Bio-Inspired Robotic Navigation On Varied Terrain

Dexter R Shepherd [candNo: 215819]
Supervisor: Dr James Knight

Final Report
Computer Science and Artificial Intelligence BSc

School of Engineering and Informatics
University of Sussex
2022

Statement of authenticity

Acknowledgements

I would like to express my gratitude to my supervisor, Dr James Knight, for the support and guidance he has given me throughout this project and prior to this within the JRA program. Additionally I wish to show my appreciation to Professor Andy Philippedes and Dr Chris Johnson for the papers they sent to me and meetings they had with me. This all inspired this project and the use of Wheg locomotion. I extend my thanks to the Centre for Computational Neuroscience and Robotics at the University of Sussex for supplying this project with stereo vision sensors and Raspberry Pi components. Finally, I wish to thank StackOverflow, to whom I owe a lot of my degree to.

Summary

This project investigates the use of evolutionary approaches to enable robotic agents to learn their own physical limitations in a visual navigation task. Within nature, biological agents understand their own limitations when it comes to crossing complex terrain. By attempting to cross terrain that an agent is not built for they risk injury or even death. A robot needs to understand its own limitation if it is to be able to explore complex terrain but not get damaged. One applications of this would be planetary exploration where robot damage would ruin a cost expensive mission. The project was split into three parts: Navigation; vision; and the development of the physical robot.

Navigation strategies were developed in a two-dimensional simulation world with procedurally generated terrain. Simulation takes less time to trial than physical experiments, therefore a proof of concept could be developed within the time constraints of this project. Environmental hazards were introduced, such as water, and experimentation into hyper-parameters and algorithm choice allowed convergence on high fitness models. Agents learned to follow contours in order to minimize energy consumption and avoid hazardous environments.

Two depth sensors were trialled for visual base navigation. One was using two cameras and the OpenCV disparity function; the other an off-the-shelf approach with the Xbox Kinect. Stereo imaging was picked as it best highlighted the environmental attributes that would be relevant for solving the problem of hazardous terrain representation. The Kinect sensor was the best at finding depth. Further processing took place to squeeze the depth images into a compressed representation that could be used in the simulated trained agent model.

The physical robot made use of the visual depth information and neural networks trained in simulation trials which were able to cross the bridge between simulation and the real world (known as the 'reality gap'). It could predict hazardous terrain for avoidance in an outdoor environment.

The chassis was inspired by how cockroaches cross obstacles, it used Whegs and a bendable back to improve movement over rocky and outdoor environments. Whegs are a hybrid between wheels and legs where the cyclic efficiency of the wheel is used, but claw features are added to improve climb like a leg. The back bending control was learned by the system using a population of hillclimbers evolving the weights and biases of a neural model.

This project successfully demonstrated that an agent can understand its hazardous environment, and learn to overcome obstacles.

Contents

1 Introduction

In this project, I have investigated autonomous robotic navigation through localised decisions based on previous experience. Nearly all organisms are limited at some point when it comes to movement. For example, an octopus can fit through an aperture that is bigger than its beak [1] and a person can climb over rocks, but only if they have grip points for hands and feet. These environmental barriers define constraints to an organism's ability to traverse across it. When a path looks too arduous, we will often choose a more accessible route if possible.

These constraints also apply to robots. Consider a Mars Rover over 395 million km away from the person controlling it. If the Rover were to get stuck, there would be no one to recover it and – as it takes 181 seconds for a signal to get from Earth to Mars – real-time control is impossible. Therefore, a robot, like a biological organism, needs a sense of self-preservation thus needs to know its limitations and not attempt tasks of movement outside these physical constraints [2].

Our agent was set a target location and had to reach it while avoiding hazardous/overly complex terrain within the environment. Success was defined as whether the agent got there safely, and how much extra energy was consumed. When deciding on routes, there may be the scenario that all paths are within the constraints but some are simpler to navigate than others. In this situation, the agent will need to pick the option which requires the least effort. The best-case scenario for the agent I am developing is governed by the level of complexity of movement necessary to overcome the obstacle. If the agent gets stuck it will get no reward; if the agent makes unnecessary movement while moving over simple terrain, it will get little reward.

I initially explored this problem using simulation, before moving to a robot platform with a back actuator, stabilizer and neck actuator. The robot needed to learn to use these features to help it navigate over the landscape. This robot performed this task by using computer vision techniques, which forms a prediction on the best route to take via movement instructions generated at each move.

2 Professional and Ethical considerations

In line with the BCS Code of Conduct, we must only undertake research within our competency. The Computer Vision and Acquired Intelligence; and Adaptive Behaviour modules have given me the relevant knowledge to attempt the task of terrain navigation. In addition to this, the Junior Research Associate scheme has given me a valuable background in robotics and evolving hardware.

While no ethical approval is needed for this project, there are indisputable risks of dual use in AI research. However, at the fundamental level that I am working on, I have made a conscious decision to prioritise Open Science and all of my code will be publicly available on Github: https://github.com/shepai/Dissertation.

Because we will train our robot on images of real world environments, we will have to consider the dangers of people being in view and thus consider the GDPR rules. However, as the resolution of images will be low, individuals will not be recognisable. Another ethical consideration is the safety involved in testing as the robot could be a trip hazard. To address this, we will only test the robot in an environment where it cannot hit people, specifically the safety-netted area of the Future Technologies Lab.

3 Background Literature

3.1 Robot Locomotion

A standard wheel has low energy consumption due to a single axis continuous rotation; however, it is limited when climbing over rough terrain. To address this, agents can use legs for example in a hexapod configuration which is incredibly stable [3]. However, robotic legs require multiple servo motors which have a high current draw and, due to the need for inverse kinematics [4], require complex control algorithms.

The 'Wheg' design is a wheel-leg hybrid that can traverse more challenging terrain than standard wheels while maintaining low energy consumption [5]. Figure 1 shows a robot featuring Whegs with claw-like spokes which dig into rocks and obstacles.

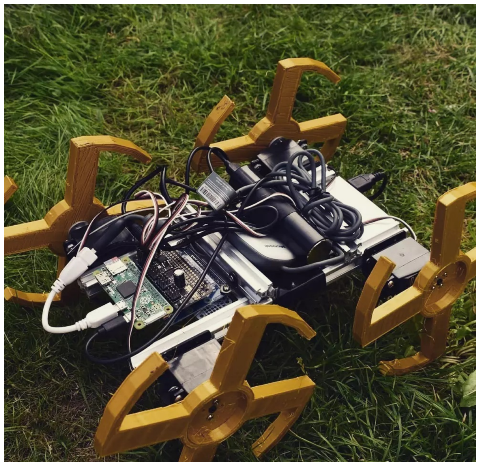


Figure 1: Prototype of the Wheg robot to test out the wheg design, in addition the hardware required to move around an environment. This prototype ran on a Raspberry Pi zero and used a phone charger cell providing 5.1V and a maximum of 2A of current.

Whegs are easily manufactured via 3D printing, and there are multiple design variations of the Wheg which suit different purposes [6]. The European Space Agency (ESA) adopted a similar design, shown in figure 2, for the PROLERO project [7]. This project used a simpler design of actuation where the legs were rods rotating on one axis. This design could at maximum travel over obstacles less than or equal to a height of 10 cm. The total payload mass was 1.5 kg making it light for space travel and the rover passed the tests at the ESA planetary utilisation facility, confirming the model's validity [7]. Although not quite Whegs, these legs operate on a similar principle.

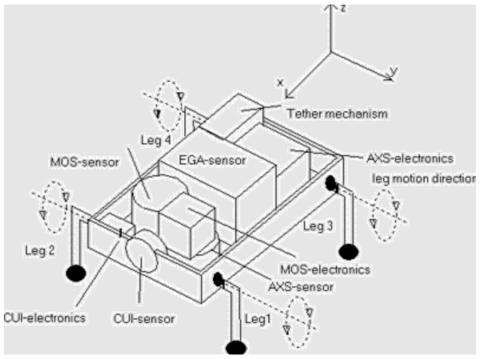


Figure 2: The PROLERO robot taken from ESA [7]

Other potential designs are caterpillar tracks which have good energy efficiency due to the optimised traction system of caterpillar tracks, where one side pulls and the other side pushes. This also allows more grip over uneven terrain. They do, however, weigh more and are more difficult to repair than wheels meaning that the robot would be inoperable if a link broke. Whereas, with wheels or Whegs, it could continue to move with worn-down tyres or grippers.

3.2 Biology and Bio-inspired Robotics

Research on cockroaches inspired the mechanical design of the chassis used in this project. Specifically, cockroaches bend their backs to climb over obstacles as illustrated in figure 3. Furthermore, when cockroaches run into a wall, they bring their limbs into phase at the point of climb. The rotational locomotion of the legs is comparable to the cyclic locomotion of wheel. The robot developed in this project uses a Wheg design to improve its advancement in a similar locomotion. A bend in the robot created a higher stance to climb [8].

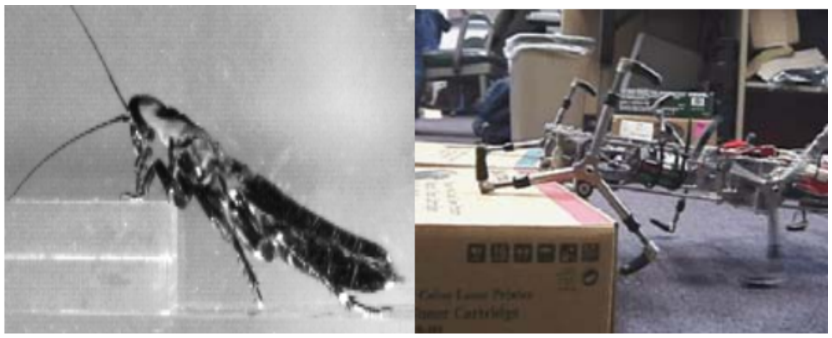


Figure 3: How the cockroach rotates its limbs to climb an obstacle compared to a bio-inspired robot (after [8]).

3.3 Genetic algorithms

Genetic algorithms (GAs) are optimisation methods which are inspired by biological evolution [9]. Over a number of generations, encoded information (*genotype*) will be changed (*mutated*), where the suitability of this change is measured via a fitness function. If the mutation has helped the overall goal then it will get a high fitness and replace other genotypes within the population.

One way to use GAs is to learn the weights and biases of a neural network. At the beginning of the process these are picked at random e.g. from a Gaussian distribution and each generation allows mutation of these values. Mutation functions fit Gaussian noise on top of the current encoded data, which alters the operation of the neural network when applied.

The Microbial algorithm [10] is a GA that generates a population of agents. The Microbial trials each of the agents in a tournament against one another. The winner overwrites the loser. Particle Swarm optimization [11] uses a particle physics-inspired approach where, if particles are energised through heat and cooled slowly, they move back into place better than if cooled quickly. This theory is translated into a genetic algorithm in which sub-optimal solutions move away, enabling new new solutions to be trialled. Using any of these GAs, an agent can be optimised for a task through guided behaviour.

3.4 Evaluating performance of robots on terrain

It is important to be able to quantify an agents' performance across trials. Current research in rough terrain robotics has used physical attributes such as the coefficient of friction and slippage plotted against the climb [12].

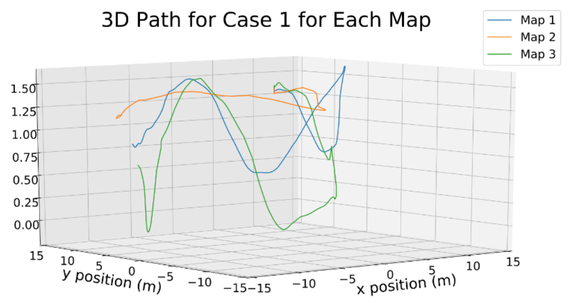


Figure 4: A 3D plot of the simulated paths for a moon rover [2]. The z-axis shows a prominent climb on the experimentation map 1 and map 3. These maps are preset terrain within the simulation. It is visible that map 2 contained smoother terrain.

Studies [13] [14] have measured locomotion performance based on several values, including slippage, required torque, and minimum friction coefficient. These metrics create an overall score, plotted throughout each trial. μ represents the friction coefficient needed for tangential (R) and normal (N) force with (r) given by the radius of the wheel with torque (T).

$$\mu_{needed} = \frac{R}{N} = \frac{T \cdot r}{N} \tag{1}$$

Slippage consumes energy without making the robot move forward. Slippage was calculated using the encoders on wheels. Distance is calculated using the following equation which uses encoder values. The wheel encoder (Δ_{coder}) is used and multiplied by the circumference of the wheel ($2 \cdot \pi \cdot r$). This divides over the pulse/turn multiplied by the gear ratio.

$$\text{Distance} = \frac{\Delta_{coder} \cdot 2 \cdot \pi \cdot r}{N_{pulse} \cdot R_{gearbox}} \tag{2}$$

The slippage is the difference of ground truth and wheel odometry [13]. The simulated robots used in Figure 4 could provide enough torque, but that does not mean equal performance. The factors that measures the robot journey success form a Max and Min G score. The G score is taken from all the torque needed. The attributes required for the Max and Min G are easily calculated in simulation, but would be harder to find on a physical robot. The use of gyroscopes and accelerometers are potentials for a substitute for measuring slippage, as well as reading current from the motors to evaluate torque. Indoors, we could also use Vicon tracking to precisely measure the 3D pose of the robot which would allow us to quantify these attributes.

3.5 Sensors

Terrain mapping sensing can take many forms. One study [15] used a lidar (Light Detection and Ranging) sensor that would read at four layers to construct a world model. This model was then used to create a 3D perception allowing the hexapod robot to predict the appropriate motor movement of its chassis to interact with the environment.

Optical flow uses the vectors between pixels in two images taken one immediately after another. It allows sensing of movement within an environment, which works well with robotics that are designed to move and navigate. This can be used for optical flow alone [16]. OPtic flow calculates the velocities per pixel in images taken one after the other. This then is used to predict where the pixels will be next. Optical flow can also be used to calculate where obstacles are and swerve an agent away from the obstruction via depth estimation maps [17]. This depth estimation method was achieved using a convolutional neural network and the Canny Edge Detector and morphological operators [18]. The method would calculate the median between a pixel and neighbouring pixels as input with a 5x5 filter within the convolutional layer. The input itself used morphological operators as modification of the canny edge detector in order to improve edge detection accuracy. One example of this is a light transformer function.

Depth perception via stereo imaging and time-of-flight is another method. Time-of-flight sensing measures depth using time delay between an object and light reflecting from it. This allows depth estimation. Stereo imaging works by triangulating pixels by looking at the distances.

The hexapod robot shown in figure 5 used this method with six legs to predict stable movement over uneven terrain [19]. This prediction method showed a high accuracy, while being computationally less advanced than the lidar method [15].

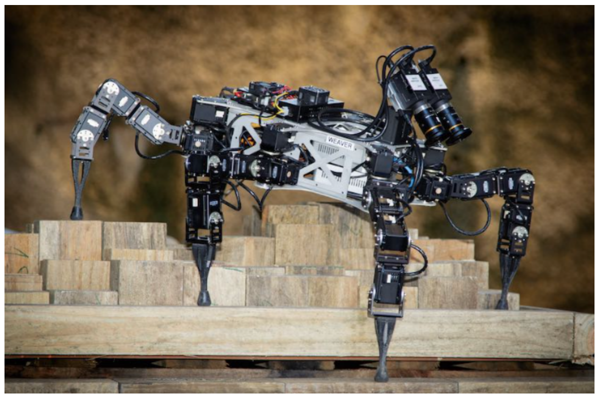


Figure 5: Weaver the stereo vision hexapod. The ground is uneven for all legs, therefore the agent must adapt across all legs.

3.6 Contributions to the field

Giving autonomous robots a sense of self-preservation is important in scenarios such as planetary exploration, or hazardous locations on earth like nuclear reactors. In general, by prolonging the lifespan of robots [20], it will save money. Our approach will use computer vision techniques to evaluate terrain and make safer and easier path decisions. Our design will also be implemented on a relatively cheap chassis, aiming to bring down the overall cost of robot development.

4 Requirement analysis

Existing robotic solutions have not combined bio-inspired chassis design with evolved learning. This project will use a physical robot platform as well as a simulation of this platform for the gathering of data. Simulation provides a proof of concept by trialing navigation much quicker than a physical robot.

4.1 User needs and ideal features

The agent will need to learn how to recognise the differences between rough and smooth terrain based on visual information and furthermore learn how to traverse it. Ideally, the physical robot will use its back actuator to improve its ability to climb over terrain. The agent will also need to understand its limitations and not attempt routes that it will get stuck on. Ideally this will be demonstrated on a physical robot but, if real world noise prevents this, it is still essential the algorithm works in simulation as proof of concept.

4.2 Limitations

This project does not intend to create a robot that can climb any terrain. While the Wheg design will improve the robot's climbing ability, it will not make it impervious to the laws of physics!

4.3 Problem the system is solving

The problem this research is solving is to improve autonomous robotic navigation for danger detection without the use of training data that is specific to an environment. A robot can be deployed within different environments which makes obstacle recognition harder. Additionally, from an engineering point of view the robot is trialing Wheg designs which are an interesting middle ground between wheeled robots and those with articulated legs. As most robotic navigation projects use conventional apparatus, Whegs remain under-explored.

5 Navigation

5.1 Simulated world

I set up the simulated environment illustrated in figure 6 using Python by generating a 2D map from Perlin noise. Perlin noise is a type of gradient noise commonly used to procedurally generate terrain [21] where the value of the space between two points changes smoothly. A grid of n dimensions is created and each position is assigned a random vector, where the dot product is calculated between the gradient vector and the offset vector. This allows interpolation between each dot product to generate noise within the grid.

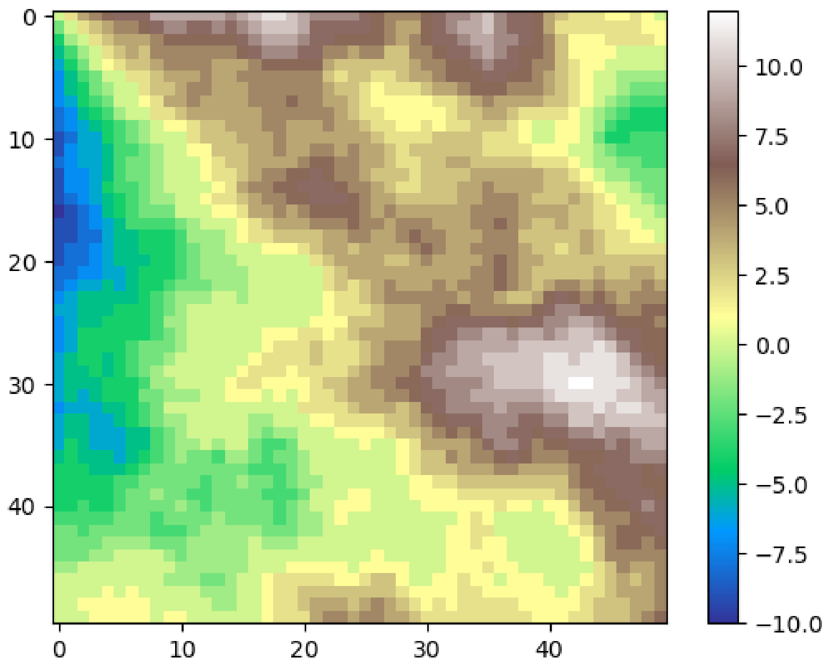
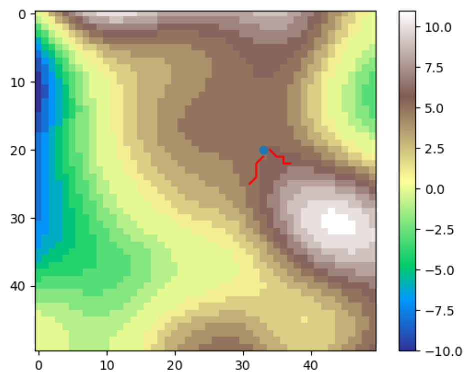


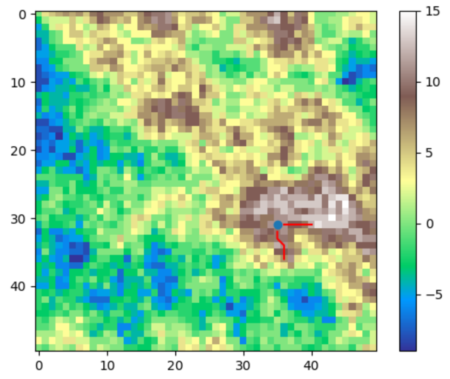
Figure 6: 2D simulation plot of a 3D terrain. Random generated points are shown to show the position of agents and rewards.

The smoothstep function represents a sigmoid-like interpolation. The persistence, octaves, and lacunarity are all altered via parameters in the world creation function. The octave is generated via a combination of frequency-amplitudes, and the number of octaves entered defines how many times this occurs. Each octave adds a layer of detail to the world, where the contribution of each octave is defined by persistence. The persistence is multiplied by the amplitude within the noise generation which effects the decreasing of the octaves. The lucunarity defines how much detail is added or removed at each layer. This will change the frequency within each octave, which will change the height of each index of noise.This function formed a map held as a two-dimensional array representing terrain heights numerically.

In such simulations, we can find how far a point is from the start position and the terrain steepness (gradient) at any point. The energy consumed (E) by a robot in a given trial is calculated from the number of steps taken (S) and the accumulated terrain value of each place (P). Each position on the map has a value that represents how high or low the ground is. Very low values would be submerged underwater and high values be at the top of the mountains. The difference between movement is the step S up, down, or across from the current height. This is called the terrain value. More energy is used going uphill than downhill, so these state transitions are valuable for calculating fitness. The Manhattan distance is used as a measure of how far the agent has travelled. This was picked in place

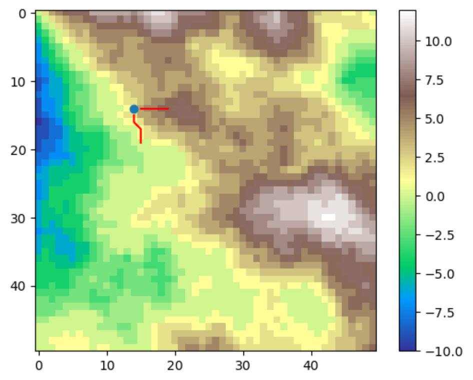


(a)

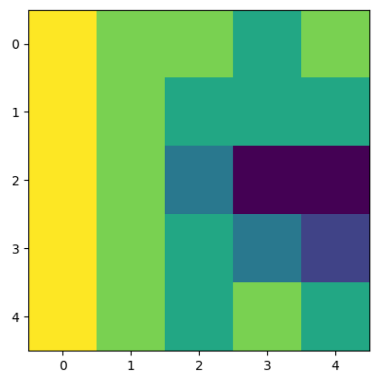


(b)

Figure 7: (**A**) Terrain with low persistence. (**B**) Terrain with high persistence.



(a) The Agent field of view



(b) Agent vision visual map

Figure 8: How the agent views the world and an aerial view that can be displayed. The agent is looking downwards on the visual map. This is recognisable by the increased climb in the in the bottom-right corner of the map. This corresponds to the hill climb facing the agent.

of the Euclidean distance as the 2D terrain guarantees a worst case scenario [22]. Euclidean distance finds the shortest path, which may not be the best path.

$$E_{nergy} = (\mid x1 - x2 \mid + \mid y1 - y2 \mid) \cdot \sum_{i=1}^{S}(P_i) \tag{3}$$

By increasing the persistence, the terrain can be made more complex and, additionally, simulated dangers such as water can be added.

For example, no real challenges would exist for an agent in Figure 7a due to the smooth terrain and very small amount of water. However, an agent navigating the terrain shown in Figure 7b has several pools and steep slops it would need to navigate.

A first person depth view was generated from the terrain by taking several vectors outwards from the agent at altered angles of 20 degrees per reading. The red lines within Figure 8 show a field-of-view of 100 degrees. x and y represent the current agent position, and the iteration of x_i and y_i represents the next coordinates at the next iteration. α represents the start angle that the agent is facing in.

$$x_i, y_i = x + ROUND(|d \cdot cos(\alpha + 20i)|), y + ROUND(|d \cdot sin(\alpha + 20i)|) \tag{4}$$

The round function converts the values to integers so that the pixel coordinates can be accessed. The image is generated by going through each pixel coordinate found and comparing it to the current coordinates of the agent $(current_x, current_y)$ height value.

$$Height = world_{current_x, current_y} \tag{5}$$

$$newPixel_{k,l} = world_{x_i, y_i} - Height + 50 \tag{6}$$

The generated $newPixel$ at the indices of k, l will have the pixel intensity of the current position of the world at the generated line x_i, y_i where i is the iteration through the array. We subtract the element of initial height so that if the next position is higher, then the pixel intensity will be brighter, and if lower then the pixel intensity is darker. We then add 50 to prevent negative values for the display, as the subtraction of height could form negative values.

Each trial would generate a random coordinate for the agent to start in, with the condition that this start point was not in water so that the agent would not receive 0 fitness before starting. An end goal point would be generated randomly from an array of coordinated on the circumference of a circle at a set radius.

The agent was written as its own class, where on initialisation the network structure could be determined. Weights and biases are generated when an array of Gaussian noise is entered through the *set_genes* method as a parameter. Using a forward function, the network uses PyTorch and generates output that is converted to a vector choice using the *argmax* function. There are eight directions that the agent can take, therefore the network is expected to generate the probabilities of a success of each vector. The success for this method is seen in Figure 12 within the results section. The vector of movement allowed the agent to traverse across the environment.

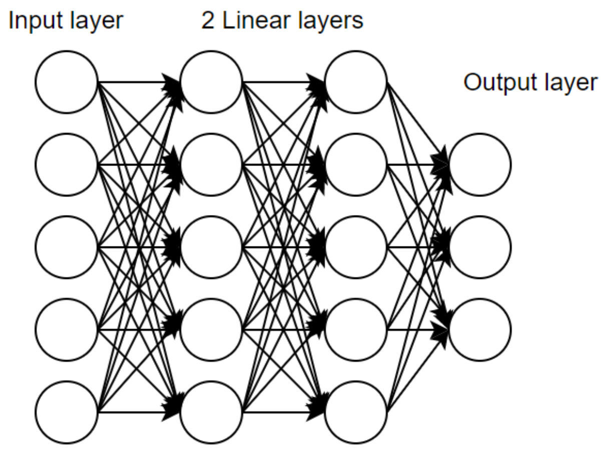


Figure 9: Architecture for the first neural network model where one input layer, one output layer and two hidden layers are implemented.

5.2 Neural network model

The first model was the simple Feed-Forward Neural Network (FNN) shown in figure 9 where an input of a flattened 5×5 image is taken as input. The output is a list of all vectors on where to move within the environment.

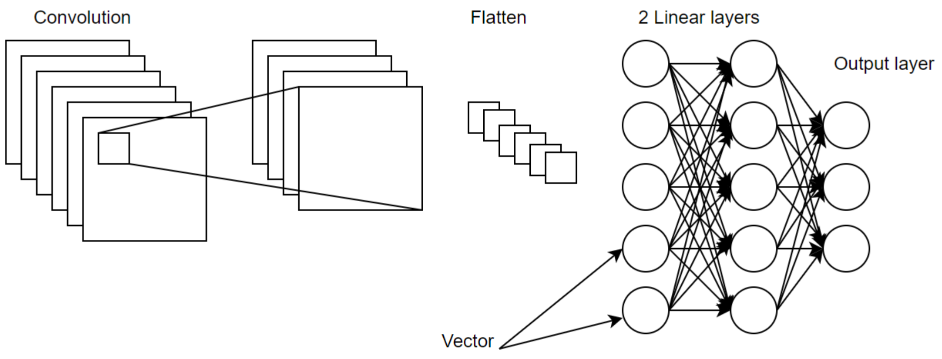


Figure 10: Architecture for the convolutional neural network where one Convolutional layer is applied with a predefined kernel filter.

Figure 10 shows the The Convolutional Neural Network (CNN) architecture. The network was implemented as a class in the code. This network class within the code could have the sizing of the hidden layer width changed on initialisation, where the size was entered as a parameter. The output layer and first linear layer would be preset sizes to take in the convoluted image and output one of the vectors. The weight sizes were calculated between the input, hidden and output by the initialisation function.

The CNN uses a two-dimensional convolutional layer where the input image is convolved with a 2D filter of size 3x3. The result of the convolution is then flattened and fed into two linear layers and an output layer. With both network architectures, the weights and biases of these layers are randomly generated from of a Gaussian distribution and evolved using genetic algorithm as described in section 3.3. The architecture hidden layer width was experimented with to find an optimal size.

All neural networks had eight output nodes, which represented the vectors of movement within a two-dimensional space. The vector would be added to the current position in order to move within that direction. The network uses argmax to decide which node to pick. The node is stated in Figure 11.

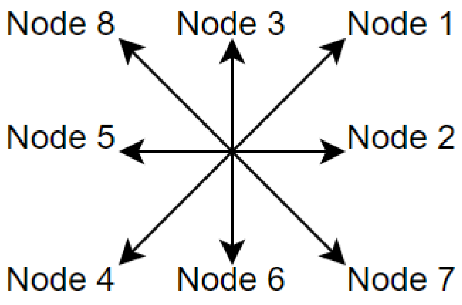


Figure 11: Output nodes labelled for direction in 2D space.

5.3 Fitness

While it is more important that an agent reaches the target, fitness calculations must take into consideration the importance of energy efficiency. Therefore, I gave a weighting of 70% to the distance travelled (calculated based off of the end distance to the point), and a 30% weighting to the energy efficiency calculation. As it is only possible to get 100% energy efficiency by travelling down hill, it is likely that most trials are only able to get a top fitness within the 90% range.

$$\text{Fitness} = \left(\frac{100 - energy}{100} \times 0.3 + \frac{10 - endDist}{10} \times 0.7 \right) \times 100$$

The final step of fitness calculation is to multiply the value by 100 to get it in percentage format. If an agent falls into water (where the position in the world has a value less than or equal to -6) or finishes further away from the end position than where it started, the fitness will be 0. A genotype is trialled three times on random coordinates, and the fitness is averaged. This is to make sure the gene isn't biased to a specific point which would result in a fluke high fitness.

5.4 Strategies

Three genetic algorithm (GA) approaches were taken to solving this task, then an alternative rule based approach was explored to find whether GAs were over-fitting the problem. All three algorithms shared some functions that would be standard across all algorithms to calculate mutation, crossover, and fitness evaluation. Furthermore, across all algorithms, parameters were sampled from the same Gaussian distributions, but were reshaped into formats appropriate to each algorithm.

Mutation was implemented by applying another Gaussian distribution over the top of a genotype. I explored different standard deviation and mean parameters and constrained values between -4 and 4 to prevent values getting too high or too low and biasing the performance of the network. Crossover functions have a probability of crossover parameter, which will copy a value from the winning genotype over to the losing genotype randomly with the probability parameter.

The first GA was a simple microbial algorithm. This generated a population of varied sizes and the optimal was found to be between 10-20 genes. Two genes would be selected from the sample and trialled on three different random maps starting from randomly generated coordinates, and then fitness calculated. The winner would copy over to the losing gene with a crossover between a mutation of the winning gene and the current gene.

Algorithm 1 Microbial

```
Require: $gene\_population$
  $gene1 \leftarrow randomPopItem(gene\_population)$
  $gene2 \leftarrow randomPopItem(gene\_population)$
  $fitness1 \leftarrow runTrial(gene1)$
  $fitness2 \leftarrow runTrial(gene2)$
  if $fitness1 > fitness2$ then
    $gene1 \leftarrow crossover(gene2, mutate(gene1))$
  end if
  if $fitness2 > fitness1$ then
    $gene2 \leftarrow crossover(gene1, mutate(gene2))$
  end if
  $gene\_population \leftarrow gene\_population.push(gene1)$
  $gene\_population \leftarrow gene\_population.push(gene2)$
  return $gene\_population$ =0
```

An alternative gene was explored [23], inspired by groups of genes within a larger population. I used this approach as research has showed that it outperformed microbial algorithms due to having more diversity of genes within winning groups, thus preventing convergence on sub-optimal solutions. Fitness was measured by the summation of the fitness of all genes within a group.

$$\text{GroupFitness} = \sum_{i=1}^{n}(\text{fitness}(gene_i))$$

Algorithm 2 Group Microbial

```
Require: $gene\_population$
  $group1 \leftarrow randomPopItem(gene\_population)$
  $group2 \leftarrow randomPopItem(gene\_population)$
  $fitness1 \leftarrow 0$
  $fitness2 \leftarrow 0$
  for $i \leftarrow 0$ to $LENGTH(group1)$ do
  $fitness1 \leftarrow fitness1 + runTrial(group1[i])$
  $fitness2 \leftarrow fitness2 + runTrial(group2[i])$
  end for
  if $fitness1 > fitness2$ then
    $group1 \leftarrow crossover(group2, mutate(group1))$
  end if
  if $fitness2 > fitness1$ then
    $group2 \leftarrow crossover(group1, mutate(group2))$
  end if
  $gene\_population \leftarrow gene\_population.push(group1)$
  $gene\_population \leftarrow gene\_population.push(group2)$
  return $gene\_population$ =0
```

Both of these strategies had few converging on optimal solutions at the end of trials. Maximum fitness would converge in the 70% range which is seen in the results section. This inspired the creation of the final GA algorithm which would perform a microbial, normally for a certain number of generations, then repopulate the population with mutations of the top gene. This best selection algorithm made little difference to improving the accuracy. The end accuracy between the algorithm and the standard microbial was only 1 to 5% apart.

Algorithm 3 Choose best

```
Require: $gene\_population, gene\_fitnesses, x$
  $topGenes \leftarrow [0...LENGTH(gene\_population)]$
  $marker \leftarrow 0$
  for $i \leftarrow 0$ to $LENGTH(gene\_population)$ do
  $fitness \leftarrow gene\_fitnesses[i]$
  $placed \leftarrow False$
  $count \leftarrow 0$
  while $count < marker$ and $NOTplaced$ do
    if $topGenes[count] < fitness$ then
      $topGenes.insert(count, gene\_population)$
      $placed \leftarrow True$
    end if
    $count \leftarrow count + 1$
  end while
  if $NOTplaced$ then
    $topGenes.insert(marker, gene\_population)$
  end if
  $marker \leftarrow marker + 1$
  end for
  $fitness2 \leftarrow fitness2 + runTrial(group2[i])$
  $gene\_population \leftarrow [0...LENGTH(gene\_population)]$
  for $i \leftarrow 0$ to $LENGTH(gene\_population)$ do
  $n \leftarrow RANDOM(0, x)$
  $gene\_population[i] \leftarrow mutate(topGenes[n])$
  end for
  return $gene\_population$ =0
```

Finally we developed a simple rule-based method which takes the closest direction to the target which avoids terrain that showed significant climb or water. This was achieved by viewing the vectors possible and looking at each decision and removing any that would cause damage to the robot, such as falling in water. Different restrictions were added to the rule based method. Different weighting was applied to these aspects where avoiding complex looking terrain, and moving in the direction of the end goal could be two different directions. It was found to have higher accuracy when the weighting of the movement in the set direction was reduced, and the routes that lead to potential dangers were ignored. This allowed the agent to try to move towards the best location, but favour avoiding difficult terrain, and completely ignoring hazardous terrain.

For initial genotype selection the population had two random indices selected. This allowed tournament selection between both. A later approach would select one index and select a neighbouring genotype to have a slower convergence on sub-optimal solutions. This improved the accuracy of the end genotype.

5.5 Results

The tests for simulation gave varied results based on the different algorithms used. Random generations of where the agent was within multiple trials was averaged for each fitness. This prevented fluke results of agents biased to specific routes. The most effective was the group of microbial with re-selection of the best genes.

The architecture of the network was decided through several trials of all networks. Using a microbial algorithm with the same network for 10 trials, the results of network performance was quantified. The tests used a population of 15 and 200 generations.

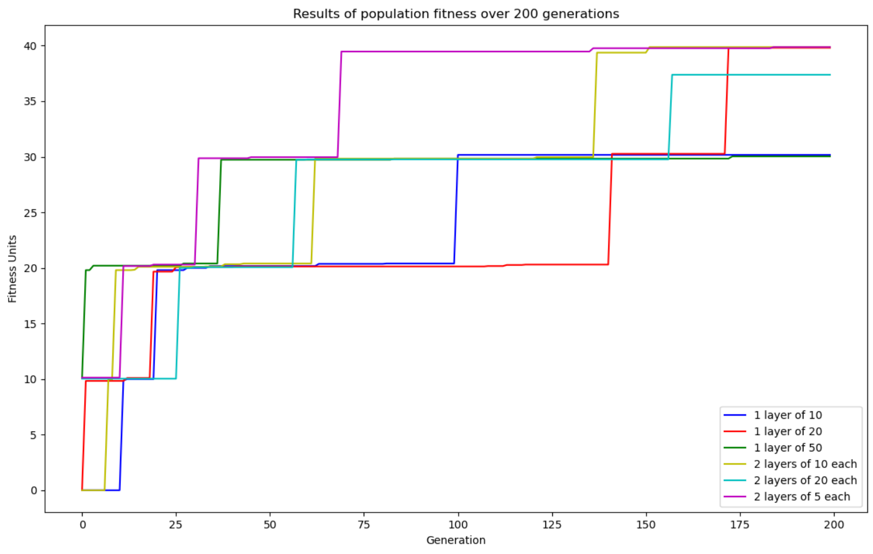


Figure 12: Averaged results of different architectures on the same task using the microbial algorithm

Notably in Figure 12 the fitness units are very low due to averaging different trials of the same network. The population is regenerated randomly each time which leads to some trials having a low fitness bearing population by default. The results were insignificant to suggest that there was a trend for best accuracy within this sample size. The network chosen for the testing was of two hidden layers of 10 nodes each as it performed well and was the between sizes 5 and 20 who also performed well. The smaller network of two layers of 5 performed equally well to two layers of 10 nodes, but as there is an element of randomness to these results, the medium architecture seemed optimal.

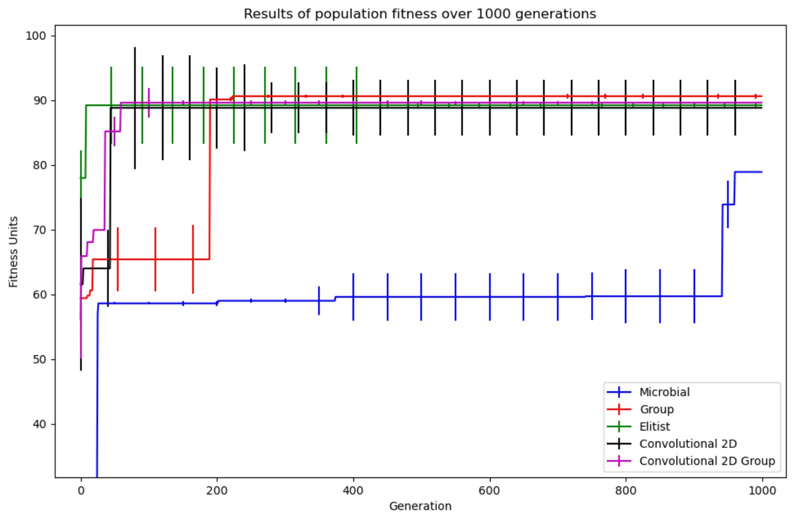


Figure 13: Results of the different network architectures after 1000 generations. The convolutional networks used the microbial algorithm, unless stated that they used the group algorithm. It is visible that the fitness results are much better in Figuree 13 than in Figure 12. This is because the results in this figure have had 1000 generations to evolve, whereas the previous figure has had 200. The error bars are plotted to show the standard deviation of three trials per network. The error bars display the standard deviation between each trial at the given generation.

It is clear that the results of many individuals converge on a solution which is sub-optimal, and eventually cannot improve above this. Using a grouped population, the theory is that more internal diversity would provide better solutions and prevent convergence of sub-optimal solutions too early. However, the results show the fitness of these grouped populations to be considerably lower than the other algorithms. Results of the elitist algorithm (which picks the best genes and repopulates at the end of each sub-generation) converged on the highest fitness quicker than the other genetic approaches, although, after sufficient generations, the microbial eventually performed better.

From these results there was little evidence to suggest whether the microbial or the elite microbial performed better. The Elite microbial works better with smaller numbers of generations but, takes longer to run than the microbial. With a simulation the microbial offers better functionality as we can run many more trials in a short period of time. This is a particularily important consideration on a physical robot where each generation will take longer and therefore the elite version would be more suitable.

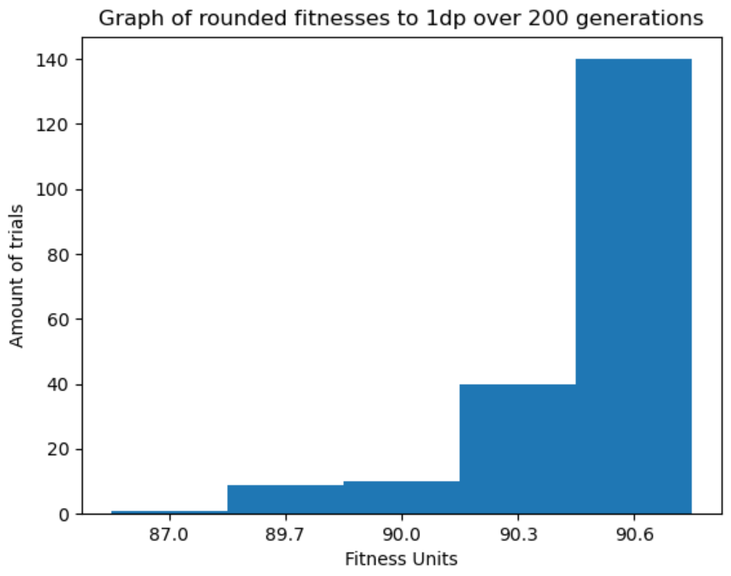
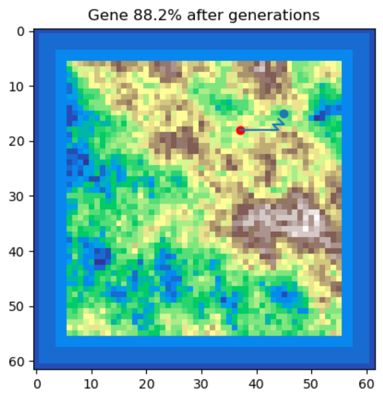


Figure 14: Distribution of rule-based agent performance. The average of 200 trials had a fitness of 88.9% which was much higher than the average fitness of the genetic approaches. This was displayed as a histogram and placed into five bins.

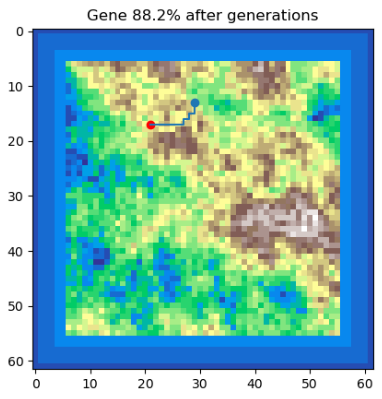
The rule based approach makes use of predefined logic such as avoiding water and picking the route that is in the direction of the end goal, unless the terrain climbing step is too high. The algorithm and instructions are unchanging, unlike the genetic approach where random mutations affect the outcome of the results. This method converged on much higher fitness than the genetic approaches without failure.

Because the rule-based algorithm doesn't require any training, in figure 14 we show the distribution of fitnesses of the rule-based algorithm as a histogram. This shows that the rule-based method performs better than the genetic approach in our simulations, suggesting that the neural network might be over-fitting the problem. Quantitative results show us how the algorithm performed overall within different scenarios. Qualitative examples clearly highlight how the best performing evolved.

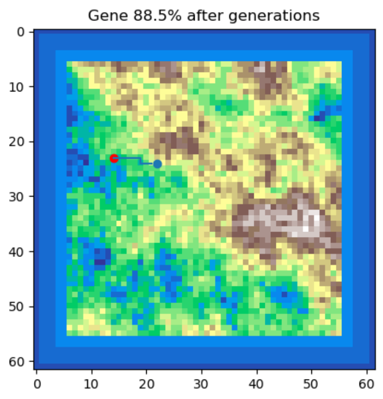
For the simulation environment, persistence was chosen at value 0.7 for the trials. This was because it generated more obstacles than lower values.



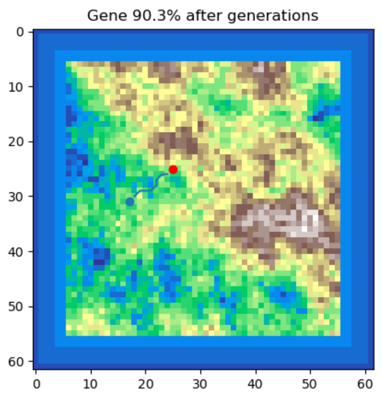
(a) Microbial evolved agent



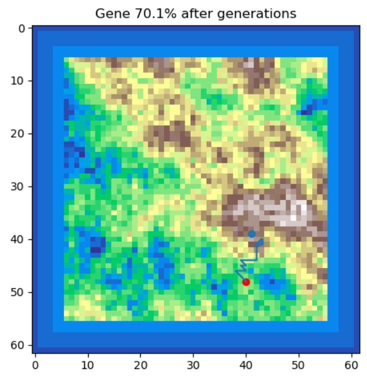
(b) Microial evolved agent



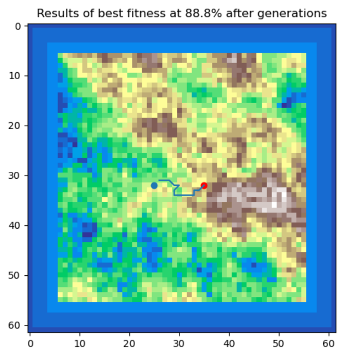
(c) Elite Microbial evolved agent



(d) Rule-based agent



(e) Convolutional 2D layer



(f) Convolutional 2D layer

Figure 15: Qualitative examples of simulated agent trajectories. The start position is denoted in a red dot, and the end target is denoted in blue.

Figures 15a and 15b show that evolved agents with high fitness follow contours across complex terrain. Travelling along contours uses less energy than following the shortest euclidean distance over complex terrain.

The rule based approach example shown in figure 15f demonstrates precise navigation around dangerous obstacles such as water pools.

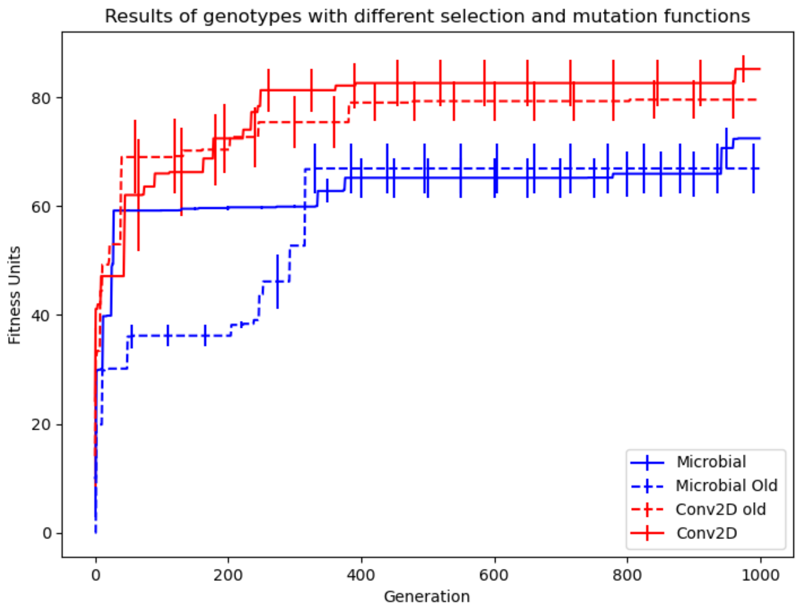


Figure 16: The results of the old and new mutation types on the fitness evolution. The old is denoted by the dotted lines. The error bars display the standard deviation between each trial at the given generation.

The old mutation would select two random genotypes from the population for trials. The new type only selects local neighbours in order to have competing genes within the population emerging. Using a mutation rate that selected neighbouring genotypes, rather than random indices within the population, had a positive affect on the algorithms. Figure 16 shows that this method always had a higher fitness. This selection method was used within the end model.

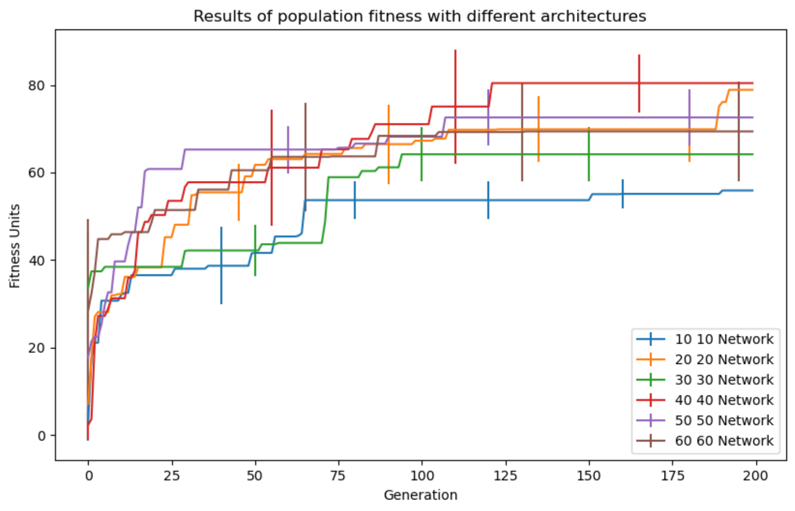


Figure 17: Architecture experimentation for the conv2D network. The values within the legend are encoded as "numInLayer1 numInLayer2 hidden layer". "10 10" means there were 10 nodes in layer one and 10 nodes in layer two. When using a convolutional layer, this refers to the linear layers that proceed the convolutions. The error bars display the standard deviation between each trial at the given generation.

A simulation was set up to test different network architectures using the 2d CNN network. 2D convolutional neural networks are popular choices within the image processing world. Images have high dimensionality and CNNs are effective with reducing parameters without loss of model quality. This happens by applying a kernel over an input image, which creates a new image out the other side of the convolutional layer. This image has reduced parameters. A series of hidden layer sizes were tested. To reduce the search space, I only considered configurations where these layers were the same size. Unsurprisingly, it was found that larger layer sizes performed better than smaller widths. Figure 17 shows that hidden layers of size 40 were the best for this task.

6 Vision

6.1 Stereo cameras

In order to obtain a depth image, similar to that produced by the simulation described in figure 7, I initially used a stereo camera setup consisting of two USB Arducam modules and extracted depth information using the OpenCV disparity function. Initially, the cameras were distanced along an aluminium profile to secure them on the same axis. The Arducam modules were chosen over the Pi cameras (which had similar quality) as the Pi only has one camera input through ribbon cables. For multiple Pi cameras, additional switch hats are needed, which would raise the cost and complexity of the problem. USB allowed simpler hardware and easy processing via the OpenCV library. Additionally, using USB cameras meant they could be directly connected to a desktop computer for testing. At this stage there was no need to use the control board of the Robot as it would be much slower than a PC.

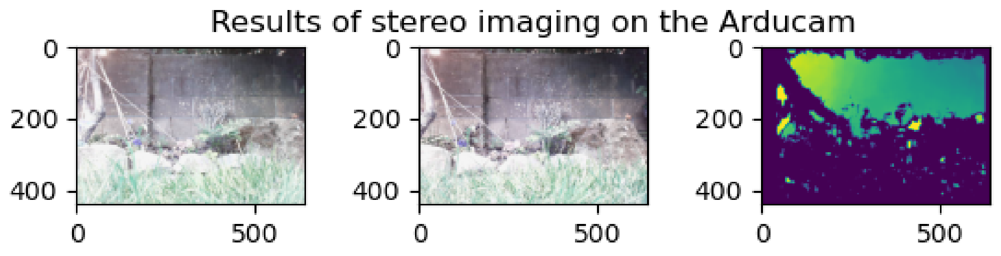


Figure 18: Disparity map generated using no pre-processing techniques

A disparity map was calculated using the OpenCV library but the initial results were disappointing, even with calibration. Calibration was attempted through changing parameters of block size and number of disparities. The larger the block size, the more noticeable features are. Noticeably in Figure 18 significant features were located but much of the image goes unnoticed. The wall in question was approximately 150cm away from the cameras, and the rocks were 50cm away.

Many of the issues came from problems within the camera position, where the cameras were never completely aligned. To avoid wasting time solving this, I instead switched to an X-box Kinect sensor which was fully calibrated and used a more precise time-of-flight sensors.

6.2 Kinect

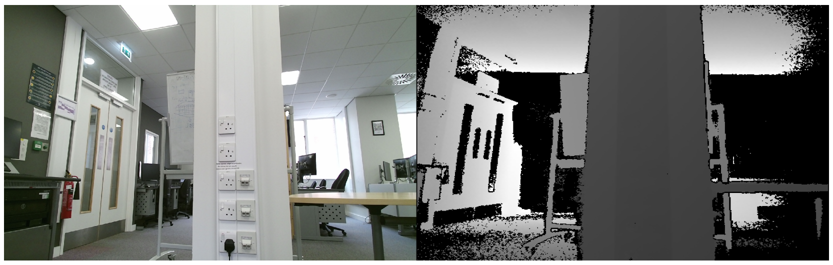


Figure 19: Depth images from the Xbox sensor, left normal image, right depth image.

Figure 19 presents how obstacles look in front of the image sensor. This is better calibrated than the previous camera system, although the sensor is much larger and mounting it on the robot would require a more powerful battery and a larger chassis. Therefore, this camera will only be used as a proof of concept for using depth navigation in a real world environment.

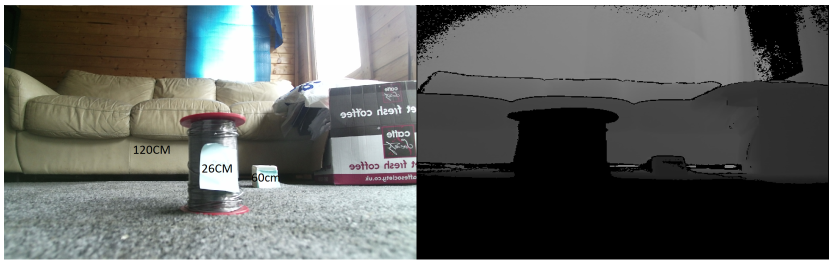


Figure 20: Xbox Kinect with measured distances of objects.

Three objects were placed in front of the camera at 26cm, 60cm and 120cm away. It is clear, based on their different colours that these three objects are at different distances. One issue is all the nearby ground appears as a large black box at the bottom of the image. The robot would need to be able to distinguish between ground going into the distance and a real obstacle. This is easily addressed by cropping the image and only looking at the centre where the agent is able to view the environment without the confusing details.

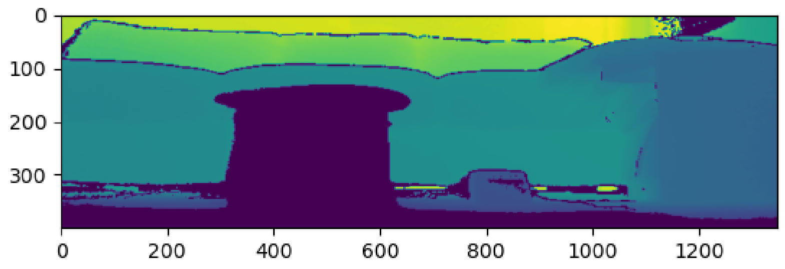


Figure 21: Xbox Kinect with measured distances of objects displayed via matplotlib.

Within my simulations, models had been trained to avoid hazardous terrain. Applying the depth imaging to these models allows testing the robustness of these models at crossing the reality gap. The image from the Kinect is cropped to a rectangle within the middle of size 200 by 400 and this is resized using the OpenCV function to an image of 5 by 5 to match the size of the views from our simulation.

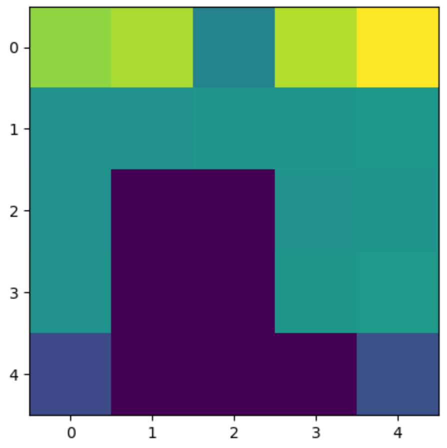


Figure 22: Reduced scale image for the model. The image used is the same in Figure 21.

Figure 22 shows that images produced in this way closely resemble those obtained from the simulation and the nearest object is highlighted clearly, while the background contours away. As there were small amounts of noise where the sensor had detected a background object as being closer to the lens than it was, I explored different interpolation methods for resizing of the image to find which one performs best.

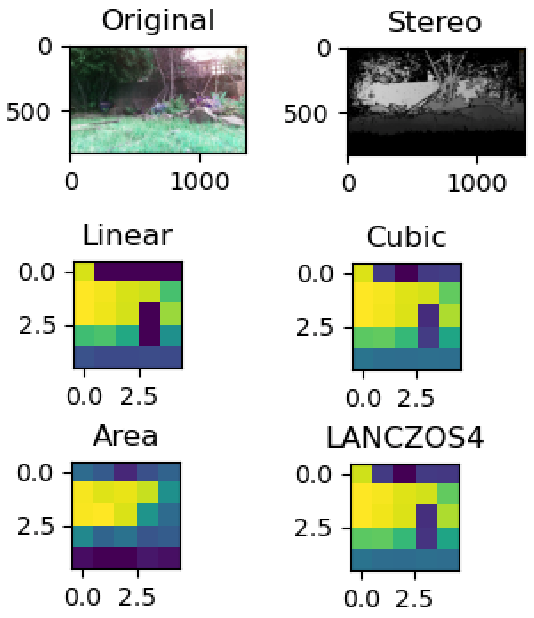


Figure 23: Qualitative example of noise creating problems within the generated image.

As figure 23 shows, linear interpolation highlights noise significantly which would risk the robot moving in directions which are not representative of the environment. Instead, area interpolation was chosen as it picked up both the wall in the distance *and* direct threats in front of the sensor. There were more rocks on the right hand side of the sensor, which is found in the darkening of the right hand side of the vision. Area interpolation works using the re-aggregation of data as sets of polygons mapped to the next polygons. The algorithm selects the nearest neighbour to a pixel and yields a piecewise-constant formation of new data. This keeps more general detail about an area compared to other approaches such as linear interpolation, which maps values to other values within a linear space.

6.3 Results

Different pre-processing was applied on the stereo images in order to highlight key areas of the images. Some techniques were more successful than others, but it transpired that using two cameras next to each other was not as accurate or calibrated as an off-the-shelf solution.

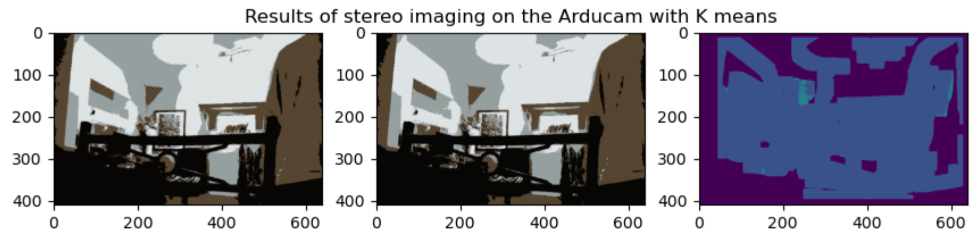


Figure 24: K-means clustering as a pre-processing technique on the images

As there was a too much noise within the images, I applied K-means clustering as this method has been used as a computer vision technique to reduce noise and reduce complexity for machine learning models. The hope was to reduce the noise and change within the stereo images. The parameters used were varied numbers of K, at 100 it took too long to process and found little, with K at 4 it took less time and found similar disappointing results.

This method was unable to distinguish the differences between far in the background and close to the camera. The basic structure could be seen in a room but no finer structure is extracted. Furthermore, the K-means reduction lead to a blank result on the outdoor trials. This suggests that the reduction did not reduce noise in the desired way. Stereo vision relies on the changes in pixels, which cannot always be found with k-means reductions. Therefore, this method will not be appropriate for pre-processing.

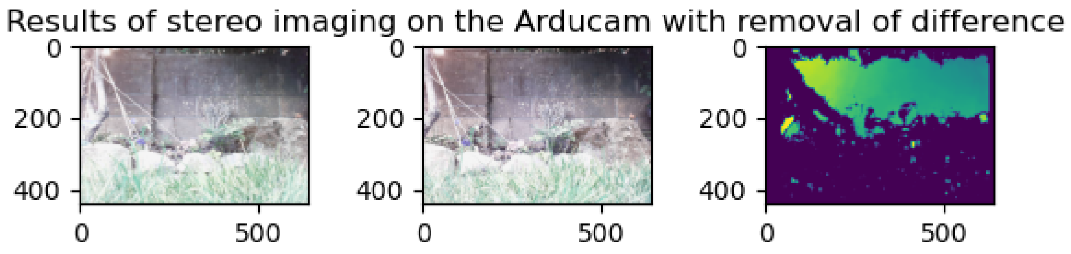


Figure 25: Removal of different pixels in image

Figure 25 shows the result of removing the pixel differences within two images (before and after) in order to remove the amount of noise. This was performed using the structural similarity function from the *skimage* library. The intuition behind this was it would scale down noise and only keep the features that were relevant and significant. This was successful as it removed noise, however not to the significant that makes it worth the extra processing step.

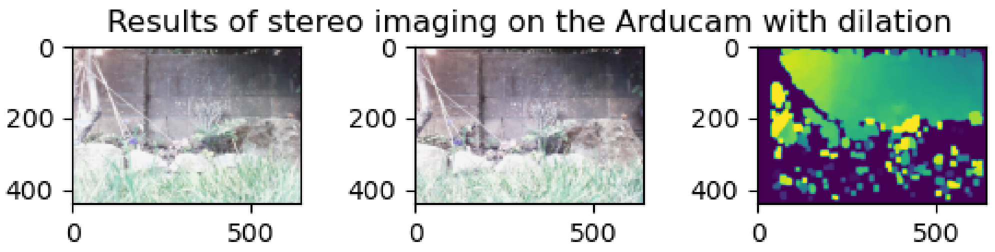


Figure 26: Dilation of image after disparity

Dilation is a morphological transformer which uses a kernel to increase the true values within a binary image. This works by generating the disparity map and then expanding around the area of this pixel for set iterations. As shown in figure 26, dilation of pixel areas is more successful at identifying hazards while removing noise. However, it could not accurately distinguish between the foreground, middle ground and background. The middle ground is ignored which is likely down to camera calibration problems. After a disparity map is generated, it is seen that only small areas can be valued significantly, however, shape can be seen. This is evident in Figure 18 where the wall and arm raises significance. The OpenCV library provides the features for this. It is clear that the use of morphological operators help to show the clearer routes through an image and avoid structure.

For vision-based navigation, the robot made use of a depth camera. The Xbox Kinect was used to gather depth imaging of the environment. This was reduced in scale to 5 by 5 pixels to highlight key features about the terrain. Using this image, the trained agent within the simulation was deployed to use a convolutional neural network. This was because the Conv2D network architecture had the best accuracy within the given trials.

The chosen visual sensor was too large to fit directly onto the chassis, therefore the sensor had its video recorded while being manually carried. The model was tested using these gathered frames and plotted as example based figures within the results. As there are multiple directions that can be valid within these examples, using numeric based validation methods was not possible without mass labelling the data. This would still be down to human judgement. Instead, human judgement over the examples will be the evaluation method of how well the model performed in the real world. The network was trained to predict a vector of movement out of eight possible moves, but as this was not controlling the robot, so the vector is simply displayed as the chosen direction of travel that should be taken.

Evolving this navigation model directly on the robot could lead to better results. This is because the model will learn how its own hardware and control deals with different sized obstacles, and which obstacles the chassis could overcome with back-bending. This method was not used because enforcing reward posed too many challenges. Knowing when the robot has collided, got stuck and used energy requires a number of sensors. Additionally, it would have taken a large amount of time to perform just one trial. Simulation mostly converged on peak fitness at generation 400 to 800. Additionally, if the simulated model performs well it will have shown the model to be robust enough to cross the reality gap between simulation and physical implementation.

The reduction of the image to a 5 x 5 pixels input for the model was applied to both depth images. One generated by the Kinect and one generated by the OpenCV disparity function from two USB webcams. Based on my previous results, both used the area interpolation resizing method.

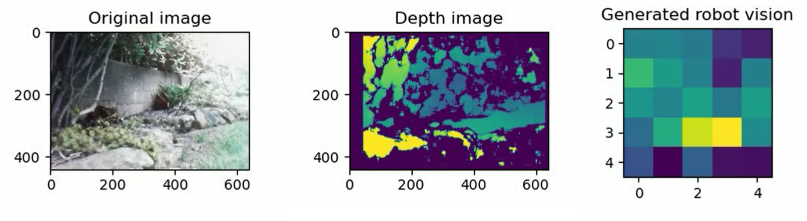


Figure 27: The generated robot vision image on outdoor terrain using the USB Arducam with the OpenCV disparity function.

The image from the stereo cameras wasn't able to pick up the clearer direction compared to the Kinect in Figure 28. The Kinect had smoother disparities than the OpenCV image. This lead to a smoother robot vision image than in Figure 27. Its likely that the OpenCV generation would lead to more chaotic predictions if used on the trained model, as there is a large amount of noise.

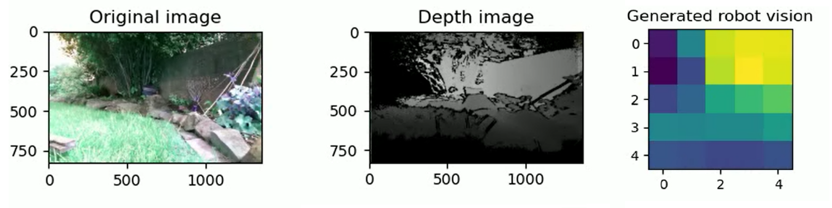


Figure 28: The generated robot vision image on outdoor terrain using the Kinect sensor.

In conclusion, the Kinect sensor is much better at detecting depth and filtering out noise than the USB Arducam webcams with OpenCV. Therefore, the Kinect sensor or another off-the-shelf depth imaging system will be used for deployment on the physical robot.

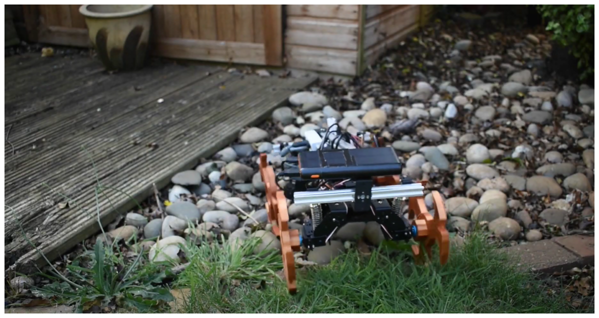


Figure 29: The robot traverses over pebbled environments with a wooden obstacle where the ground texture changes.

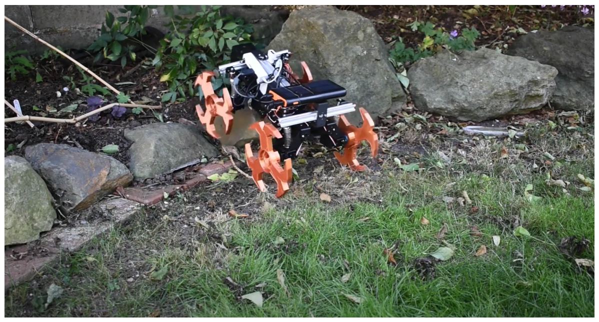


Figure 30: This figure shows the robot attempting to climb over larger rocks.

The outdoor environment shown in Figures 29 and 30 give qualitative examples of the types of obstacles that the agent will face. The larger rock is the largest challenge due to the non-linearity of movement and different forces acting on the chassis.

Evaluation of how the simulated trained model reacts to the real world robot vision is denoted by the following qualitative examples. These show the original image, how the depth sensor views this, how the robot views this and the direction the model predicts. The vector is in the direction of movement of the line triangular marker. The unmarked end is the start position of the vector.

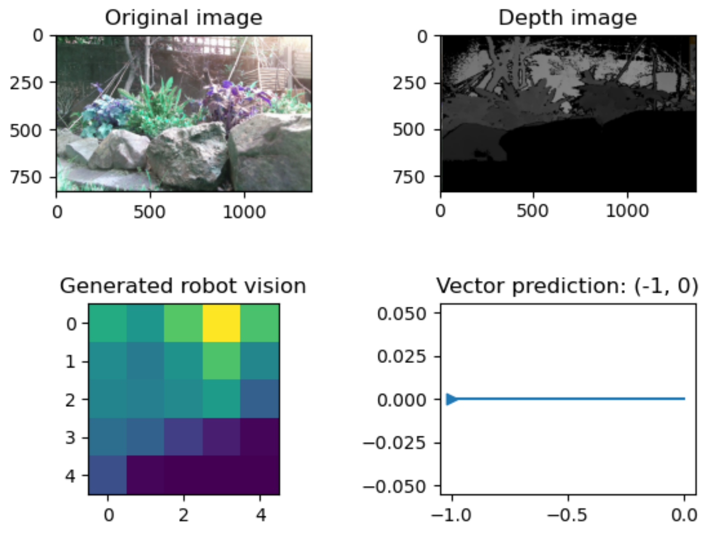


Figure 31: This example shows where the depth sensor has picked up a large amount of rocks in front, and a continuation of this obstacle moving across.

This terrain in Figure 31 is deemed too hazardous by the model, where the vector tells the agent to turn sharply left. This is considered a success case.

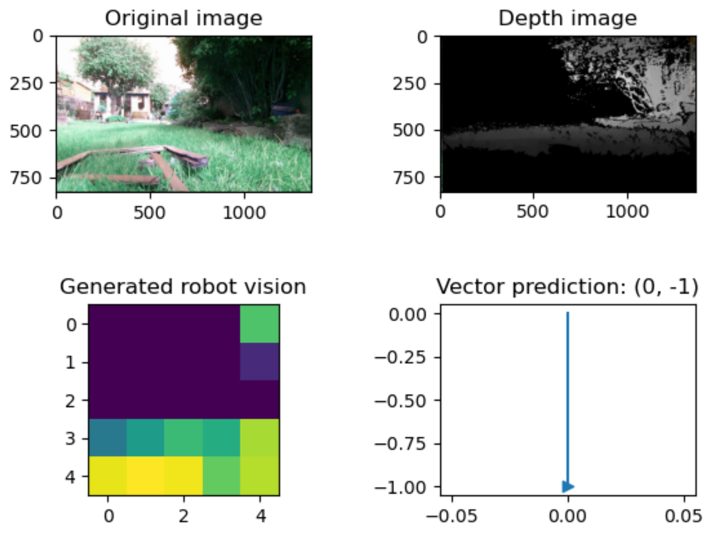


Figure 32: Robot vision detects that there is a large obstacle in front of the current space.

The sensor has picked up the distance as nearby in Figure 32, despite it being far in the distance. This is a failure case for the sensor, however, it is a success case for the model where it has successfully predicted to turn back when there is no terrain to move to. This suggests that the model makes use of a form of spatial-temporal reasoning to know there is no point moving forward, despite the immediate ground being free.

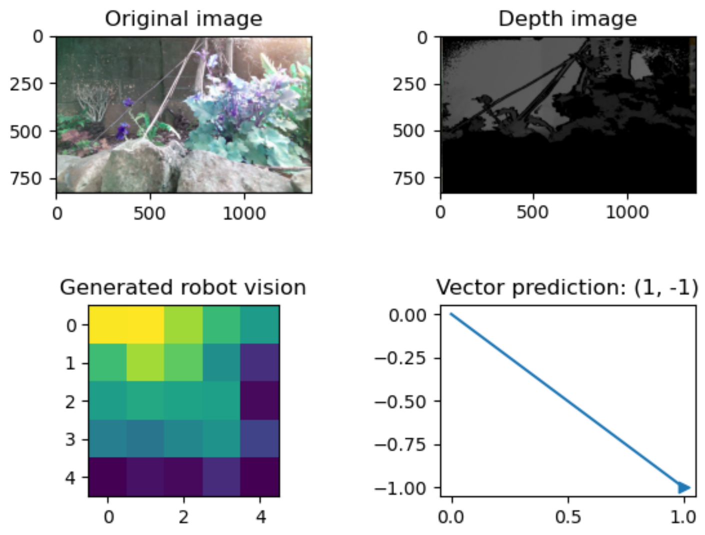


Figure 33: Model move backwards from immediate threat.

Figure 33 detects the immediate obstacle, however moves in the wrong direction. The direction chosen wouldn't be a failure for the robot, but is not the optimal direction for traversing the terrain. It is likely the model has biased this side as the vector of movement, as input vector to the network was set to (1,1). This means the model is trying to find the end destination by turning right towards it.

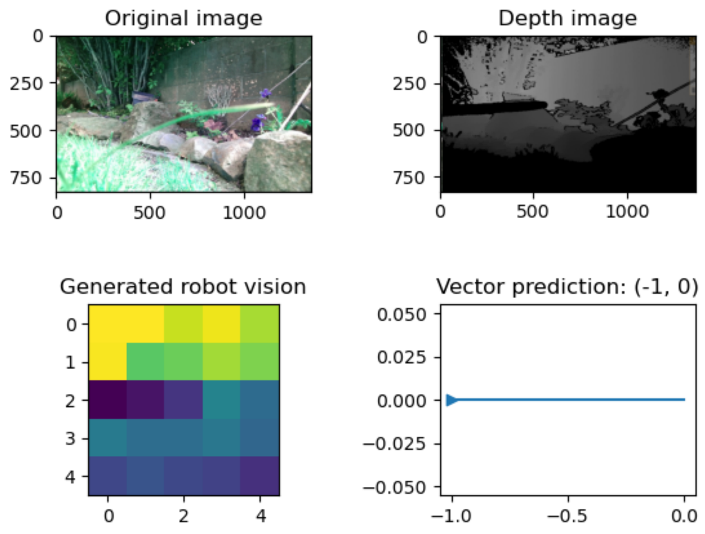


Figure 34: The sensor has picked up a nearby piece of grass an obstacle. The model has chosen to ignore a small obstacle.

Figure 34 is fail-case of the model. The model is going left and avoiding difficult terrain within the immediate ground, however has been given an input vector of (1,1) which is directing it through the less terrain-complex route. The model has not taken this route, and instead has chosen the more complex route.

Overall the model has been deployed on real life imagery and performed well. Through the qualitative examples there was not one prediction found that would have harmed the robot. This implementation has been able to cross the reality gap for stereo-vision based navigation. The reduction of the search space by using a 5 x 5 image enabled the removal of real world noise. One potential problem holding back the crossing of the reality gap was that the robot in the simulation was set-up to be able to turn on the spot. The chassis built for this project required rotation of the front Whegs to move left and right. This should not be a problem as each input image is independent of the last, however testing this would be the work of future research.

Though the model isn't necessarily the most efficient, it does keep the agent out of danger.

7 Robot platform

7.1 Chassis

The prototype version of the chassis used aluminium profiles with a lithium ion battery held in the centre under pressure. Continuous rotation servos are attached to the sides of the profiles.

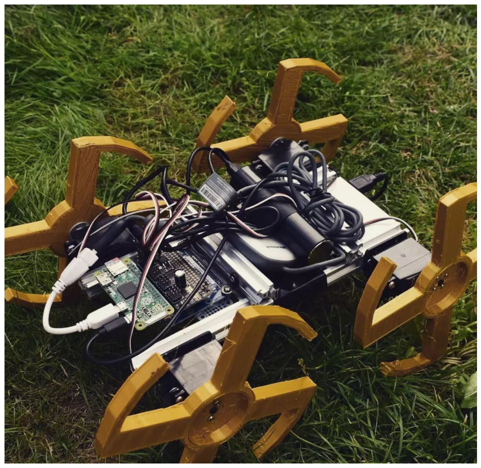


Figure 35: Wheg robot prototype (version 1)

This robot would get stuck over obstacles where the weight could not be redistributed. A bendable back actuator was implemented to the chassis to redistribute the weight. Additionally, suspension was introduced to absorb shock and prevent tilt of the chassis over rocks. These alterations should solve the problems of the version 1.

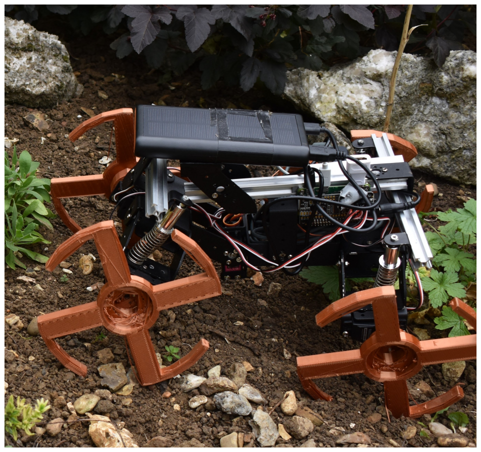
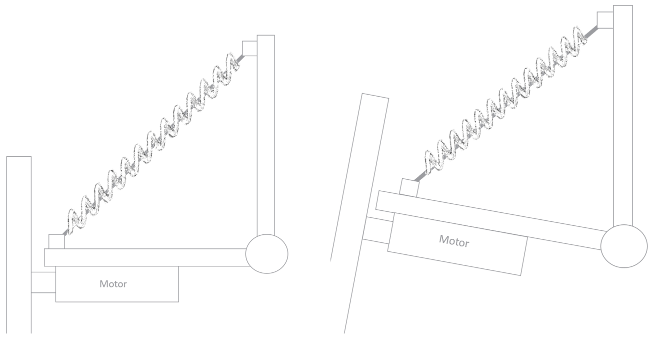


Figure 36: Wheg robot version 2 introduction of back actuators and suspension.

The chassis was built using aluminium servo parts, typically used for biped robots and robotic arms. These not only easily fit standard servos such as the continuous rotation servos used for the robot's locomotion but, in addition, made it easy to fit suspension and the cockroach inspired back. Figure 36 shows the robot within outdoor trials. The robot chassis also incorporated a Wheg design, enabling it to climb over terrain while maintaining stability under initial tests with an Xbox controller. The main chassis of the robot used the suspension system shown in Figure 37 so that a higher incline on one extremity does not tilt the overall robot. This design was inspired by a Tamiya radio controlled car [24].



(a) Neutral position (b) Activated position

Figure 37: Figure showing the movement of the suspension system design moving over hardware. This uses a single spring shock absorber per Wheg.

The initial Wheg design would get caught while reversing as the claws were only grabbing one way and the robot also struggled to turn left and right on the spot. The addition of a tilting mechanisms for left and right turns was added to solve this. In some circumstances, when the robot would try to climb, it would need an element of height to get over obstacles. This was achieved by adding a stabiliser servo with a wheel caster on the bottom. When deployed, the wheel caster allowed the robot to carry on moving without getting caught. The heavy battery was placed at the back of the robot as it was discovered that the front needed to be lighter than the back to have improved climbing abilities.

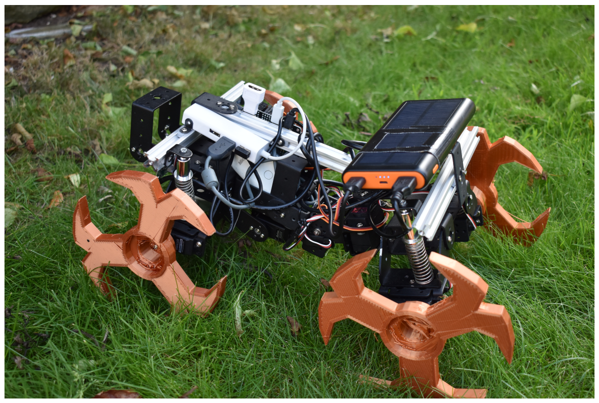


Figure 38: Wheg robot version 3 with improved weight distribution, stabiliser and better whegs.

The neck used a pan and tilt mechanism which was placed at the front of the robot chassis and panned left to right, to improve turning left and right within an environment. The tilt will be up and down on the axis of gravity acting as a back. A bend in the back will redistribute weight – the exact process applied in the cockroach research [8].

For control, the chassis used a Raspberry Pi for its low energy consumption, yet reasonable processing power. The Pi zero was chosen as it is the lowest consumption of power for Pis on offer, while being small and light in physical size. It was also setup with Linux, which allowed SSH control of the robot and ease of gathering results from trials. The configuration of the Raspberry Pi used a 32GB SD card to give plenty of expansion space. Due to the use of neural networks and computer vision, having a large SD card allowed RAM over-spill into virtual memory, which would be much slower but would prevent the experiment being cut short. On top of the Pi there is an Adafruit servo driver that allows up to 16 servos to be attached. The Raspberry Pi allows the stacking of hats which provides ease of connection and reduction of wires compared to using external modules. The Adafruit servo hat has vast online support and documentation, making it well suited for this project. One challenge of using the Raspberry Pi Zero is that PyTorch is harder to install on ARM processors than on standard desktp machines. It is worth noting that the Pi Zero will only compile PyTorch arm python wheels that end with 'linux_armv6l.whl', however, the GitHub build for this file was not compiled with NumPy. The solution was to download the 'linux_armv7l.whl' version and rename it to 'linux_armv6l.whl' before running it.

The continuous rotation servos used 100ma while active and the back actuation servos 500ma. In a worst case scenario we assume that all current is needed throughout each servo motor. This is denoted by the following calculation:

$$(4 \times 100) + (2 \times 500) = 1400ma \tag{7}$$

The battery chosen provided this current with the additional current the the Raspberry Pi requires as a minimum. An error found in testing earlier models was where the current draw was too great which made the Pi reboot or would restart the USB drivers.

On the front of the robot two cameras were mounted for stereo vision. The camera sensors changed throughout the design process, but through the use of aluminium profiles, components could easily be interchanged. A gyroscope/accelerometer MPU-6050 sensor was mounted on the front to detect unexpected incline. There is the additional option of current monitoring or wheel encoding to detect whether there is movement within a wheel. Current monitoring is more suitable than wheel encoding for this task for two main reasons. Firstly, embedding encoding requires additional hardware as well as modification to the chassis. Furthermore, if the chassis is suspended in the air, the wheels will rotate despite having no affect to the chassis.

7.2 Back actuation

As described in section 3.2, organisms like cockroaches can bend their backs to increase climb. Research on cockroaches explored the use of back bending with cockroach-inspired limbs on a robotic chassis [8]. In this robot, back bending was controlled by a servo motor which introduced a pan-tilt style movement. The movement of the back was performed on one axis rotation. The other tilt helped the robot turn left and right while travelling around an environment. The back bending proved successful while controlling the robot manually as, while the chassis previously got stuck at points within an environment, the redistribution of weight via back actuation allowed climb, preventing obstruction. This can be seen in the qualitative example of Figure 40.

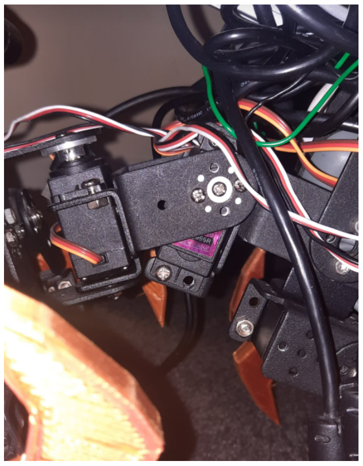


Figure 39: The back rotation and turning system in the centre of the robot chassis. One for direction and one for the back. These two servos were responsible for rotating more weight per servo than any of the locomotion joints. This meant they were of heavier duty standard thus drew more current.

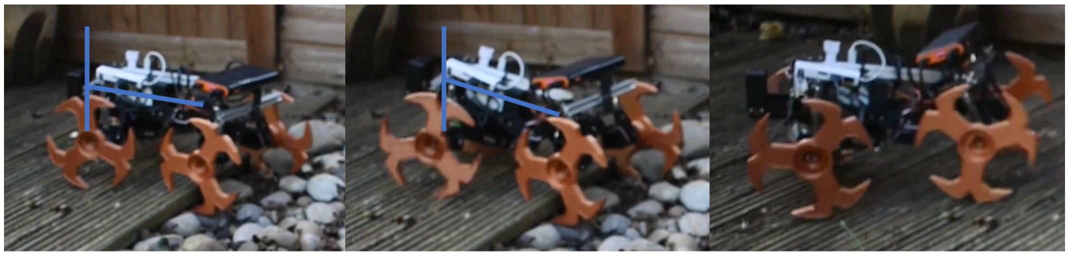


Figure 40: How the angle on the robot changes with back actuation. The first image is where the robot got stuck going over a barrier. The second image shows the distribution of weight to the centre of the robot, which allowed more torque to be focused on the back wheel rotation rather than holding the weight of the robot. The final image shows the robot having climbed over the obstacle. Outlined in Figure 40 is a drawn axis in blue to demonstrate the movement undertaken by the robot.

Figure 40 was produced by driving the robot manually using the Xbox360 remote and a USB dongle. This allowed proof of concept that the robotic back was useful for the task. However, autonomous movement requires detection of being stuck. Acceleration and tilt within a direction can be measured with the MPU6050 sensor. Tilt in the axis direction of up for the MPU6050 positioning on the robot will be the x-axis. Acceleration is measured and movement can be detected. These two features make the MPU6050 a candidate for this task. The sensor itself is mounted on the front of the robot, as this will be the place that first detects tilt when going over rough terrain.

A gyro class was created that could be imported into the main Wheg robot code. This is used to trigger when an obstacle is hit, and will get predictions from a neural network model for how to overcome such obstacles. The steepness of the ground can be extracted from the sensor by low-pass filtering the accelerator vector and this could also be integrated to calculate velocity.

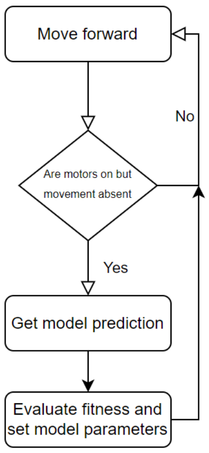

Move forward
No
Are motors on but movement absent
Yes
Get model prediction
Evaluate fitness and set model parameters

Figure 41: The back bend model is only needed when an obstacle is predicted. The task requires evolution of the model. Through initial trials, the algorithm is used to evolve movement over time by taking motor positions and gyroscopic data as input, and motor movement as output. The genetic process for evolution will use similar approaches as the navigation.

Initial evaluation of the accuracy of this approach showed that the accelerometer produced a lot of noise and therefore could not be trusted for accurate collision detection. This idea was abandoned and replaced with a method that exploited spikes in the current caused by the motors being unable to rotate as the servos would stall and draw high current. As example current trace is shown in Figure 42.

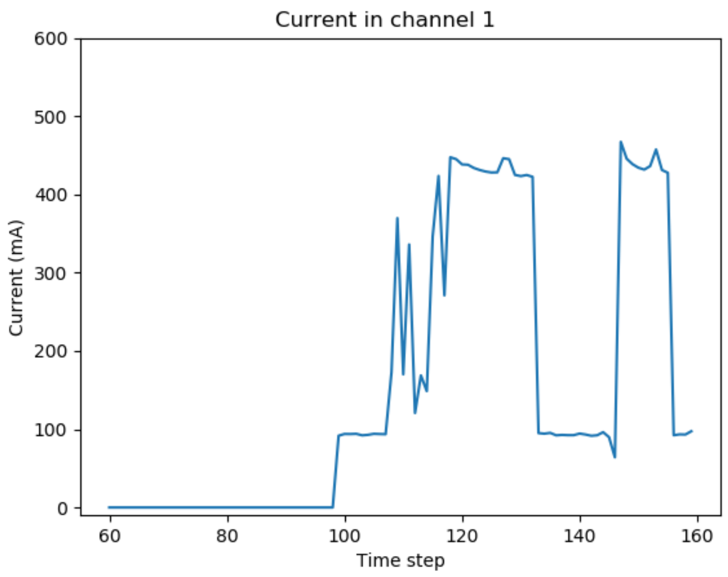


Figure 42: Display of the current read from a rotating servo attached to the robotic chassis. At different time steps, different amounts of pressure were applied to the servo.

Between the first time step and time step 98, no signal had been sent to the motors for rotation. After times step 98 a signal was sent and the current draw is working at approximately 100mA. This is expected as the current draw of a servo is 110mA. At time step 110, the robot is lowered onto a surface but some of the weight is maintained by the motor. The Whegs mostly rotate with some disturbance to speed and occasionally get stuck. At time step 120, the robot is placed fully on the ground with weight preventing it from moving. The current dramatically spikes up to approximately

450mA. Between time step 120 and 135, the robot was picked up, and placed back down. This is why the current draw decreases at this interval. Based on these measurements, I converted the current readings to an array, then counted how many times the current exceeded the threshold of 0.4mA. If this was found to exceed 30% of the readings taken, the robot would be registered as stuck. 0.4mA was chosen through a trial and error process. 0.1mA is the expected current, with added noise depending on the complexity of the terrain. Anything above 0.4mA means the motor is not moving. The I2C address of the servo hat had to be changed to 0x44 from 0x40 due to the power monitoring hat using address 0x40 for channel 1. This was achieved by soldering bridges on the PCB address bar. Using the command prompt and changing the default parameters to the library, the new address was found. If this was not done, the motors would spin out of control due to mixed signals causing chaos.

The algorithm used to generate movement was the hill-climber. When the robot is stuck it will need to get unstuck, and a microbial needs to perform two genotypes in the same environment. If one genotype gets the robot unstuck, then the second is not needed and causes an unfair trial. The hill-climber can modify itself on different trials to get the best outcome. As the search space is low, the hill-climber is unlikely to have convergence problems.

The hill-climber uses a genotype that represents the weights and biases of the neural network. This neural network is contains linear layers taking in the current motor positions, the MPU-6050 data and then predicts whether to move the back up or down using the argmax function on two nodes.

7.3 Proprioception

The physical robot can sense its own state through a series of sensors. The robot can sense whether wheels are stuck through a rise in current being supplied from the battery. Every motor state is tracked through a Python library, which means angles of standard servos is known as well as the expected speed of the continuous rotation servos. The angle tilt of the robot is known using an MPU-6050. When the robot has tilted to move over an obstacle, the robot is able to use this information with motor angles as input to the back bending neural network. As output there are two nodes to decide whether to move the back up or down using the argmax function. For the robot to know whether the network had performed well, a fitness function was written. This made use of the knowledge that 0.4A was the highest these motors would raise to, and 0.1A was the default spin. The fitness would perform the following, where C is a set of all the current reads from each continuous rotation servo:

$$Fitness = \frac{(0.4 \times 4) - \sum_{i=1}^{4} C_i}{0.4 \times 4} \times 100 \tag{8}$$

The value 4 represents how many motors there are. If only two motors are being read from the robot the value would be 2. The multiplication by 100 converts the value to a percentage.

7.4 Results

Back actuation improved climb over terrain, which can be seen in the qualitative example of Figure 40. Redistribution of weight allowed more torque to the front Whegs. The physical agent was trialed over different terrain. In order to prove the concept of back-bending and obstruction detection, the robot was placed on a flat surface and it would be physically held onto the floor to detect being stuck. Once detected the robot would be released. This performed well, though would on occasion be delayed at detecting it. This is likely due to the event loop creating delays while listening for instruction from the Xbox remote. The heuristic performed without failure when it was trialed.

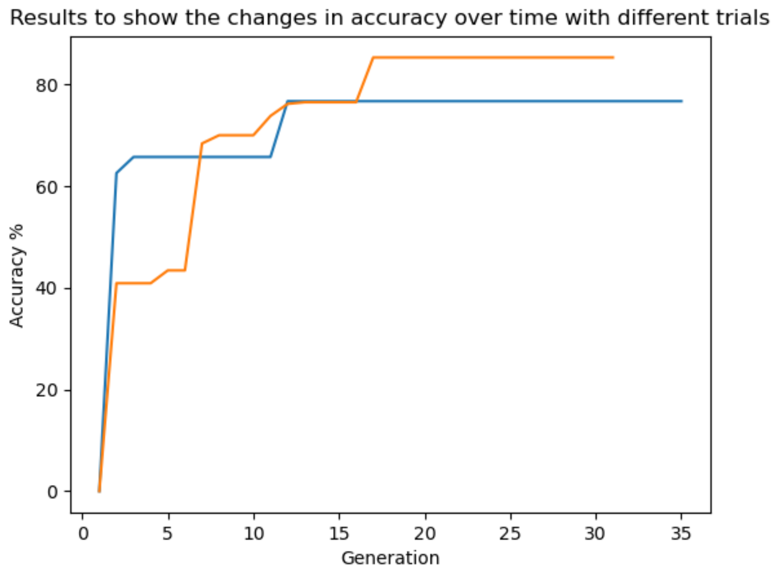


Figure 43: Genetic algorithm back bending accuracy over generation.

Solutions using the hill-climber algorithm for evolving the back converged on high fitness quickly. By 15 generations at least one genotype was close to maximum fitness. This is likely due to the small search space that this problem has. Small changes within the back actuator angle can provide successful locomotion over terrain.

The final environment was the outdoors. This was a dynamic environment where pebbles and rocks could easily slip and change throughout testing. This environment was used to evaluate how well the robot performed in the real world.

8 Future work

One of the main limitations between robotic navigation and simulated navigation was having a reliable depth sensor. The most reliable sensor the project had access to could not fit onto the robot for secure mounting. Additionally, the sensor used a high amount of energy that the robot battery could not provide. Future chassis would need to use smaller and energy inexpensive stereo sensors. This would allow more rigorous testing of the robot. The Intel Realsense is a small stereo image developed for robotics. This is much smaller than the Kinect. There is also the ZED2 AI-vision camera that uses neural models to predict depth sensing. This has been deployed on a number of industrial applications.

Additionally, it was a mistake to use the Raspberry Pi Zero for such a task. Though it worked well with the battery and interfaced with all the given hardware, it was slow and required a lot of extra peripherals attached to it.

The sensors available for the robot were limited to stereo vision. This had calibration issues with a dynamic environment. Using sensors such as LIDAR would allow more accurate representations of an environment. This would be the work of future research.

A critique of the methods carried out within this project would be everything was done on the same chassis. In hindsight it would have made sense to make smaller and lighter chassis, starting with a small chassis for stereo navigation. This would have been more energy efficient and allowed more testing time. After this, a back bending model would have been introduced to experiment with a working navigation model.

9 Conclusion

This project successfully showed that an agent can avoid collision with environmental hazards by understanding its limitations. Evolved and rule-based navigation were both successful approaches to this. Rule-based was more reliable at traversing terrain than evolved navigation. However, the evolved navigation made use of terrain contours for energy efficiency. Additionally, the rule-based agent fitness was not significantly higher than the evolved fitness. The agent was able to recognise hazardous terrain through trials of crashing into it. The goal of this project was for an agent to understand its own limitations, which was achieved.

Simulated trials proved the concept that this paper aimed to develop. The agent used depth imaging which was also implemented on the physical robot. The real world depth imaging was not as clear as the simulated depth imaging, but this was expected due to real world noise. Crossing between simulated and physical implementation, referred to as the reality gap, posed many challenges. The robot had a larger search space due to real-world noise, thus increasing complexity of converging on a solution. The physical robot would have taken a long time to converge on solutions that the simulated agent found. Instead, the image input was reduced in order to be more representative of the simulated information, thus reducing the real world noise.

The stereo imaging using the Kinect sensor gave the best representation of the depths of an environment. This was reduced to the scale of a 5 x 5 image to highlight key obstacles in the terrain. Using a trained agent model from the simulation, it was successfully demonstrated that the reality gap could be crossed for physical robot navigation. The predictions from the model made the agent avoid large obstacles such as rocks in an outdoor environment.

Using a biologically inspired chassis improved the robot's ability to traverse terrain, which was expected. The locomotion features of cockroaches were successfully implemented using 3D printed Whegs and a bendable back. This allowed the chassis to climb over rocks and redistribute weight to improve torque at separate parts of the robot. A back that can rotate for an off-road chassis shows great potential for future 4-wheeled drive robotic locomotion. The back actuation produced a method of feedback for learning which terrain the chassis should avoid. If back movement does not allow the chassis to get unstuck, the agent has met a limitation. Therefore, terrain such as this would be avoided by an agent through evolved navigation.

References

[1] James B. Wood and Roland C. Anderson. Interspecific evaluation of octopus escape behavior. *Journal of Applied Animal Welfare Science*, 7(2):95–106, 2004. PMID: 15234886.

[2] Tamir Blum and Kazuya Yoshida. Ppmc rl training algorithm: Rough terrain intelligent robots through reinforcement learning, 2020.

[3] T.-T. Lee, C.-M. Liao, and T.K. Chen. On the stability properties of hexapod tripod gait. *IEEE Journal on Robotics and Automation*, 4(4):427–434, 1988.

[4] Priandana, Karlisa, Buono, Agus, and Wulandari. Hexapod leg coordination using simple geometrical tripod-gait and inverse kinematics approach. In *2017 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, pages 35–40, 2017.

[5] Schwendner, Jakob, Grimminger, Felix, Bartsch, Sebastian, Kaupisch, Thilo, Yüksel, Mehmed, Bresser, Andreas, Akpo, Joel Bessekon, Seydel, Michael K.-G., Dieterle, Alexander, Schmidt, Steffen, Kirchner, and Frank. Cesar: A lunar crater exploration and sample return robot. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3355–3360, 2009.

[6] R.D. Quinn, Offi, J.T., Kingsley, D.A., and R.E. Ritzmann. Improved mobility through abstracted biological principles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 2652–2657 vol.3, 2002.

[7] European Space Agency. Prolero. https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Automation_and_Robotics/PROLERO.

[8] R.T. Schroer, M.J. Boggess, R.J. Bachmann, R.D. Quinn, and R.E. Ritzmann. Comparing cockroach and whegs robot body motions. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, volume 4, pages 3288–3293 Vol.4, 2004.

[9] Melanie Mitchell. *An introduction to genetic algorithms.* MIT press, 1998.

[10] Inman Harvey. The microbial genetic algorithm. In George Kampis, István Karsai, and Eörs Szathmáry, editors, *Advances in Artificial Life. Darwin Meets von Neumann*, pages 126–133, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[11] Mohammad Reza Bonyadi and Zbigniew Michalewicz. Particle swarm optimization for single objective continuous space problems: A review. *Evolutionary Computation*, 25(1):1–54, 2017.

[12] J. Pijuan, M. Comellas, M. Nogués, J. Roca, and X. Potau. Active bogies and chassis levelling for a vehicle operating in rough terrain. *Journal of Terramechanics*, 49(3):161–171, 2012.

[13] Thomas Thueer, Ambroise Krebs, Roland Siegwart, and Pierre Lamon. Performance comparison of rough-terrain robots—simulation and hardware. *Journal of Field Robotics*, 24(3):251–271, 2007.

[14] Marc Raibert, Kevin Blankespoor, Gabriel Nelson, and Rob Playter. Bigdog, the rough-terrain quadruped robot. *IFAC Proceedings Volumes*, 41(2):10822–10825, 2008.

[15] Dominik Belter and Piotr Skrzypczyński. Rough terrain mapping and classification for foothold selection in a walking robot. In *Institute of Control and Information Engineering, Pozna ́n University of Technology*, 2011.

[16] Kahlouche Souhila and Achour Karim. Optical flow based robot obstacle avoidance.

[17] Yang Wang, Peng Wang, Zhenheng Yang, Chenxu Luo, Yi Yang1Wei Xu, and Baidu. Unified unsupervised optical-flow and stereo-depth estimation by watching videos.

[18] S.M.Abid Hasan and Kwanghee Ko. Depth edge detection by image-based smoothing and morphological operations. *Journal of Computational Design and Engineering*, 3, 02 2016.

[19] Marko Bjelonic1, Timon Homberger, Navinda Kottege, Paulo Borges, Margarita Chli, and Philipp Beckerle. Autonomous navigation of hexapod robots with vision-basedcontroller adaptation.

[20] SK Chiu, D Araiza Illan, and KI Eder. Bio-inspired self-preservation to protect robots from threats. in towards autonomous robotic systems: 18th annual conference, taros 2017, guildford, uk, july 19–21, 2017, proceedings (pp. 166-181).(lecture notes in computer science (including lecture notes in artificial intelligence).

[21] Ken Perlin. Improving noise. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, page 681–682, New York, NY, USA, 2002. Association for Computing Machinery.

[22] M. D. Malkauthekar. Analysis of euclidean distance and manhattan distance measure in face recognition. In *Third International Conference on Computational Intelligence and Information Technology (CIIT 2013)*, pages 503–507, 2013.

[23] Nicholas Tomko, Inman Harvey, Nathaniel Virgo, and Andrew Philippides. Many hands make light work: Further studies in group evolution. *Artificial Life*, 20(1):163–181, 2014.

[24] Tamiya. Rc car datasheets. https://www.tamiya.com/english/rc/chassis/cc-02/index.htm0.

10 Appendices

All code, models and files can be found on my GitHub: https://github.com/shepai/Dissertation
Included within the folder are the main aspects of the project.

agent.py - The main code to generate an agent that can use different network architectures set as parameters. Used for genetic algorithm optimisation.

MicrobialConv2D.py - This uses the microbial algorithm with a convolutional neural network to evolve the weights and biases of an agent moving round procedurally generated terrain.

Microbial.py - This uses the microbial algorithm to evolve the weights and biases of an agent moving round procedurally generated terrain.

Group.py - Uses a genetic approach where agents are evolved in sub populations rather than just one population.

Explicit.py - Uses a rule based approach to moving around the simulated world.

EliteGroup.py - Keeps repopulating with the best genotypes at the end of et generations.

wheg.py - Communication with the servo motors for physical implementation.

back.py - Evolving back bending based on Xbox remote feedback.

current.py - senses the current and plots it.

distanceSense.py - Looks through a given depth video and processes the images to 5x5 for model prediction.

Bio-Inspired Robotic Navigation On Varied Terrain

Dexter R Shepherd [candNo: 215819]
Supervisor: Dr James Knight

Interim report
Computer Science and Artificial Intelligence BSc

Department of Informatics
University of Sussex
2021

Contents

1 Introduction

In this project, I am going to investigate autonomous robotic navigation through localized decisions based on previous experience. Nearly all organisms are limited at some point when it comes to movement. For example, an octopus can fit through an aperture that is bigger than its beak [1] and a person can climb over rocks, but only if they have grip points for hands and feet. These environmental barriers define constraints to an organism's ability to traverse across it. When a path looks too arduous, we will often choose a more accessible route if possible.

These constraints also apply to robots. Consider a Mars Rover over 395 million km away from the person controlling it. If the Rover were to get stuck, there would be no one to recover it and – as it takes 181 seconds for a signal to get from Earth to Mars – real-time control is impossible. Therefore, a robot, like a biological organism, needs a sense of self-preservation so needs to know its limitations and not attempt tasks of movement outside these physical constraints. [2]

Our agent will be set a target location and will need to reach it while avoiding hazardous/overly complex terrain within the environment. Success will be defined as whether the agent got there safely, and how much extra energy was consumed. When deciding on routes, there may be the scenario that all paths are within the constraints but some are simpler to navigate than others. In this situation, the agent will need to pick the option which requires the "least" effort. The best-case scenario for the agent I am developing is governed by the level of complexity of movement necessary to overcome the obstacle. If the agent gets stuck it will get no reward, if the agent makes unnecessary movement while moving over simple terrain, it will get little reward.

I will initially explore this problem using simulation before moving to a robot platform with a back actuator, stabilizer and neck actuator. The robot will need to learn to use these features to help it navigate over the landscape. This robot will perform this task by using computer vision techniques, which forms a prediction on the best route to take via a movement instructions generated at each move.

2 Professional and Ethical considerations

In line with the BCS Code of Conduct, we must only undertake research within our competency. The Computer Vision and Acquired Intelligence; and Adaptive Behaviour modules have given me the relevant knowledge to attempt the task of terrain navigation. In addition to this, the Junior Research Associate scheme has given me a valuable background in robotics and evolving hardware.

While no ethical approval is needed for this project, there are indisputable risks of dual use in AI research. However, at the fundamental level that I am working on, I have made a conscious decision to prioritise Open Science and all of my code will be publicly available on Github.

Because we will train our robot on images of real world environments, we will have to consider the dangers of people being in view and thus consider the GDPR rules. However, as the resolution of images will be low, this will not be recognizable.

Another ethical consideration is the safety involved in testing as the robot could be a trip hazard. Therefore, we will test the robot in an environment that it cannot hit people. This will be the safety netted area of the Future Technologies Lab.

3 Background Literature

3.1 Robot Locomotion

A standard wheel has low energy consumption due to a single axis continuous rotation; however, it is limited when climbing over rough terrain. To improve their ability to traverse such terrain, agents can use legs for example in a hexapod configuration which is incredibly stable [3]. However, robotic legs use multiple servos which have a high current draw and, due to the need for inverse kinematics [4], require complex control algorithms.

The Wheg design is a wheel-leg hybrid that can traverse more challenging terrain than standard wheels while maintaining low energy consumption [5]. Figure **??** shows a robot featuring Whegs with claw-like spokes which dig into rocks and obstacles.

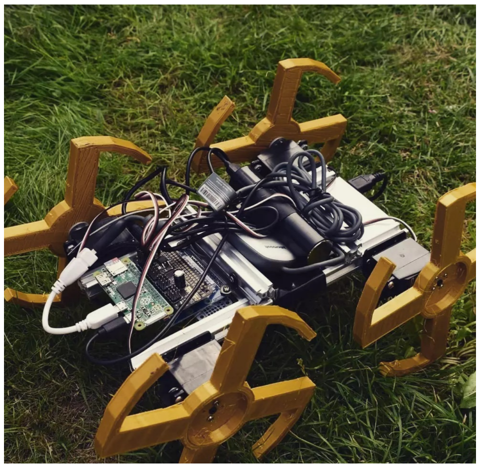


Figure 1: Prototype of the Wheg robot to test out the wheg design, in addition the hardware required. This prototype ran on a Raspberry Pi zero and used a phone charger cell providing 5.1V and a maximum of 2A of current.

Whegs are easily manufactured via 3D printing, and there are multiple design variations of the Wheg which suit different purposes [6].

The European Space Agency (ESA) adopted a similar design for the PROLERO project [7]. Although not quite a Wheg, it operates on a similar principle. This project used a simpler design of actuation where the legs were rods rotating on one axis. This design could at maximum travel over obstacles less than or equal to a height of 10 cm. The total payload mass was 1.5 kg making it light for space travel and the rover passed the tests at the ESA planetary utilisation facility, confirming the model's validity [7].

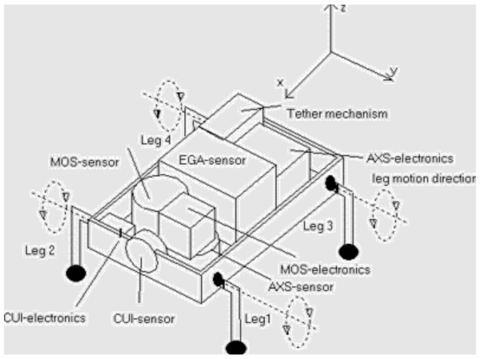


Figure 2: Figure showing the PROLERO robot taken from ESA [7]

Other potential designs are caterpillar tracks which have good energy efficiency due to the optimized traction system which allows more grip over uneven terrain. They do, however, weigh more and are more difficult to repair meaning that the robot would be inoperable if a link breakage, whereas, with wheels or Whegs, it could continue to move with worn-down tyres or grippers.

3.2 Biology and Bio-inspired Robotics

Research on cockroaches explored the use of cockroach-inspired limbs and movement. This involved a bend in the back in increasing climb over obstacles. Another biological advantage was how the cockroach would run into a wall. The cockroach limbs would be brought into phase at the point of climb. The rotational locomotion of the legs is comparable to the cyclic locomotion of wheel. This robot used a Wheg design to improve its advancement in a similar locomotion. A bend in the robot created a higher stance to climb. [8]

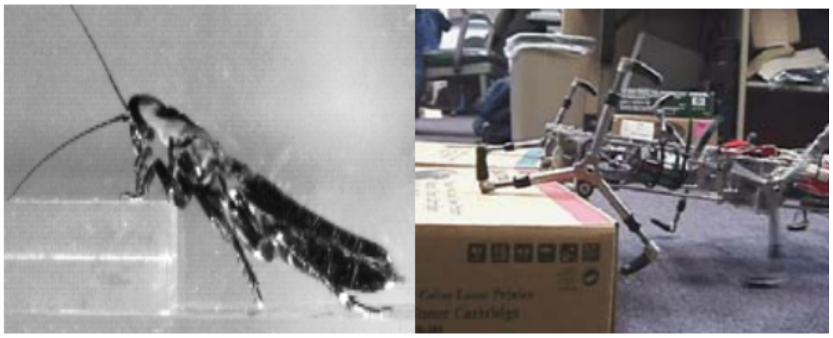


Figure 3: Figure taken from showing the cockroach rotation to an obstacle. This is compared to a bio-inspired robot. [8]

3.3 Reinforcement Learning

Reinforcement learning is a machine learning approach that does not need labelled training data. The focus is on exploring solutions and exploiting current knowledge through actions to maximize cumulative reward [9].

This approach has been deployed for route planning robots to plot a solution on different terrain. The reinforcement learning approach was successful in a small number of trials. This method used a neural network that took state information to predict the next step. The research was carried out in simulation rather than a physical agent.[2]

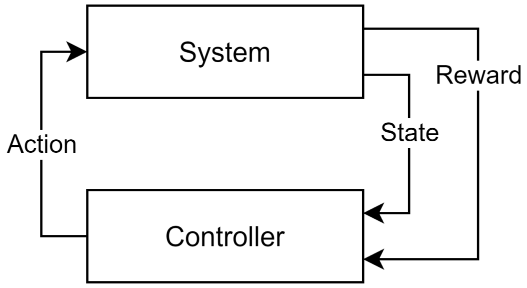


Figure 4: Reinforcement learning simplified showing.

Reinforcement learning is often described in terms of an agent, environment, state, reward, and action. The agent is the entity to which the reinforcement learning applies, such as a robot. The environment is the area in which the agent will roam and the state presents the agent's current position within the environment. The desired agent behaviour is then rewarded at the end of each trial. Figure 4 shows the process of state transition through action and reward.

This approach can include a neural network that trains under a series of trials.

3.3.1 Markov decision process

The Markov decision process formulates reinforcement learning mathematically. The model states that the future is independent of the past given the present. This is denoted mathematically by using P[t] to represent the current state of the agent, therefore P[t+1] represents the next state [10].

$$P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_1, ..., S_t] \tag{1}$$

State transition probability is given by the following equation:

$$P_{ss'} = P[S_{t+1} = s' \mid S_t = s] \tag{2}$$

The agent transitions from one state to another. This is important to reinforcement learning as it allows values to be functions of the current state. The state representation must be informative for an effective reinforcement learning approach to take place. Manually determining optimal state space is challenging, therefore the use of neural networks can be used.

Our agent will use this principle to apply the reinforcement learning. The state of the environment is taken and changed by the action produced. The agent will now have a new state.

3.4 Genetic algorithms

Genetic algorithms are optimization methods which are inspired off of biological systems such as evolution [11]. Over a number of generations encoded information (*genotype*) will be changed (*mutated*), where this change is measured via a fitness function. If the mutation has helped the overall goal then it will get a high fitness and replace other genotypes within the population.

An approach to evolutionary strategies is to generate the weights and biases via a Gaussian distribution within a one-dimensional array. Each generation will allow mutation of these values. Mutation functions fit Gaussian noise on top of the current encoded data, which alters the operation of the neural network when applied.

One optimization approach uses a Microbial algorithm that generates a population of Gaussian distributions and trials each of them in a tournament against one another. The winner overwrites the loser [12]. Particle Swarm optimization uses a method inspired by particle physics. When particles are energized through heat and cooled slowly, they move back into place better than if called quickly. This theory applied as a genetic algorithm allows sub-optimal solutions to move away and trial new solutions [13].

The optimization of the agent can be achieved by implementing evolutionary strategies to guide training towards a solution.

3.5 Evaluating performance of robots on terrain

It is important to be able quantify an agents' performance across trials. Current research in rough terrain robotics has used physical attributes such as the coefficient of friction and slippage plotted against the climb [14].

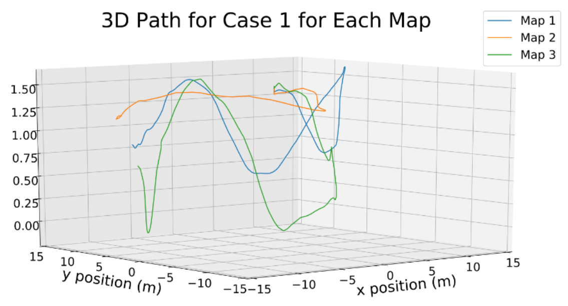


Figure 5: Figure Taken from a 3D plot of the simulated paths for a moon rover [2] The z-axis shows a prominent climb on map 1 and map 3.

Studies [15] have measured Locomotion performance based on several values, including slippage, required torque, and minimum friction coefficient. These metrics create an overall score, plotted throughout each trial. μ represents the friction coefficient needed for tangential (R) and normal (N) force with (r) given by the radius of the wheel with torque (T).

$$\mu_{needed} = \frac{R}{N} = \frac{T \cdot r}{N} \tag{3}$$

Slippage consumes energy without making the robot move forward. Slippage was calculated using the encoders on wheels. Distance is calculated using the following equation which uses encoder values.

The wheel encoder (Δ_{coder}) is used and multiplied by the circumference of the wheel ($2 \cdot \pi \cdot r$). This divides over the pulse/turn multiplied by the gear ratio.

$$\text{Distance} = \frac{\Delta_{coder} \cdot 2 \cdot \pi \cdot r}{N_{pulse} \cdot R_{gearbox}} \tag{4}$$

The slippage is the difference of ground truth and wheel odometry [15]. These factors form a Max and Min G score. Both robot robots could provide enough torque, but that does not mean equal performance. The G score is taken from all the G values, calculated from the torque needed. These attributes are easily calculated in simulation but would be harder to find on a physical robot. The use of gyroscopes and accelerometers are potentials for a substitute. There is also access to Vicon tracking which would allow 3D generation of the robot. This would allow us to quantify these attributes.

3.6 Sensors

Terrain mapping sensing can take many forms. One study used a lidar sensor that would read at four layers to construct a world model. This model creates a 3D perception allowing a hexapod robot to predict movement [16].

Optical flow uses the vectors between pixels in two images taken one immediately after another. This can be used for optical flow alone [17]. Optical flow can also be used to calculate obstacles and swerve an agent away from the obstruction via depth estimation maps [18]. This same depth estimation method was achieved using a convolutional neural network and the Canny edge detector and morphological operators [19].

Depth perception via stereo imaging is another method that requires less training. A hexapod robot used this method with six legs to predict stable movement over uneven terrain. This prediction method was accurate [20].

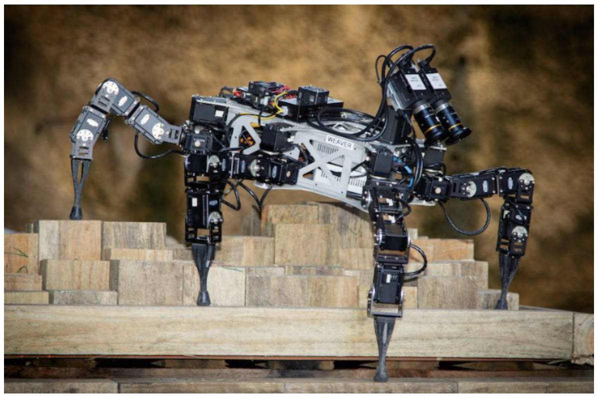


Figure 6: Figure showing Weaver the stereo vision hexapod. The ground is uneven for all legs, therefore the agent must adapt across all legs.

3.7 Contributions to the field

Self preserving robotics are important for remote locations such as planetary exploration, or hazardous locations on earth like nuclear reactors. The preserving of autonomous robotics will save money as well as prolong the lifespan of the agent [21]. Our approach will be using computer vision techniques to evaluate terrain and make safer and easier path decisions. Our design will also be implemented on a cheaper design chassis, aiming to bring down the overall cost of robot development.

4 Requirement analysis

Existing robotic solutions have not combined bio-inspired chassis design with reinforcement learning. This project will use a physical robot platform and a simulation of this platform for the gathering of data.

4.1 User Needs and ideal features

The agent will need to learn how to recognize the differences between rough and smooth terrain based on visual information and furthermore learn how to traverse it. Ideally, the physical robot will use its back actuator to improve its ability to climb over terrain.

The agent will need to understand its limitations and not attempt routes that it will get stuck on. Ideally this will be demonstrated on a physical robot but, if time pressure prevents this, it is still essential the algorithm works in simulation.

4.2 Limitations

This project does not intend to create a robot that can climb any terrain. While the Wheg design will improve the robot's climbing ability, it will not make it impervious to the laws of physics!

4.3 System overview

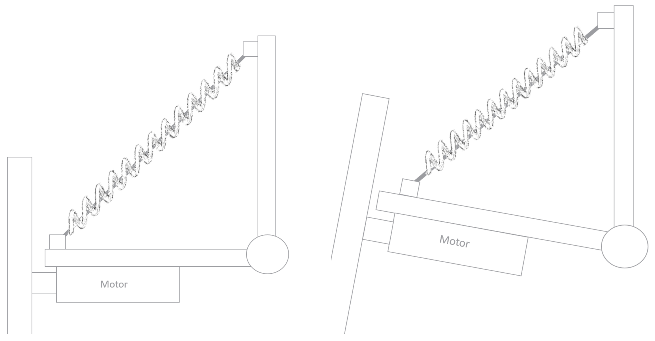
4.3.1 Simulation and application of the Robot

The agent will be evaluated in an environment, where we give it a target location and it must avoid rough terrain to get there. These constraints will be applied by calculating energy consumption. Within an environment there can be dangers, which we will simulate using sea and snow (suggesting the ground is too high) – entering these regions will result in 0 fitness. Using a neural network with a visual representation and vector as an input layer. The output will be a left, right, forward command. When we deploy this on the robot, we will most likely use a depth camera.

4.3.2 Robot platform

The robot chassis will incorporate a wheg design. This design will need to be able to climb over terrain while maintaining stability. The Whegs will be 3D printed, so must have a diameter smaller than 200 mm due to the printer capacity. Current designs typically use claw-like hooks on the whegs. This design is limited on specific terrains due to it not being able to reverse.

The main chassis of the robot will use the suspension system shown in figure 7 so that a higher incline on one extremity does not tilt the overall robot. This design was inspired by a Tamiya radio controlled car [22].



(a) Neutral position (b) Activated position

Figure 7: Figure showing the movement of the suspension system design moving over hardware. This uses a single spring shock absorber per Wheg.

In terms of back and neck actuation, we will need a pan and tilt style mechanism where the front of the robot chassis will pan left to right, to improve turning left and right within an environment. The tilt will be up and down on the axis of gravity acting as a back. A bend in the back will redistribute weight – the exact process applied in the cockroach research [8].

4.4 Programming languages

There are numerous programming languages which could be used for the simulation and robot components of this project. Ideally, the language I use will be is suitable for both.

Languages that are particularly well-suited to working with hardware include C++ and Python.

Arduinos use a version of C++. This can be quickly booted up and execute code sooner due to storing it in binary format. This method is more complicated for computer vision and machine learning tasks than simpler languages such as Python.

Python is a high-level language that abstracts away from syntactic detail compared to other languages. This design makes it slower at execution but has a wide ecosystem of machine learning tools. These tools are written in C++ as Python is slow, however, Python can combine them without a significant loss of time.

When Python deploys on hardware, there are different methods to undertake. One of which is circuit Python as a Micro-controller, where a small Micro-controller compiles to work as an Arduino but using Python. This is found on the Raspberry Pi Pico, Adafruit feather and Microbit. This approach overcomes speed and complexity issues; however, these boards typically have a low amount of memory. On the higher end is the Jetson, a Nvidia-developed board design for machine learning tasks. These are heavily used within brains on boards projects.

In this project I will use a Raspberry Pi. These are low-cost computers which have hardware access through GPIO pinouts and that run a Linux-based operating system. Linux is a good platform to do machine learning as there is plenty of library support. These are typically more expensive than Arduinos, but still much cheaper compared to higher specification models such as the Jetson. Though slower at booting than the other micro-controllers listed, it will provide the machine learning tools required for the project.

5 Project Plan

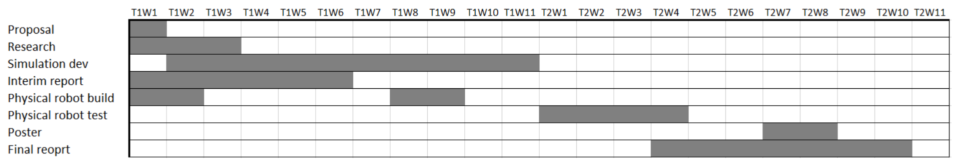


Figure 8: Timetable of development displayed as a Gantt chart

5.1 Phase one simulation

Phase one will focus on developing the navigation methods and a virtual environment to test within. This phase is critical for the project as simulation can test concepts more quickly and cheaply than physical robot testing. We will use the Python noise generation library to develop a 3D map of the terrain.

5.2 Phase two simulation trials

The deployment of the simulation will test the theory of using vision to exploit navigation over rough terrain. The agent will need to pick the most accessible routes which minimize the overall energy consumption. There are multiple ways of testing this concept.

5.3 Phase three building of physical robot

To understand whether or not this concept works in practice, we must try it on a physical robot. The real world has many more variables which affect the performance of algorithms. Lighting, colour, and shade can have significant effects on the way the robot perceives the environment. We will build the robot chassis from lightweight aluminum servo apparatus, and 3D printed Whegs. It is essential that this design can successfully climb over terrain and can travel in both directions. We will test this using a simple remote control. Once we know the physical limitations, it can help us design the testing environment.

5.4 Phase four applying simulation on the physical robot

The working reinforcement learning algorithm will deploy on the physical robot. This will test whether we can cross the "reality gap" and determine whether the algorithm functions with the real-life background noise.

5.5 Phase five evolving back movement

Evolving back movements is not an essential feature of this project but, it will significantly improve the agent's ability to climb. If there is time, this will be trialed using a reinforcement learning approach to help redistribute weight, thus improving the climb.

6 Methods and Preliminary Results

6.1 Simulation

We set up a Python simulation by generating a 2D map using Perlin noise, seen in figure 9. Perlin noise is a type of gradient noise commonly used to procedurally generate terrain [23] where the value of the space between two points changes smoothly. A grid of (n) dimensions is created. Each index is assigned a random vector, where the dot product is calculated, which allows interpolation to generate noise.

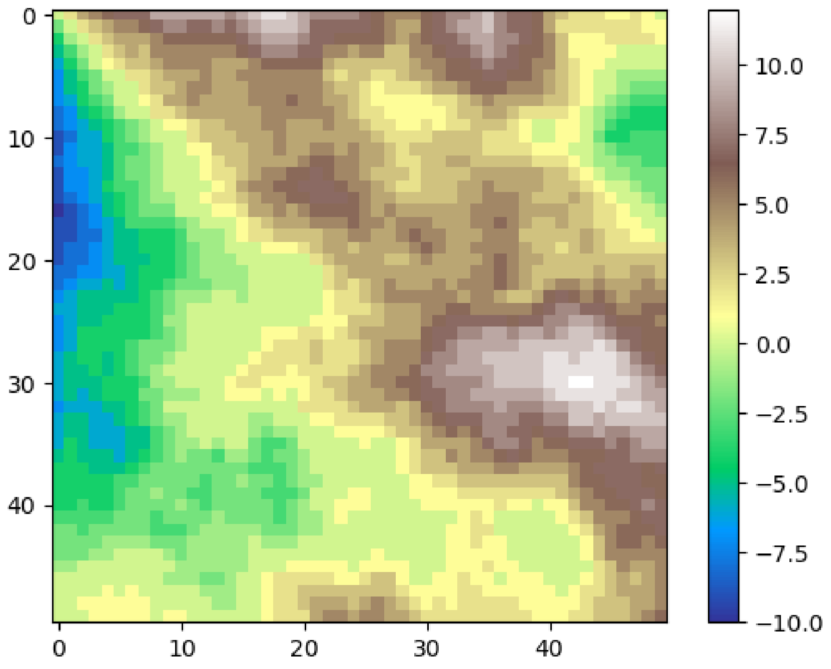


Figure 9: 2D simulation plot of a 3D terrain. Random generated points are shown to show the position of agents and rewards.

The smoothstep function represents a sigmoid-like interpolation. The persistence, octaves, and lacunarity are all altered via parameters in the world creation function. The octave is generate via a combination of frequency-amplitudes. Each octave adds a layer of detail to the world, where the contribution of each octave is defined by persistence. The lucunarity defines how much detail is added or removed at each layer. This function formed a map held as a two-dimensional array representing terrain heights numerically.

In such simulations, we can find how far a point is from the start position and the terrain steepness (gradient) at any point. The energy consumed (E) by a robot in a given trial is denoted by the number of steps taken (S) and the accumulated terrain value of each place (P). This terrain value is the step S up, down, or across from the current height. The Manhattan distanced is used as a measure of how far the agent has travelled. This which was picked in place of the euclidean distance due to increased accuracy with high dimensionality [24].

$$E_{nergy} = (\mid x1 - x2 \mid + \mid y1 - y2 \mid) \cdot \sum_{i=1}^{S}(P_i) \tag{5}$$

6.2 Robotic hardware

Throughout the initial testing, I developed three Wheg designs and evaluated two of these on a physical robot. In the first Wheg design shown in figure 10, each spoke had a single claw which would power more climb.

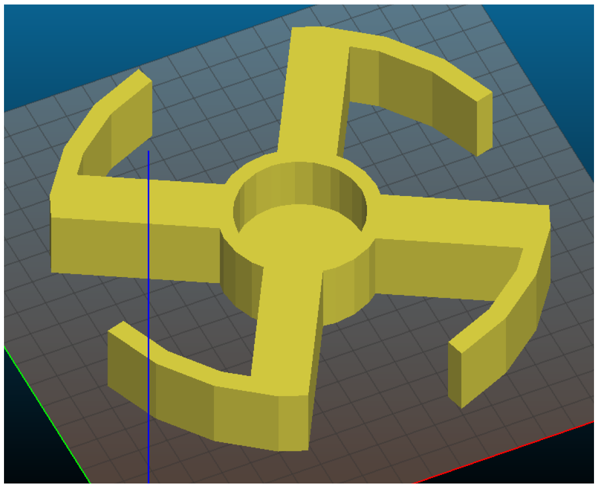


Figure 10: Initial claw design with 4 claws for stable rotation.

However, this design would struggle to reverse so I, instead, developed the reversible claw design shown in figure 11.

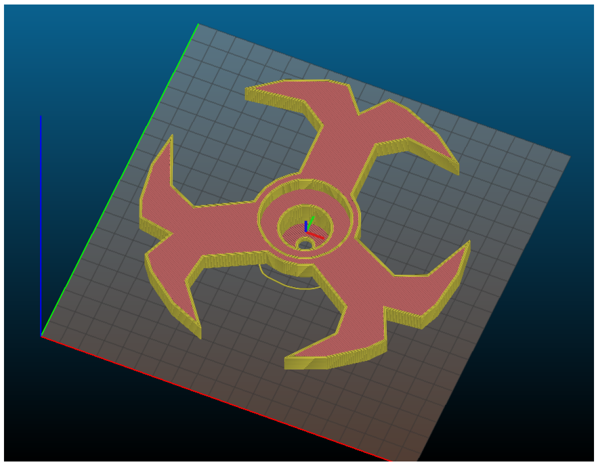


Figure 11: Reversible claw design using a tri-claw system for strength. The 4 claw stable rotation was replaced as this design had less gaps thus making it naturally more stable.

The hardware will use light Aluminium servo parts which are typically used in biped robotics and can be purchased cheaply on the internet. The design will allow rotation servos as the back and neck, alongside continuous rotation servos for the Whegs.

7 Supervisor meetings

7.1 13/10/21

Discussion on subsections missing from our initial draft, areas of research we should investigate and how to improve the simulation method. We need to research more about bio-inspired robots like what we are designing and show more figures in general to the papers we have referenced and reviewed.

7.2 27/10/21

We discussed how we will test the agent. Two main methods came up which was finding the highest point and making the most energy efficient path there or being set a vector and moving to it while avoiding obstacles. We looked at a paper on ants and trap avoiding which tied into our bio inspired research. Changes were suggested and made in the report. A discussion on sensors on the physical robot and what will work the best happened.

7.3 10/11/21

Discussion over Max G score and how it ties in with the calculation of terrain. Further discussing Perlin noise and how best display my simulation work within the report. We then spoke about the next stage which was to get the simulation agent working.

8 Appendices

Bio-Inspired Robotic Navigation On Varied Terrain Proposal

Dexter Shepherd [candNo: 215819]
Supervisor: Dr James Knight

November 10, 2021

1 Introduction

This dissertation studies autonomous robotic navigation through localized decisions based on previous experience. Nearly all organisms are limited at some point when it comes to movement. An octopus can fit through an aperture that is bigger than its beak [1]. A person can climb over rocks, but only if they have grip points for hands and feet. These environmental barriers define conditions to an organism's ability to traverse across it. When a path looks too arduous, we will often choose a more accessible route if possible.

These constraints comply with robotics also. Consider a Mars Rover over 395 million km away from the nearest person, if the Rover were to get obstructed there would be no one to recover it. In addition, it takes 181 seconds for a signal to get from Earth to Mars; thus, real-time control over changes in an environment will have taken effect by the time the signal arrives. A robot to self-preserve, like an organism, will need to know its limitations and not attempt tasks of movement outside these physical constraints. [2]

When deciding on routes, there may be the scenario that all paths are within the constraints. Some are simpler to navigate through than others. The agent will need to pick the option which requires the "least" effort. The best-case scenario is defined by the level of complexity of movement necessary to overcome the obstacle.

We will build an agent to perform this task by using computer vision techniques, which forms a prediction on the best route to take via a clock-face prediction method. This chassis will have a back actuator, stabilizer and neck actuator. Once attempting the terrain, the robot will need to use its chassis features to help it navigate over the landscape.

2 Aims and Objectives

Our aim is to make an explorer robot which navigates around terrain autonomously and avoids rough or potentially dangerous terrain.

- Research and evaluate current terrain navigating robotics
- Improve the chassis of current wheeled and Wheg robotics
- Critique current robot chassis and fix the limitations of them on our chassis
- Simulate an agent which avoids rough terrain when exploring
- Explore a range of computer vision techniques and research which one will be most effective for this task.
- Implement the simulation on an explorer Wheg robot
- Conduct real world testing

3 Relevance

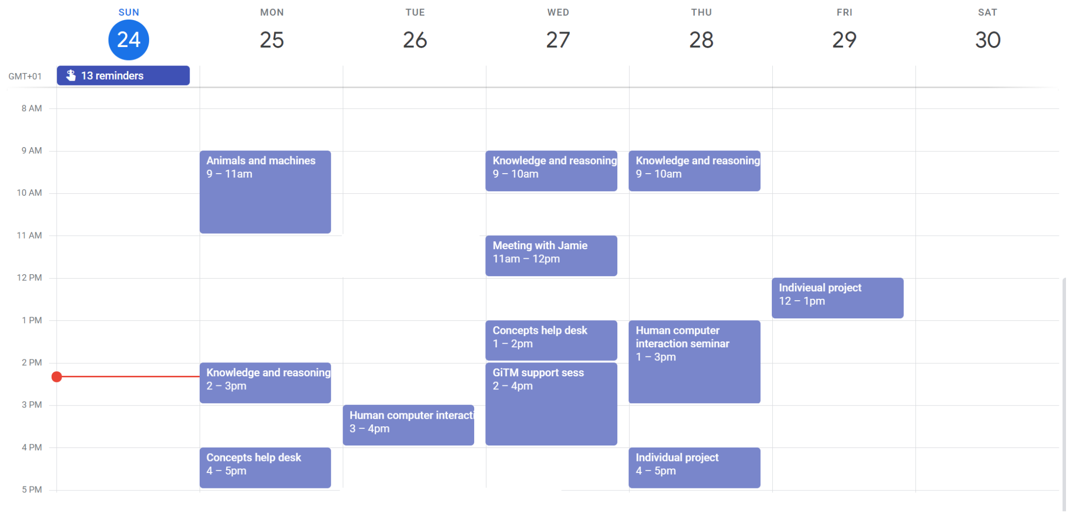
Robots being aware of their own limitations shares relevance with many aspects of Computer Science with AI. On the theoretical aspects it incorporates intelligence in animals and the cognitive science of perception of information. On the technical side we are exploiting computer vision techniques, neural networks and evolving systems.

The project links to my JRA research where I was evolving obstacle avoidance. This project is a step further where we are not only avoiding obstacles, but avoiding challenging terrain when there is an easier option.

4 Resources required

A chassis using continuous rotation servos will be required. This will run using the Raspberry Pi. Access to a 3D printer is imperative so the prining of Whegs can take place.

5 Weekly timetable



SUN 24 MON 25 TUE 26 WED 27 THU 28 FRI 29 SAT 30

GMT+01 13 reminders

8 AM 9 AM 10 AM 11 AM 12 PM 1 PM 2 PM 3 PM 4 PM 5 PM

Animals and machines
9 – 11am

Knowledge and reasoning
9 – 10am

Knowledge and reasoning
9 – 10am

Meeting with Jamie
11am – 12pm

Indivieual project
12 – 1pm

Concepts help desk
1 – 2pm

Human computer interaction seminar
1 – 3pm

Knowledge and reasoning
2 – 3pm

GiTM support sess
2 – 4pm

Human computer interacti
3 – 4pm

Concepts help desk
4 – 5pm

Individual project
4 – 5pm

Figure 1: Typical week of lessons

References

[1] James B. Wood and Roland C. Anderson. Interspecific evaluation of octopus escape behavior. *Journal of Applied Animal Welfare Science*, 7(2):95–106, 2004. PMID: 15234886.

[2] Tamir Blum and Kazuya Yoshida. Ppmc rl training algorithm: Rough terrain intelligent robots through reinforcement learning, 2020.

```python
#noise generator for simulation plot
import noise
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
import random as rnd
import copy
from matplotlib.collections import EllipseCollection

SIZE=50
def generateWorld():
    shape = (SIZE,SIZE)
    scale = 100.0
    octaves = rnd.randint(2,20)
    persistence = 0.5
    lacunarity = 2

    world = np.zeros(shape)
    for i in range(shape[0]):
        for j in range(shape[1]):
            world[i][j] = noise.pnoise2(i/scale,
                                        j/scale,
                                        octaves=octaves,
                                        persistence=persistence,
                                        lacunarity=lacunarity,
                                        repeatx=1024,
                                        repeaty=1024,
                                        base=42)
    world=world*100 #normalize numbers
    world=world.astype(int)
    print("Octaves:",octaves)
    return world,shape



def pickPosition(terrain,value,deny=[],LBounds=8,UBounds=4):
    current=-100
    cords=[0,0]
    deny.append(cords)
    while cords in deny:
        while current<value-LBounds or current>value+UBounds:
            cords=[rnd.randint(0,SIZE-1),rnd.randint(0,SIZE-1)]
            current=terrain[cords[0],cords[1]]
    return cords


#canReach will make sure the problem is solvable
def canReach(terrain,start,goal,endmarked=[[False for i in range(SIZE)] for j in range
    (SIZE)]):
    #check whether the bot can reach the other
    #expand out from end and make paths
    y,x=start[0],start[1]
    val=False
    if terrain[x][y]!=-1 and not endmarked[x][y]:
        endmarked[x][y]=True
        if x-1>=0:
            val=canReach(terrain,[x-1,y],goal,endmarked=endmarked)
        if x+1<SIZE:
            val=canReach(terrain,[x+1,y],goal,endmarked=endmarked)
        if y-1>0:
            val=canReach(terrain,[x,y-1],goal,endmarked=endmarked)
        if y+1<SIZE:
            val=canReach(terrain,[x,y+1],goal,endmarked=endmarked)
    if x==goal[1] and y==goal[0]:
        val=True
    return val



#getBestRoute will be used to measure how fit an evolved route is
def getBestRoute(terrain,start,end):
    #find the least cost route from A to B
    #return metrics

    return []
def expand(terrain,Map,start):
    for i in range(len(terrain)):
        for j in range(len(terrain[i])):
            if terrain[i][j]<-5:
                Map[i][j]=-1
            elif [j,i]==start:
                Map[i][j]=0
            else:
                Map[i][j]=terrain[i][j]+getDist([j,i],start)
    return Map

def getDist(start,end):
    d1=((start[0]-end[0])**2 + (start[1]-end[1])**2)**0.5
    return int(d1)

def readIm(r=5):
    #read the ground around the agent at a radius of i
    pass



while True:
    #generate the world terrain
    world,shape=generateWorld()
    #randomly pick a start position
    startPos=pickPosition(world,4,LBounds=6)
    vectors=[(1,1),(1,0),(0,1),(-1,-1),(-1,0),(0,-1),(-1,1),(1,-1)] #possible moves

    #print(canReach(Rmap,startPos,endPos))
    im = plt.imshow(world,cmap='terrain')
    cb = plt.colorbar(im)
    plt.setp(cb.ax.get_yticklabels([-1,0,1]), visible=False)

    plt.show()
    """
    lin_x = np.linspace(0,1,shape[0],endpoint=False)
    lin_y = np.linspace(0,1,shape[1],endpoint=False)
    x,y = np.meshgrid(lin_x,lin_y)
    fig = plt.figure()
    ax = fig.add_subplot(111, projection="3d")
    ax.plot_surface(x,y,world,cmap='terrain')


    plt.show()
    #"""

"""
    maxPath=30
    pathx=[]
    pathy=[]
    current=startPos.copy()
    energy=0
    last=startPos.copy()
    for i in range(maxPath):
        v=rnd.choice(vectors)
        pathx.append(current[0]+v[0])
        pathy.append(current[1]+v[1])
        last=current.copy()
        current[0]+=v[0]
        current[1]+=v[1]
        if current[0]>=0 and current[0]<len(world) and current[1]>=0 and current[1]<
    len(world[0]):
            if world[current[0]][current[1]]<=-6:
                print("water")
            else:
                climb=max(0,world[current[0]][current[1]]-world[last[0]][last[1]]) #
    non 0 value of total climb
```

```
                energy+=1+climb

    print("total energy consumed",energy)
    plt.plot(pathy,pathx)
    #"""
```

Listing 1: Simulation noise generation code

References

[1] James B. Wood and Roland C. Anderson. Interspecific evaluation of octopus escape behavior. *Journal of Applied Animal Welfare Science*, 7(2):95–106, 2004. PMID: 15234886.

[2] Tamir Blum and Kazuya Yoshida. Ppmc rl training algorithm: Rough terrain intelligent robots through reinforcement learning, 2020.

[3] T.-T. Lee, C.-M. Liao, and T.K. Chen. On the stability properties of hexapod tripod gait. *IEEE Journal on Robotics and Automation*, 4(4):427–434, 1988.

[4] Priandana, Karlisa, Buono, Agus, and Wulandari. Hexapod leg coordination using simple geometrical tripod-gait and inverse kinematics approach. In *2017 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, pages 35–40, 2017.

[5] Schwendner, Jakob, Grimminger, Felix, Bartsch, Sebastian, Kaupisch, Thilo, Yüksel, Mehmed, Bresser, Andreas, Akpo, Joel Bessekon, Seydel, Michael K.-G., Dieterle, Alexander, Schmidt, Steffen, Kirchner, and Frank. Cesar: A lunar crater exploration and sample return robot. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3355–3360, 2009.

[6] R.D. Quinn, Offi, J.T., Kingsley, D.A., and R.E. Ritzmann. Improved mobility through abstracted biological principles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 2652–2657 vol.3, 2002.

[7] European Space Agency. Prolero. https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Automation_and_Robotics/PROLERO.

[8] R.T. Schroer, M.J. Boggess, R.J. Bachmann, R.D. Quinn, and R.E. Ritzmann. Comparing cockroach and whegs robot body motions. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, volume 4, pages 3288–3293 Vol.4, 2004.

[9] B. Jaganatha Pandian and Mathew Mithra Noel. Control of a bioreactor using a new partially supervised reinforcement learning algorithm. *Journal of Process Control*, 69:16–29, 2018.

[10] Csaba Szepesvári. Algorithms for reinforcement learning synthesis lectures on artificial intelligence and machine learning.

[11] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[12] Inman Harvey. The microbial genetic algorithm. In George Kampis, István Karsai, and Eörs Szathmáry, editors, *Advances in Artificial Life. Darwin Meets von Neumann*, pages 126–133, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[13] Mohammad Reza Bonyadi and Zbigniew Michalewicz. Particle swarm optimization for single objective continuous space problems: A review. *Evolutionary Computation*, 25(1):1–54, 2017.

[14] J. Pijuan, M. Comellas, M. Nogués, J. Roca, and X. Potau. Active bogies and chassis levelling for a vehicle operating in rough terrain. *Journal of Terramechanics*, 49(3):161–171, 2012.

[15] Thomas Thueer, Ambroise Krebs, Roland Siegwart, and Pierre Lamon. Performance comparison of rough-terrain robots—simulation and hardware. *Journal of Field Robotics*, 24(3):251–271, 2007.

[16] Dominik Belter and Piotr Skrzypczyński. Rough terrain mapping and classification for foothold selection in a walking robot. In *Institute of Control and Information Engineering, Pozna ́n University of Technology*, 2011.

[17] Kahlouche Souhila and Achour Karim. Optical flow based robot obstacle avoidance.

[18] Yang Wang, Peng Wang, Zhenheng Yang, Chenxu Luo, Yi Yang1Wei Xu, and Baidu. Unified unsupervised optical-flow and stereo-depth estimation by watching videos.

[19] S.M.Abid Hasan and Kwanghee Ko. Depth edge detection by image-based smoothing and morphological operations. *Journal of Computational Design and Engineering*, 3, 02 2016.

[20] Marko Bjelonic1, Timon Homberger, Navinda Kottege, Paulo Borges, Margarita Chli, and Philipp Beckerle. Autonomous navigation of hexapod robots with vision-basedcontroller adaptation.

[21] SK Chiu, D Araiza Illan, and KI Eder. Bio-inspired self-preservation to protect robots from threats. in towards autonomous robotic systems: 18th annual conference, taros 2017, guildford, uk, july 19–21, 2017, proceedings (pp. 166-181).(lecture notes in computer science (including lecture notes in artificial intelligence).

[22] Tamiya. Rc car datasheets. https://www.tamiya.com/english/rc/chassis/cc-02/index.htm0.

[23] Ken Perlin. Improving noise. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, page 681–682, New York, NY, USA, 2002. Association for Computing Machinery.

[24] M. D. Malkauthekar. Analysis of euclidean distance and manhattan distance measure in face recognition. In *Third International Conference on Computational Intelligence and Information Technology (CIIT 2013)*, pages 503–507, 2013.