10, no. 6, pp. 26–31, 2007.

[5] G. Levitin, J. Rubinovitz, and B. Shnits, "A genetic algorithm for robotic assembly line balancing," *European Journal of Operational Research*, vol. 168, no. 3, pp. 811–825, 2006.