ES327 – Desire Path Formation

U1706124

Abstract

Agent based modelling can be used for path formation when pedestrians go off-road sacrificing their comfort for a reduction in walking distance. When this occurs on surfaces with vegetation over a period of time, it can lead to the creation of new paths. The models that describe this path formation are often limited by their inability to simulate a spectrum of properties for the walkers. In this project, the behaviour of the Active Walker Model was modified and extended to allow a spectrum of walking behaviours to be simulated.

An observation of pedestrian traffic was completed where the properties of 867 pedestrians were recorded to find the trends that lead to detouring or not. The observation showed that trends existed between walkers and their decision on whether to detour or not – most walkers that detoured were young, walked at a fast speed and had a good walking ability. The observation results were used to modify the existing active walker model to individualise the walker's parameters when simulating.

The extended model predicted paths with qualitatively similar layouts to those seen through the observation. This study was limited by only being able to gather data on a bounded set of weather conditions – wintry, cold conditions – so a full set of path usage patterns could not be obtained. Further work could look to investigate whether the combination of an individualised walker model and a model that simulates ground conditions more effectively and enhances simulation accuracy.

List of Tables

List of Figures

Table of Contents

1. Introduction

When pedestrians travel along unpaved surfaces, damage can be made to the ground, and this is how paths are formed. The paths – known as "desire paths" [1] – represent a detoured route for moving between locations that are more favourable than the existing pathways. For civil planners, they can be undesirable due to them either being visually unappealing or more heavily affected by external conditions such as weather conditions. Modelling and simulating the conditions required for these paths to form allows the development of a model that can mimic a human's decision making when choosing to detour off an existing path.

The project focuses on the existing agent-based model by Helbing et al. [2] on human trail formation which cannot exhibit different walker behaviours when moving across a surface. This report aims to see if differentiating the walkers' properties lends to better simulations on what would be experienced in a real observation of pedestrian traffic. There is also an observation of pedestrian traffic to develop the factors involved in individualising the walkers, and an attempt to see if implementing these changes into the model's walkers will make a difference to the results.

The solution could act as both a prevention tool for initial path design or as a remedial tool to fix existing issues with pathways. The impact this has is both a saving on potential pathing costs and a reduced amount of damage to the local environment.

1.1 Aims

This project aims to investigate whether it is possible to improve the simulation accuracy of the Active Walker Model by Helbing et al. [2] by introducing new parameters individualising the agents. This can be accomplished by using a pedestrian traffic observation to generate data to extend the walkers properties, then comparing the new simulations to the observed behaviour from the traffic observations. The result of this will allow a qualitative assessment of how accurately the simulation matches a real-world desire path with the included changes.

1.2 Objectives

To measure this project's success, it needs to be broken down into a list of concise, clear and measurable objectives. Setting these objectives allows for the progress of the project to be gauged.

The objectives of this project can be summarised as:

1. To fix existing bugs in the Active Model's code to improve simulation accuracy.
2. To implement new features into the model to allow for better planning of modelling of certain pathway systems.
3. To complete observations on pedestrian traffic, recording pedestrian path choices and the behaviour involved in making those choices.
4. To analyse the collected data, and create new parameters the model can use when determining paths.
5. To introduce the new conditions into the model to be able to generate new path outlines where the agents are individualised.
6. To tune parameters and simulate the updated model on the ground used from observations to compare with currently existing layouts. Validate if the model can predict the paths that have formed versus the ones that exist already.

1.3 Scope

The Active Walker Model [2] does not take into consideration any parameters that would make each agent's properties unique within the simulations. It operates under the assumptions that the agent's path choice depends on their inherent "want" to get to the destination and the grounds "attractiveness". This project aims to investigate what external effects might exist that the model doesn't currently account for, and how to incorporate these effects into the simulation. A social effect would be considered a factor that is unique to each agent, such as their walking speed (whether they are in a rush to reach their destination) or their walking ability. The lack of these effects can lead the model to produce semi-realistic results that do not correctly reflect the different factors that influence the path choice pedestrians make when travelling.

Part of the project involves an observation on pedestrian traffic, and as the project is being completed during the winter months of the year (October-February), not all possible parameters influencing the walkers can be explored such as environmental conditions such as varying weather (sunny weather versus rainy weather) and ground conditions (hard versus soft ground). As a result of this, data containing all possible conditions cannot be obtained. However, this project will provide insight into whether this is something worth further observations and research.

1.4 Risk Management

Care must be taken during the project, as part of it involves the inclusion of a practical observation. The associated risks with this project fall mainly within the simulation timeframe, which may become too small if large scale simulations are to take place after the code modifications have been completed. This can be accounted for by using risk management techniques defined in Munier [3]. In terms of risks related to data handling, full ethical approval has been obtained from the sub-committee of the Biomedical & Scientific Research Ethics Committee (BSREC), the Engineering Ethical review panel, at Warwick University. It details what information is being captured and how to manage it correctly. This confirmation can be found in Appendix A.

2. Existing literature research

Trail formation by pedestrians is a relatively new field of study [4]. It can be described as a complex relationship between human orientation, pedestrian motion and changes to the environment. When observing pedestrian movements, two main cases exist. The first case involves pedestrians trying to find the shortest distance to their destination, and the second is pedestrians attempting to avoid walking on uneven ground as it is uncomfortable. This supports the idea that pedestrians prefer to use existing trails, but they build a new shortcut if the relative detour would be too large [2]. If a new path is to be created, they can generate a new trail due to their footprints clearing some of the existing vegetation.

An example of the resulting trail system can be found in areas like public parks, which can be seen in Figure 1 below:

Figure 1: Within the straight paved walkways on the University campus in Stuttgart-Vaihingen, a trail system has evolved. There are two types of nodes involved: Intersections of multiple trails running in a straight line, and junctions where multiple trails merge into a single trail.

This project is based on two previous pieces of work, the first being a paper from the paper from Helbing et al. [2], and the second being its continuation by a previous 3rd Year Project Students [28,29]. Helbing's paper outlines the fundamental equations that can be used to model human behaviour when determining where desired paths might form. The model incorporates three main factors when determining the path any walker might take, these being the ground potential, destination potential and the trail potential.

2.1 Ground Potential

The ground potential is the comfort of the ground that the person is walking on, with low potentials signifying ground that is difficult to traverse (possibly rocky, steep inclines, uneven pathing), and high potentials being ground that is easy to traverse.

The model describes the ground potential $G_k(\boldsymbol{r}, t)$, by treating it as a spatiotemporal distribution of the existing markings on the ground, that are affected by each agent that walks over it. Trails are characterized by particularly large values of $G_k(\boldsymbol{r}, t)$, where the subscript *k* allows distinguishing of different kinds of markings. Due to effects such as weathering the markings have a certain lifetime $T_k(r)$ which characterizes their local durability. [2]

The creation of new markings by agent α is described by the term $Q_\alpha(r_\alpha, t)\delta(r - r_\alpha)$, where the Dirac delta function $\delta(r - r_\alpha)$ is used to set the value of the term to its position $r_\alpha(t)$ of the specific walker [2]. The quantity $Q_\alpha(r_\alpha, t)$ represents the strength of the new markings and can be specified for walkers using the following equation (1):

$$Q_\alpha(r_\alpha, t) = I(r)\left[1 - \frac{G(\boldsymbol{r}, t)}{G_{max}(\boldsymbol{r})}\right] \qquad (1)$$

Where $I(r)$ represents the spatially dependent intensity of clearing vegetation. The saturation term exists since the clarity of a trail is maximum when $G(\boldsymbol{r}) = G_{max}(\boldsymbol{r})$. Using this, the following equation (2) can be used to describe the spatiotemporal evolution of the ground:

$$\frac{dG_k(\boldsymbol{r}, t)}{dt} = \frac{1}{T_k(r)}[G_k^0(\boldsymbol{r}) - G_k(\boldsymbol{r}, t)] + \sum_\alpha Q_\alpha(r_\alpha, t)\delta\big(r - r_\alpha(t)\big) \qquad (2)$$

2.2 Motion of a Walker

From this, the model then begins to derive the motion of the walker on a two-dimensional surface, which in this case can be given using the following Langevin equation (3):

$$\frac{dr_\alpha(t)}{dt} = \boldsymbol{v}_\alpha(t), \qquad \frac{d\boldsymbol{v}_\alpha(t)}{dt} = -\gamma_\alpha v_\alpha(t) + \boldsymbol{f}_\alpha(t) + \sqrt{2\varepsilon_\alpha\gamma_\alpha}\boldsymbol{\xi}_\alpha(t) \quad (3)$$

From (3), $\boldsymbol{v}_\alpha$ denotes the *actual velocity* of walker ɑ. γ_α represents a friction *coefficient*. It is given by the *relaxation time* τ_α of velocity adaptation, specified as: $\gamma_\alpha = 1/\tau_\alpha$.The last term describes random variations of the motion per the fluctuation-dissipation theorem. ε_α is the *intensity* of the stochastic force $\boldsymbol{\xi}_\alpha(t)$, which was assumed in this case to be Gaussian white noise [2].

Assuming the time τ_α is relatively short when compared to the time scale of trail formation (which is characterized by the durability T_k), we can end up with the following equation of motion (4):

$$\frac{d\boldsymbol{v}_\alpha(t)}{dt} = \frac{v_\alpha^0\boldsymbol{e}_\alpha(\boldsymbol{r}_\alpha\boldsymbol{v}_\alpha, t) - \boldsymbol{v}_\alpha(t)}{\tau_\alpha} + \sqrt{2\varepsilon_\alpha\gamma_\alpha}\boldsymbol{\xi}_\alpha(t) \quad (4)$$

2.3 Destination Potential

The destination potential describes the direction in which the walker would like to go and gives a basis for the extent to which they would divert from this course to reach their destination.

When considering human walkers on flat homogenous ground with no trails, the desired direction $\boldsymbol{e}_\alpha$ of a pedestrian α at place $\boldsymbol{r}$ is given by the next destination $\boldsymbol{d}_\alpha$ such that:

$$\boldsymbol{e}_\alpha(\boldsymbol{r}_\alpha\boldsymbol{v}_\alpha, t) = \boldsymbol{e}_\alpha^*(\boldsymbol{d}_\alpha, \boldsymbol{r}) = \frac{\boldsymbol{d}_\alpha - \boldsymbol{r}}{\|\boldsymbol{d}_\alpha - \boldsymbol{r}\|} = \nabla U_\alpha(\boldsymbol{r}) \quad (5)$$

Where the destination potential is:

$$U_\alpha(\boldsymbol{r}) = -\|\boldsymbol{d}_\alpha - \boldsymbol{r}\| \quad (6)$$

When considering that any existing trails will cause an attractive effect $\boldsymbol{f}_{tr}(\boldsymbol{r},t)$ on the walker, which will again be defined by the gradient of the trail potential $V_{tr}(\boldsymbol{r},t)$, specified later on. The equation (7) for this effect can be defined as:

$$\boldsymbol{f}_{tr}(\boldsymbol{r},t) = \boldsymbol{\nabla} V_{tr}(\boldsymbol{r},t) \qquad (7)$$

As both the V_{tr} and U act on the pedestrian at the same time, it is possible to introduce an orientation relation by taking the sum of both potentials. This can be given as the equation (8):

$$\boldsymbol{e}_{\alpha}(\boldsymbol{r},\boldsymbol{v},t) = \frac{\boldsymbol{f}_{tr}(\boldsymbol{r},t) + \boldsymbol{e}_{\alpha}^{*}(\boldsymbol{d}_{\boldsymbol{\alpha}},\boldsymbol{r})}{N(\boldsymbol{r},t)} = \frac{1}{N(\boldsymbol{r},t)} \boldsymbol{\nabla}[U_{\alpha}(\boldsymbol{r}) + V_{tr}(\boldsymbol{r},t)] \qquad (8)$$

Where $N(\boldsymbol{r},t) = \|\boldsymbol{\nabla}[U_{\alpha}(\boldsymbol{r}) + V_{tr}(\boldsymbol{r},t)]\|$ acts as a normalisation factor. From these equations, we can reach that the vector $\boldsymbol{e}_{\alpha}(\boldsymbol{r}_{\alpha},t)$ points into a direction which is a compromise between the shortness of the direct way to the destination and the comfort of using an existing trail [2].

Each of the factors above focuses on two different timescales, short term and long term, each helping to determine the path which each walker takes. On a short-time scale, the model looks at describing how the agent moves across the surface, using its ground potential to get to its location. The long timescale would involve how the ground changes over time, depending on what paths the previous agents have taken. Simulation of possible path formation is very useful when it comes to the planning of way systems in cities [2] [5] as we can predict where these paths are most likely to form, and their use can be adopted from the beginning, rather than post-design.

2.4 Trail Potential

The final focus when modelling trail formation in humans is to consider that a trail potential exists. A trail must able to be recognised by walkers and must be near enough to them to be used. Whereas the ground potential $G(\boldsymbol{r},t)$ describes the existence of a trail segment at position $\boldsymbol{r}$, the trail potential $V_{tr}(\boldsymbol{r}_{\alpha},t)$ reflects the attractiveness of a trail from the actual position $\boldsymbol{r}_{\alpha}(t)$ of the walker. Since this will decrease with the distance $\|\boldsymbol{r} - \boldsymbol{r}_{\boldsymbol{\alpha}}\|$, the following relation can be applied [2]:

$$\boldsymbol{V}_{tr}(\boldsymbol{r}_\alpha, t) = \int d^2 r e^{-\|\boldsymbol{r}-\boldsymbol{r}_\alpha\|/\sigma(\boldsymbol{r}_\alpha)} G(\boldsymbol{r}, t) \qquad (9)$$

where $\sigma(\boldsymbol{r}_\alpha)$ defines the sight range (visibility) of the walker. The trail potential allows the model to describe the level to which a walker is interested in using a path, allowing for a realistic model of a human pedestrian to be used. Not all possible characteristics of a walker have been accounted for through the three potentials, but they result in a well-described trail formation model in good agreement with empirical observations [2].

2.5 Path Formation Modelling

When looking into how the paths form and the patterns that can be determined from them, a relation exists between path formation and path optimisation. It is difficult to separate both topics as design should be completed to maximise their use and minimise the chance a pedestrian creates their own desire path. Path optimisation itself spans across a lot of different areas ranging from Computer Science applications such self-movement capabilities by robots [6] , to business supply path optimisations where the aim is to make the most effective decisions in advertising spending [7].

When researching the relationship between path optimisation and formation, there are many references to what is known as "Minimum Steiner Trees" – MSTs. MSTs are themselves a combination of two other optimisation problems: Dijkstra's shortest path and the minimum spanning tree [8] [9] [10]. They create what are known as Steiner points, artificial points that allow for the minimum amount of path to be used, but still allow for travel between any of the points. [9]

Figure 2: Left: Minimal Steiner Tree for A set of 6 points (black circles) with Steiner points (white circles) shown. [24] Right: Image of custom paths made by Virgina Tech University students from Google Maps. [25]

What the images above (Figure 2) show is that if the main goal is the shortest amount of path between all destinations, the left image would be suitable. But in the case of wanting to travel between the locations noted with gold stars, a pedestrian is likely to walk directly between the points rather than following the path shown by the MST. We can conclude that designing for a minimum amount of path required is not always suitable for pedestrian traffic. This can be further explored through the right-side image (Figure 2) where pedestrians have made many different paths across a single open field. An MST could be applied here to reduce the number of paths across the field if a central region was created in the centre of the field.

This concept when applied to path formation, can give a baseline to work with when it comes to comparing what the agent-based model can produce, versus what the most efficient path layout is. As previously mentioned, the defined agent-based model [2] can give a similar approximation to an MST when the right values of V_{tr} are selected without having to deal with the complexities involved with explicitly solving it.

Some examples of how different types of formations exist for even a simple ground of only three points can be seen below, and how varying levels of attractiveness of the trails and their decay rate could affect the paths that are formed varying from a minimal Steiner tree (A) to a direct way system (B):

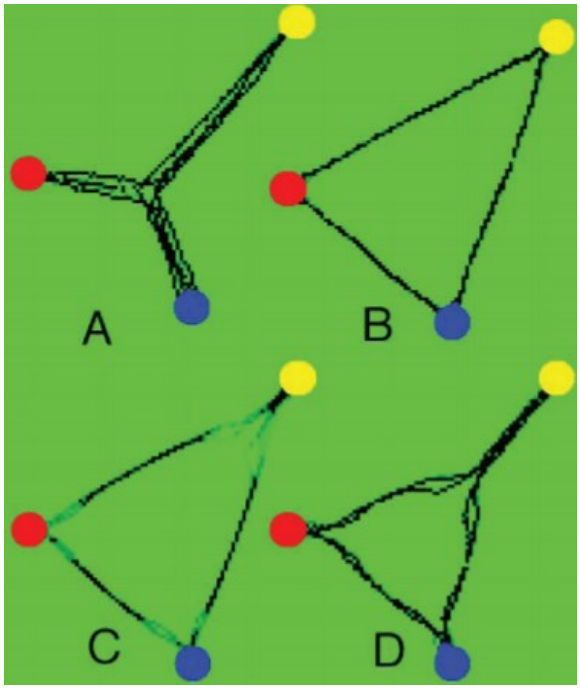

Figure 3: The influence of parameters on path formation within [2]. The destinations are shown by coloured circles. The darkness of a patch is positively related to its level of comfort for walking and indicates the eventual paths after 1000 iterations. (A) Decay rate = 0.1, impact on comfort level = 1,000, visibility = 100; (B) decay rate = 0.1, impact = 1,000, visibility = 1; (C) decay rate = 0.001, impact = 10, visibility = 10; (D) decay rate = 0.1, impact = 1,000, visibility = 10. [11]

From Figure 3 above, the most efficient path system is found in A, in which walkers are strongly influenced by the comfort level of the patches (visibility of paths is high), but the influence of steps on travel comfort quickly dissipates (paths decay quickly). [11]

2.6 Model Applications

The active walker model described above can be modified to simulate the formation of mountain trails [12]. It involves the creation of a new rule which prohibits direct descent or ascent on steep inclines, simulating aversion to falling.

In Figure 4 below are some images [12] that show how paths that form on land with varied elevation are often curved to reduce the rate at which the walker ascends/descends. It also shows that on some paths that have elevation changes, handled by using steps, that walkers can cause additional wearing (B2) causing a slope to develop. It is especially clear in images C1 & C2 that in areas where steepness varies rapidly, the need for the walker to be on a safe path becomes very important as large detours are taken when compared to what a minimal detour system would look like.

Figure 4: (A1, A2) The left-hand path has been augmented by humans since its formation. The right-hand path is spontaneously formed in the grass. Both paths have a similar angle relative to the steepest direction and regularity of direction change. (B1, B2) Another zigzag path from further up the trail. The top paths are on inclines of around 1:8. (C1, C2) is on a much steeper incline of around 1:2. This path breaks off from the main path that can be seen curving around towards the back of the picture. White lines are included on the right-hand versions of the Figures to highlight the paths.

2.7 Why do detours occur? How can we control them?

A Q&A session [13] was held with one of the authors of the original active walker model paper, Dirk Helbing, where he went into a discussion how people might have caused these paths to form over time, without directly noticing it. It also includes a mention to the term "emerging common good" which exemplifies an attraction effect, where a loop of humans change the environment, which in turn changes human behaviour etc.

"Only when the total distance becomes 20 to 30 percent longer through the detour, do people begin to create their own paths. They are very consistent in this because they also do this on routes that are only ten meters long. The result is astonishing mini shortcuts, as you can see in parks, where people would rather take four steps through the meadow than take a slightly longer path." [13]

This statement above leads to the belief that even if paths exist that are well thought out and designed, there is a nature for people to create their own detours [14], even if they are across small distances of *"only ten metres long"*. Adding to this, pedestrians have shown the ability to self-organise and self-evaluate the choices they make on where they travel. The use of this information could allow for the prediction of traffic-flow allowing for strategical positioning of attractions [15].

When focusing on groups of individuals and how this might affect path formation, it becomes apparent that collective behaviour [11] is more controllable than isolated behaviour. What this means is that if the targeted area is likely to contain more groups of pedestrians than individual pedestrians, the design of the path formation software may need to be different to accommodate for this (different width paths, different likelihood of taking a detour).

Looking at a paper into a multi-agent cellular automata model of pedestrian movements [16], it's possible to identify how to create a model to exhibit crowd behaviour and how design can influence this. A "social force" can be observed, being the intentions of a pedestrian to not collide with other people and to move in a specific direction (e.g. toward an exit) at a given speed [17].

When specifically looking into how certain age demographics may move between locations, a study looked [18] into urban park pathway design and senior walking behaviour. Within the paper, it splits the pathway design characteristics into three separate areas, the attributes of the path (physical attributes such as path material, width and length of the path, and pieces on the path such as benches, flowers and lighting), the paths surroundings (categorised via a physical element such as nearby features such as a body of water, or a visual element such as a viewpoint) and finally by its connection with different activity zones [18]. It suggested that senior-aged walkers preferred to not use steps due to limited mobility – which could link back to why, in Figure 4, a slope began to form. It also suggested that if sufficiently motivated, i.e. if they travelled via pathways with water on side or visual connection with water, they would be more inclined to travel further [18]. This could be because water can elicit positive emotions due to its soothing and calming effects [19].

In conclusion, a body of work has been carried out on the social dynamics behind pedestrian travel and what possible factors are involved in their choice of direction. However, less work has been completed on the walkers themselves and how they behave irrespective of their surroundings.

3. Walker Model Deconstruction

When breaking down the code created in [2] and the work done by previous students [28,29] for simulating walkers, there are a few key areas which follow the same logic as the equations 1-9 described above. It allows the model to match what would be seen in empirical scenarios.

3.1 Calculation of Potentials

To calculate the Trail Potential, the code takes advantage of a 2D convolution of a weighted array of the map of the size of the map supplied to it, and an array of exponential distances describing a negative effect on the potential increasing exponentially the further you get from the original point. This is the most time-consuming part of the code of the program as the convolution step is complex.

```
def calc_tr_new():
    TrailPotential[:,:]=sg.convolve2d(z[:,:]*Weight[:,:],subexpdist[:,:],mode="same")
```

Figure 5: Trail potential Equation from Active Walker Model Code

When focussing on how the exponential distance map is created, it can be treated as a 2D array where an initial point can have an arbitrary value, then an exponentially increasing value as you get further from the array. The code for this and an example

```
for xi in range(1,Nx+1):
    for yi in range(1,Ny+1):
        expdist[xi-1,yi-1]=np.exp(-isigma*(np.sqrt((x[Nx-xi]-xmin)**2+(y[Ny-yi]-ymin)**2)))
        expdist[-xi,-yi]  = expdist[xi-1,yi-1]
        expdist[-xi,yi-1] = expdist[xi-1,yi-1]
        expdist[xi-1,-yi] = expdist[xi-1,yi-1]
```

Figure 6: Code describing exponential distance matrix formulation from Active Walker Model code

```
ExpDistMatrix =

    18    13    10     9    10    13    18
    13     8     5     4     5     8    13
    10     5     2     1     2     5    10
     9     4     1     0     1     4     9
    10     5     2     1     2     5    10
    13     8     5     4     5     8    13
```

Figure 7: Example Matrix using a 2-D coordinate system where origin is (0,0) For each value of the $Matrix$, $(x, y) = x^2 + y^2$, *i.e point (3,3) top right corner =* $3^2 + 3^2 = 18$. *As distance increases from centre 0, values increase exponentially as you get further from origin.*

array can be shown below, where the matrix is symmetrical across the x and y axes:

One important factor that is used in the code to decide on the level of smoothing is the value of "isigma". The value works by scaling the results of the matrix by a factor, making the values either larger or smaller, thus allowing for more aggressive smoothing, or less aggressive smoothing. Examples of different "isigma" values can be seen below on an example map:

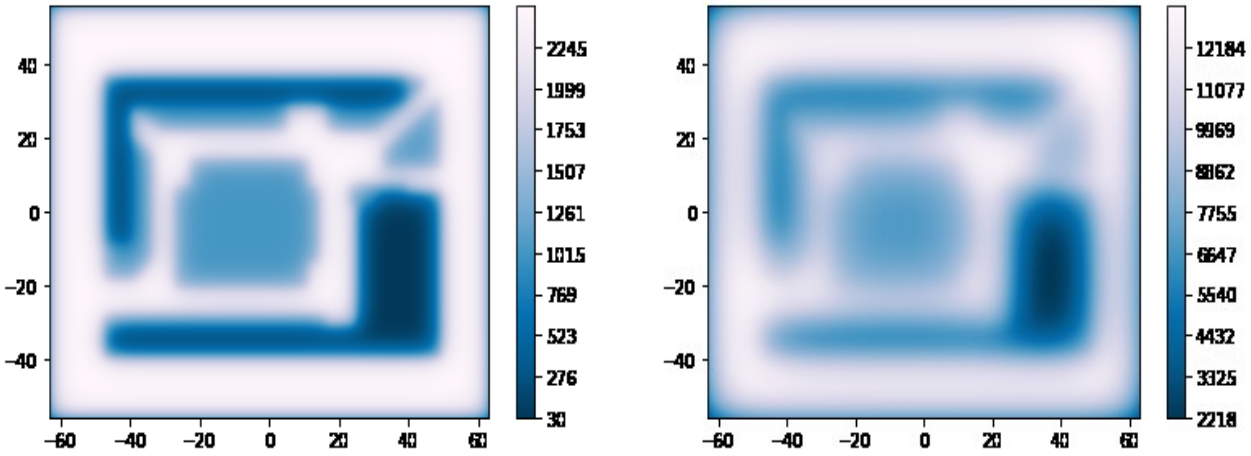

Figure 8: Maps of ground showing different levels of smoothing and how smoothing can heavily affect ground. Left Image: Ground smoothing relatively small where isigma = 0.2, Right Image = Ground smoothing relatively harsh where isigma = 0.5

Before the convolution, a calculation is performed to sub-sample the matrix so that any values below a certain threshold are removed as their effect will be negligible. This can be seen via the code below in Figure 9 and due to this implementation, the convolution operation can perform faster without affecting the overall result. This also affects the time taken for the calculation of the model to complete

```
# find index range > conv_thresh
subexpdist=expdist[(expdist>conv_thresh).any(1)]
subexpdist=subexpdist[:, np.any(subexpdist>conv_thresh, axis=0)]
```

Figure 9: Sampling of exponential distance matrix based on a set value of convolution threshold. [2]

The calculation for the ground potential within the code assumes the same exponential matrix in Figure 7 above and uses the origin point as the walker's destination set by the "set_up_walker" function mentioned below:

```
def set_up_walker(route_id):
    global vel,pos,track,intens,dest,start,route
    #START AND FINISH POSITIONS
    start=np.array(route[route_id,0,:])
    dest=np.array(route[route_id,1,:])
    #INITIAL VELOCITY AND POSTION
    vel=np.array([0.,0.])
    pos=np.array(start)
    #ARRAY CONTAINING WHERE THE WALKER GOES
    track=np.zeros((2000,2))
```

```
#Destination potential
    destp=-np.sqrt((dest[0]-x[:,None])**2+(dest[1]-y[None,:])**2)
```

Figure 10: Function describing the setup of a walker including their initial positions, initial motion components, and an array which contains their whole path as they move. Also, in the Figure is a line of code from the setting_potentials function

3.2 Changes Made Pre-Study

Within the code, there were a few bugs and changes that had to be addressed to make the model more useful and more accurate when simulating trail formation. The first bug that required addressing within the code was that the modelling of the average position of the walker during the simulation was not correct.

When this code is run, the result of the top equation in Figure 11 below will result in a value of zero. It is meant to store the average position of the walker being updated in a new cell each time. Instead, it overwrites its old position with its new position. This stops the model generating an array with the positions the walker has been in. To fix this, the statement can be modified to set the correct value to be used in the *'avpos'* array, so the positions of the walkers can be correctly stored. This can be shown in the code snippet below:

```
if (i%samp==0): avpos[:,(i%hist)//samp]=pos[:] #ORIGINAL

if (i%samp==0): avpos[:,(i%(hist*samp))//samp]=pos[:] #FIXED
```

Figure 11: Code representing changes to average position assignment, original on top and the modified code on the bottom.

Another error within the code is that when the velocity of the walker is calculated, there is an inertia related term in the formulation. This is not realistic in the real world as every time a pedestrian moves, they do not have to overcome their own inertia. The term *'dvel'* contributes to the inertia of the walker and for ease of simulation here it can just be assumed to be one.

The velocity equations associated with this can be seen in the code snippet below:

```
vel[0]+=-1/tau*vel[0] + (dvel/tau)*desdirx(pos[0],pos[1])/gradmagnitude+np.sqrt(2.*eps/tau)*xi[0]
vel[1]+=-1/tau*vel[1] + (dvel/tau)*desdiry(pos[0],pos[1])/gradmagnitude+np.sqrt(2.*eps/tau)*xi[1]
```

```
vel[0]+= -1/tau*vel[0] + (1/tau) * desdirx(pos[0],pos[1])/gradmagnitude+np.sqrt(2.*eps/tau)*xi[0]
vel[1]+= -1/tau*vel[1] + (1/tau) * desdiry(pos[0],pos[1])/gradmagnitude+np.sqrt(2.*eps/tau)*xi[1]
```

Figure 12: Code Representing changes to Velocity assignment to remove inertia term from calculation, original code on top and modified code on the bottom.

The plotting of the direction of the walker has been modified as the information presented by the graphs colour bar was confusing to interpret. To fix this, two new arrays have been introduced to the code to translate the angles on the colour bar to cardinal directions. This bar then allows you to see the desired direction the walker would want to move in at any given point on the map. The code for this can be seen below with a side by side of the given map and old and new colour bars.

```
#Plot the direction
Directions = ['W', 'SW' , 'S' ,'SE', 'E' , 'NE' ,'N', 'NW' ,'W'] # Array of Directions
DirectionAngles = [-180, -135, -90, -45, 0, 45, 90, 135, 180] # Array of Angles representing Directions

scgrad=np.arctan2(grad[1],grad[0]) * 180/np.pi #convert Rads to Degrees
levels = np.linspace(-180, 180, 360) # -180 to 180 = 360 increments of 1 degree

cs = plt.contourf(X, Y,scgrad, levels = levels, cmap='hsv')
#Define Both Colourbars
cbar = plt.colorbar(cs, ticks = DirectionAngles)# Set colourbar to be active with ticks at each angle increment of direction
cbar2 = plt.colorbar(cs)

cbar.ax.set_yticklabels(Directions) # Set Lables of Colour Bar to Directions of Travel
cbar.set_label("Direction to Travel") # Label ColourBar Axis

plt.scatter(track[0:1999,0],track[0:1999,1])
#Plot Start and end Co-ords of specific walker
plt.scatter(start[0], start [1])
plt.scatter(dest[0], dest[1])

plt.show()
```

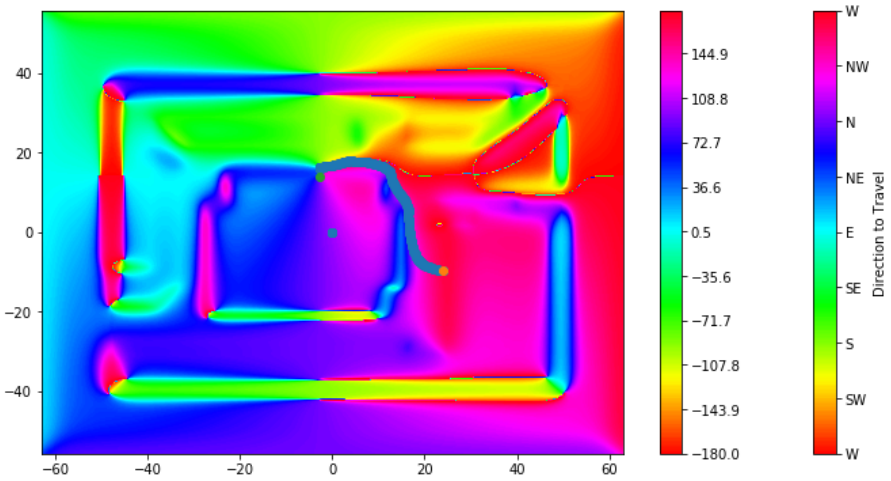

Figure 13: Top: adjusted code for direction plotting, defines two new arrays with colour bar information then creates and sets labels to display to the user. Bottom: Example graph showing a colour map with the old colour bar design (Left) and new colour bar design (right). New bar uses cardinal directions to indicate where the walker wishes to move. Note the curling line in the top right corner of the image is a plotting artefact.

3.3 Model Parameters

The model uses multiple different parameters which can be varied to affect the behaviour of the simulation. The first of these is a magic number [20] when calculating the gradient of the trail potential, which acts as a scaling factor. This can be replaced by a named variable detailing the effect it has on the model. The

```
def setup_potentials():
    global grad,desdirx,desdiry,dest
    grad=0.003*np.array(np.gradient(TrailPotential))
```

Figure 14: Code describing the equation that determines the influence of the trails on the overall gradient array. Included is the magic number (0.003) which operates as a scaling factor for the trail potential. This magic number is addressed fully in Figure 31

changes to this can be seen in Figure 14 below:

The other main parameters of the code are contained within the initial setup of the model. The role and values can be summarised in the following table:

Name	*Value*	*Description*
t_track	50	Describes the rate at which the ground potential changes. The larger this number the slower the previously made tracks disappear.
dt	0.1	Time step for the simulation.
tau	5.	Describes the rate at which the walker's velocity decreases over time, detailing how quickly the velocity decays (how the walker's motivation decreases over time)
isigma	3./5.	How far-sighted the walker is and how far can the walker see to evaluate a paths attractiveness. A linear scalar parameter where the bigger the value, the quicker the values decrease.
conv_thresh	10.e-4	Sets the minimum value for cut-off within the exponential distance matrix, removes values of little impact to the convolution operation.
precision	1.**2	A value that determines how close the walker must be to their destination for the model to evaluate if the walker has reached their final location.
eps	0.025	Amount of variance/noise in the velocity calculations accounting for randomness in the walker's path, i.e. not perfectly smooth motion.

Table 1: Table Defining the main adjustable parameters of the program that affect the simulation in terms of changing potentials or reducing computation cost in terms of simplification. Alongside the names, are the values that the parameters contain and the description of what they do,

All these parameters allow the simulation to be modified in its output and its complexity in terms of run-time. They can be used to help tune the model to change its behaviour depending on the situation required.

3.4 Diagnostic Tool for maps

An addition to the model has been made to allow for a diagnostic to be performed on the maps combined potentials to observe if there are any points where a simulation of a walker may get stuck.

The first step for this was to use import two new python library packages into the model code, SciPy Filters and SciPy Morphology [21]. After importing these libraries, a function can be created to generate the local troughs within a map – locations wherein every direction the potential is less than where they currently are. These points could signify locations where the walker cannot escape because their "want" to get to the destination is not strong enough. The base code for this function was obtained [22] and with minor changes to swap to maxima instead of minima, the function can return an array of local maximums on the potential map. The code for this can be seen below:

```
def detect_local_maxima(arr):
    # https://stackoverflow.com/questions/3684484/peak-detection-in-a-2d-array/3689710#3689710
    # http://www.scipy.org/doc/api_docs/SciPy.ndimage.morphology.html#generate_binary_structure
    neighborhood = morphology.generate_binary_structure(len(arr.shape),2) # define an connected neighborhood

    # apply the local minimum filter; all locations of minimum value
    # in their neighborhood are set to 1
    # http://www.scipy.org/doc/api_docs/SciPy.ndimage.filters.html#minimum_filter
    local_max = (filters.maximum_filter(arr, footprint=neighborhood)==arr)

    # local_max is a mask that contains the peaks we are looking for, but also the background.
    # In order to isolate the peaks we must remove the background from the mask.
    background = (arr==0) # create the mask of the background

    # a little technicality: we must erode the background in order to successfully subtract it from local_min,
    # otherwise a line will appear along the background border (artifact of the local minimum filter)
    # http://www.scipy.org/doc/api_docs/SciPy.ndimage.morphology.html#binary_erosion
    eroded_background = morphology.binary_erosion(background, structure=neighborhood, border_value=1)

    # we obtain the final mask, containing only peaks, by removing the background from the local_min mask
    detected_maxima = local_max ^ eroded_background
    return np.where(detected_maxima)
```

Figure 15: Code gathered from [22] [21] that work to identify the local maxima within the map by breaking it down into smaller neighbourhoods and filtering and removing the excess information. Then returns an array containing the points where maximums have been detected.

This function can then be used in conjunction with the plotting function like the graph plotting previously mentioned, allowing for a combined potential map with the maximums on top. A special change was made within this function in the addition of a ColourDepth variable. This variable works by generating a number between 1-1000 proportional to the difference between the maximum and minimum potential values. Doing this ensures that the colour bar will have a smooth transition regardless of the "TotalPot" values. The code to achieve was created and can be seen below:

```
def plot_potentials():
    global dest, PlotTickNo
    TotalPot = np.zeros((Nx,Ny)) # Define Blank Arrays of Map Size
    TotalPot -= np.sqrt((dest[0]-x[:,None])**2+(dest[1]-y[None,:])**2) # Combine Equ from paper
    TotalPot += TrailPotential # Combine Equ from paper

    Maximums=detect_local_maxima(TotalPot) # Run Detection Function to return array of Maximums

    PlotTicks = ['Most','Least']; # Text for Labels for Colourbar
    PlotTickNo = [TotalPot.min(),TotalPot.max()]; # Setup array of Min and Max values
    plt.figure(figsize=(12,6)) # Set Figure Size to 12in x 6in
    # Plot Potential Map
    ColourDepth = int((PlotTickNo[1]-PlotTickNo[0])) % 1000 # Create Scale For Pot Map with max value of 1000,
    #Scales Range of PlotTickNo by 10
    cs = plt.contourf(X, Y, TotalPot, levels=np.linspace(PlotTickNo[0],PlotTickNo[1],ColourDepth),cmap='PuBu_r') # Plot Pot Map
    cbar = plt.colorbar(cs, ticks = PlotTickNo) # Set Colour bar to only have 2 values
    cbar.ax.set_yticklabels(PlotTicks) # Set Lables of Colour Bar to Directions of Travel
    cbar.set_label("Attractiveness scale, Larger = More ") # Label ColourBar Axis
    #print(Maximums) #Can Print maximums if you want to see them in array form
    plt.scatter(x[Maximums[0]],y[Maximums[1]]) # Plot onto the graph the points in Maximums Array
    plt.show # Show Graph
```

Figure 16: Code Combining the new function described within Figure 15 and the plotting code within Figure 19 (top). Creates a new potential using the same methods as the main simulation, but instead of then plotting the paths

As a result of the code change above, understanding the plotting of the local maxima, thus the areas where walkers may get stuck while simulating is now easier to understand. An example of this can be seen below:

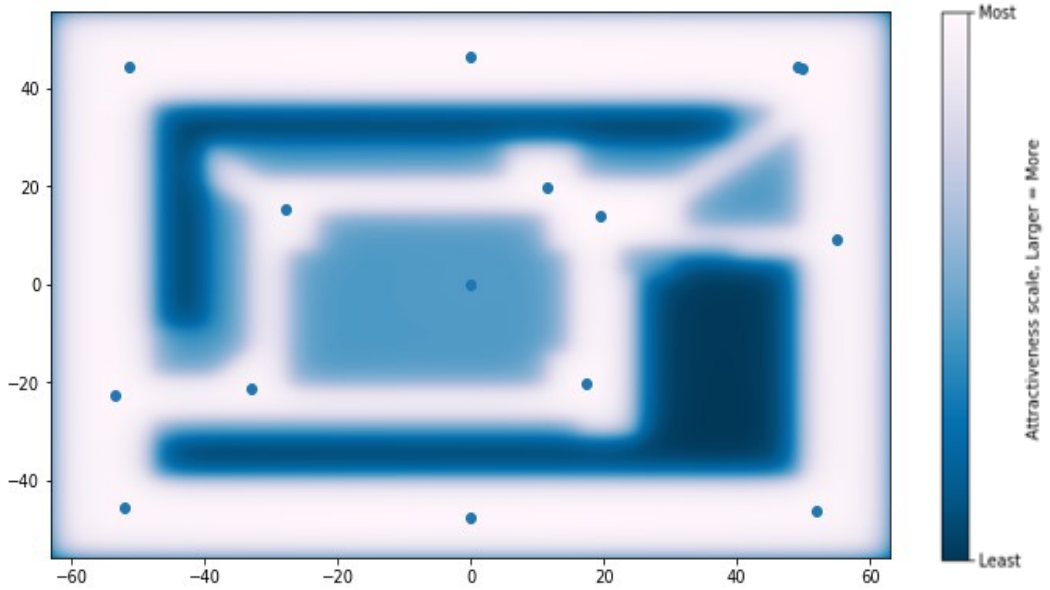

Figure 17: Example Map showing the points where a local maximum can be found. This could cause simulation issues if a walker goes near to these points, then does not have a large enough destination potential to escape from its local point of attraction.

3.5 Observational Study

A significant part of the project involves completing an observational study on pedestrian traffic. This data collection facilitates analysis into what possible parameters can be set up to begin to individualise the walkers of the active model. It also gives the chance to compare the model with empirical data, to see if real patterns can be recreated. Also, to see if the model can correctly simulate unique walkers' paths and if the addition of these new parameters changes the simulation.

The observation information can be given as follows:

Property	**Detail**	**Justification**
Location	See Figure 18 below	An area with a clear-cut path and a sizable potential detour as an alternative route, easy to distinguish within the model what direction the pedestrian takes.
Number of Sessions	2	Taking in data that is spread across multiple different time periods allows for a more varied list of utilisation patterns.
Timing of sessions	8:30-10:30am, 15:00-17:00pm Date: 29th January 2020	Two time periods were identified as potential high traffic times, early in the morning – as students walk to morning lectures – and mid-evening – as students are likely finishing their day and walk back to their accommodation.

Table 2: Details of the relevant information for the observation of pedestrian traffic.

Before any data was gathered, an ethical request was completed and approved allowing the capture of data of the pedestrians. This data was only relevant to the project, and other necessary data was taken down. The data recorded followed the ethical approval which can be found in Appendix A.

3.5.1 Walker speed:

The walker's speed is important to capture as it could assist in creating a relationship between the destination potential and the decision making when choosing a specific path. For instance, if the walker is in a rush to get to their destination, they may be more inclined to take a shortcut or to walk faster than the average person. The data was captured with a stopwatch to measure the time each person took to go from a given start destination to an end destination, an average was then taken of the

values to find approximate slow/medium/fast walking paces. On its own, this data does not allow trends to be interpreted from the walkers, but when combined with other variables, such as path choice or age groups, it allows for more some specific traits to be developed for certain groups of walkers.

3.5.2 Walking Ability:

When talking about the walking ability of the walker, it is an observational measure of how the walker is moving through the target area of the observation. It can cover aspects such as, is the user on their phone, e.g. talking to someone, so their focus is diverted from getting from point A to B, or are they having to stop in order to do something such as tie a shoe or talk to someone. It allows for a measure of the user's ability to traverse the terrain and to gauge their want to get from point to point. This is an interesting piece of data to record as it in developing ideas behind what conditions might cause someone to deviate off path or change direction. If someone is not focused on reaching their goal, they may be less likely to take a shortcut as there is no incentive for them to do so (save time) when they do not need to.

3.5.3 Start and end destinations:

It's important to note down the start and end destinations for each of the walkers during the observation as it could give information into patterns of data that could suggest certain routes cause certain types of behaviour e.g. if travelling in one specific direction you are much less likely to take a detour versus the route in the opposite direction. Recording this could also allow for a parameter to be introduced into the code that allows for certain types of walkers to be more likely to revel in certain paths, making the model act more like empirical data.

3.5.4 Age group:

To investigate the effects that age might have on path decision choice, noting down the approximate age of the walkers could allow investigation of its effects. It is known from [18] that senior walkers can have their motivations for path choice affected by the environment around them, so this effect could try to be recognised within the study completed here. If a notable pattern is detected through the observation, the model could be adapted to give each walker a random age group,

which could change its behaviour when simulated. The age groups have been kept relatively wide for the study as gauging a walker's age from a distance with a level of confidence is difficult if the age groups are too small.

3.5.5 Detouring

Capturing the information regarding who took the detour while walking in the study area allows the investigation into whether certain walker parameters give a fewer probability of detouring. If we record down the details of each pedestrian that detours, it may be possible to develop a relationship between different parameters and taking a detour, allowing it to be adapted into the model.

The final parameters to be used during the observation for the study can be given below, with their named descriptors and the values used to identify them:

Parameter Name	**Descriptor**	**Value Identifier**
Age Group	<25, 25-50, >50	1, 2, 3
Start & End Locations	Different Journey ID's	1, 2, 3, 4, 5, 6
Walking Speed	Slow, Medium, Fast	1, 2, 3
Walking Ability	Bad, Average, Great	1, 2, 3
Did they Detour?	Y/N	1,0

Table 3: Final Study Parameters with their descriptors and identifiers that will be used when taking down the observations.

3.6 Study Area

The area being utilised in the study is next to The Slate [23] on the Warwick University Campus and can be seen below via the google maps and photo images in Figure 18 [26,27]. The study will operate by starting a timer as a walker crosses an invisible boundary line determined by the person taking the observations, then timed as they walk through the observation area to the boundary lines on the opposite side. As well as noting down the speed the walker was moving, a judgement will be made by the observer on the walker's moving ability and approximate age group. This will be completed for single walkers and groups, with the latter being noted down as groups of walkers may present different patterns to those created by individual walkers.

Figure 18: Top : Images [26] [27] taken from Google Maps of Map and Satellite View of study area. Below: Image of Study area taken from bottom left star relative to top left image. Centred in bottom image: Potential desire path location.

To apply this to the active walker model, a re-creation of the study area and entry locations was created and can be seen in the diagram below. Alongside the image, is a table identifying all the possible routes that a walker could take, with their identification number that will be used in the study. This re-creation allows the key components of the map to be used in the model to simulate the ground that was observed.

Figure 19: Created map of the Study Area with location identifiers a, b and c.

ID	**1**	**2**	**3**	**4**	**5**	**6**
Start Location	a	a	b	b	c	c
End Location	b	c	a	c	a	b

Table 4: Table describing journey IDs and location information from fig 19.

4. Results and Applications

4.1 Study Results

There are three main areas of interest when analysing the results of the study, and they can be given as follows:

- The relationship between Journey and Frequency, to set realistic modelling of choice of route for the model.
- The relationship between Age Group and Speed, to look at whether different age groups are more likely to travel at certain speeds on average.
- The relationship between Age Group, Speed and Detouring, to see if a relationship exists between different parameters.

Once the observation had been completed, it was digitised and collated in order to begin to analyse the data. In total across both recording sessions, 867 journeys were observed, with a relatively even spread of journeys where (518:60%) were recorded in the morning and (349:40%) were recorded in the evening. This seems to be reasonable as the morning session would be a peak time with high traffic flow as it falls within the start time for most University lectures at 9 am. The reason for fewer journeys in the evening in comparison to this could be due to students on average finishing at different times of the day, so their distribution of journeys would be more spread out across the afternoon/evening periods.

The results for the journey information of each session are shown below, where the total number of journeys were recorded as well as the path each walker took:

Sessions				
Time of Session	**8:30-10:30**			
Journey ID	1	2	3	4
Frequency	418	59	36	5
				Total journeys: 518
Time of Session	**15:00 - 17:00**			
Journey ID	1	3	4	6
Frequency	80	252	10	7
				Total journeys: 349

Table 5: Table detailing the observation results for journey choice and frequency across the two sessions. The table contains the total number of journeys for each session as well as the time that each session took place

From table 5 above, some interesting trends were discovered. Within the results, it was found that specific journeys were a lot more common than others. This can be seen within the morning session as journey one had an 81% share of the total journeys made. It can also be seen in the evening session where journey three has a 72% share of all journeys. These two journeys combined are representative of 77% of all the observations journeys, so it is accurate to assume that they represent the bulk of all the walkers in the target area. Instead of excluding the 23% of walkers that picked other paths, it may be possible to put a bias on the model so that it is more likely to pick certain journeys over others. This would then match the behaviour found empirically. The analysis of the results of the frequency of each journey can be seen below:

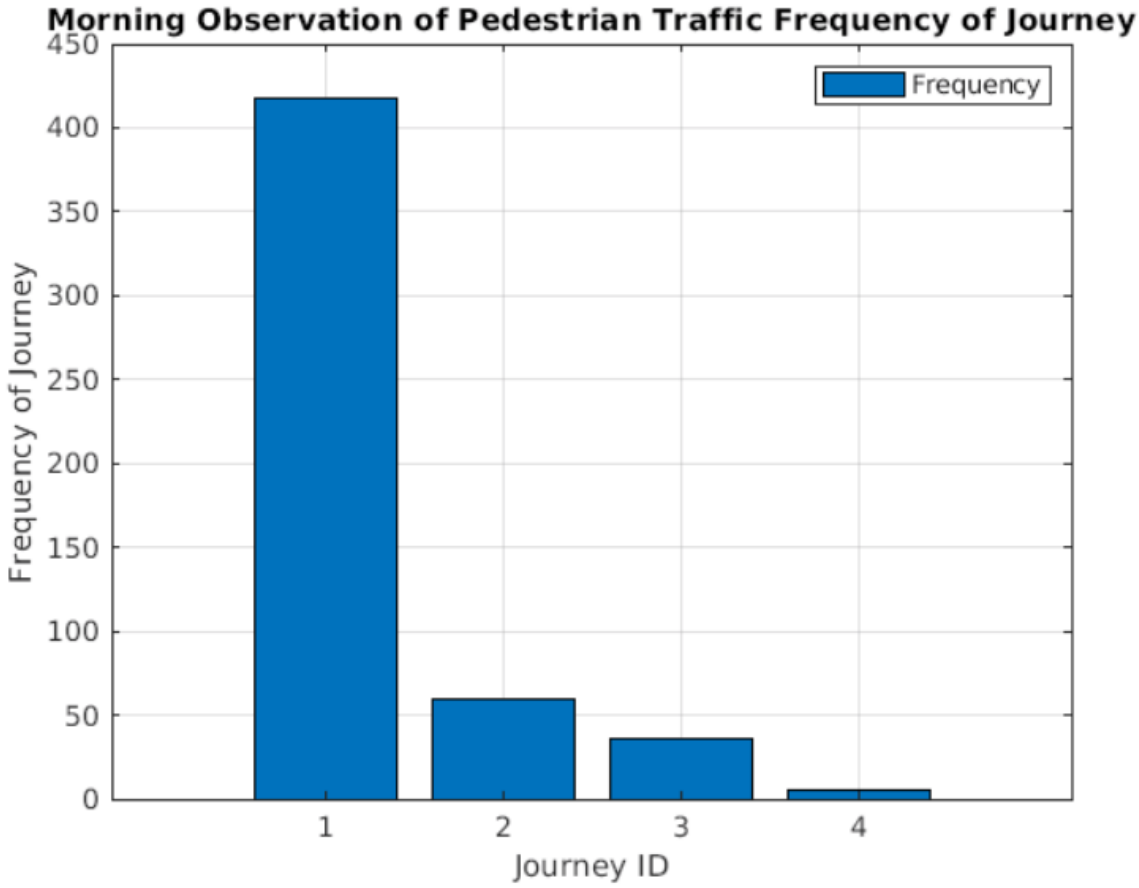

Figure 20: Graph Plot of Frequency and journey ID's for Morning Session showing how many pedestrians were recorded

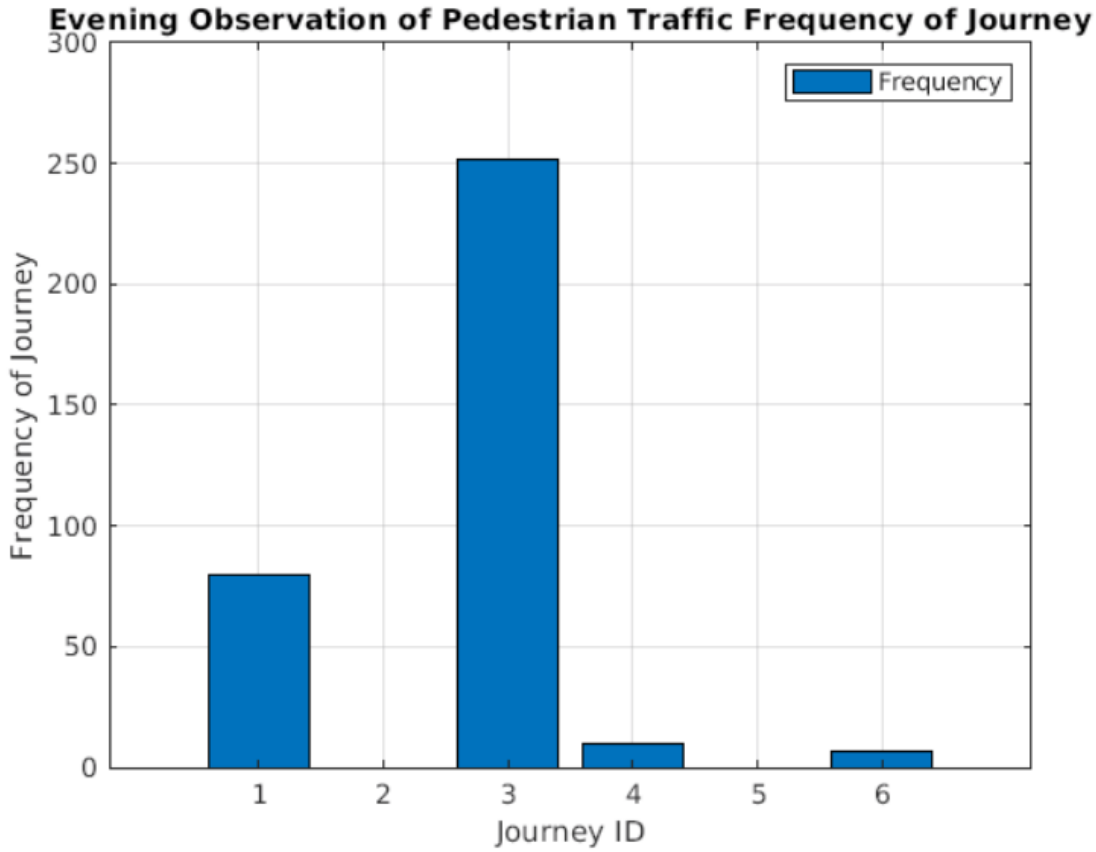

Figure 21: Graph Plot of Frequency and journey ID's for Evening Observation Session showing how many pedestrians were recorded

Journey one was observed as the most frequent journey in the morning session, and this would make sense as it represents the path that students take on their way to university lectures from their accommodation. This is also reflected in the evening session where the dominant journey is number three, which represents students walking back to their accommodation after they finish university for the day.

4.2 Age Groups & Speed

The relationship between age groups and the speed of the walkers could allow the model to differentiate between old and young walkers average walking speeds. Seeing if there is a trend within these two groups will allow the model to give an appropriate speed for a specific walker based on their age. The modelling of the study results for age groups and speed can be done by combining both observation sessions and checking for how often a certain speed journey occurs within an age group.

The graph for this can be shown below, along with a table detailing the specific values of the observation:

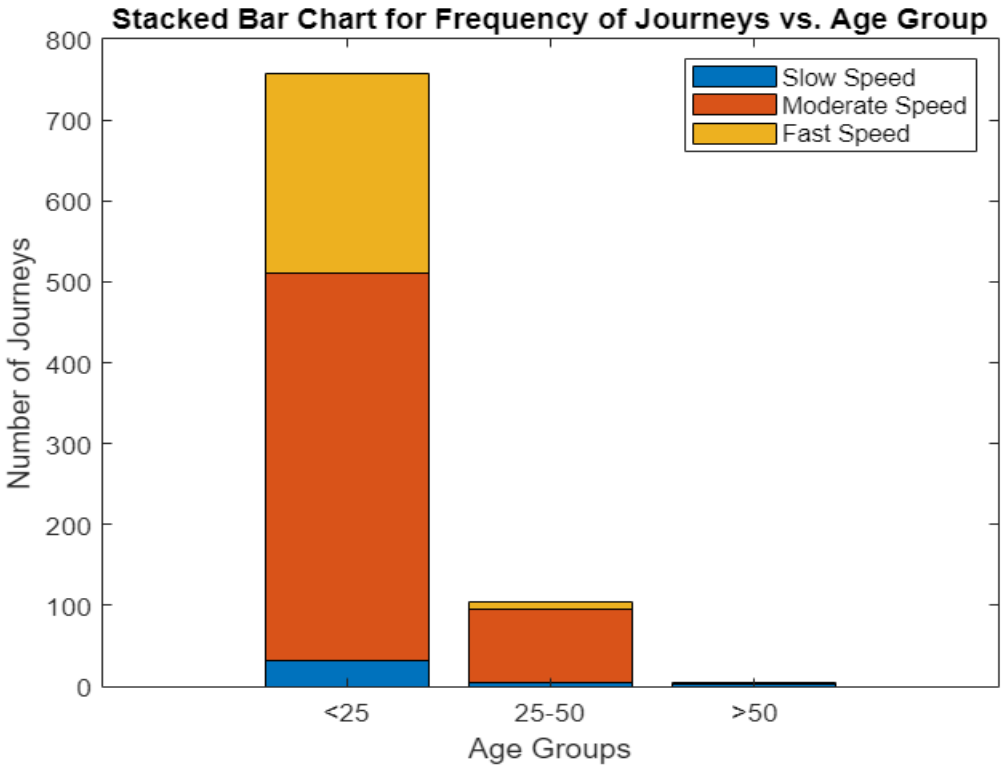

Figure 22: Stacked Bar Graph for walker frequency and age groups with stacked speeds for each age group.

Age Group / Speed	Slow	Medium	Fast
<25	32	478	248
25-50	4	92	8
>50	2	2	1
			Total Journeys: 867

Table 6: Breakdown of all journeys combined splitting up age groups and speed. The values in the table provide a 2-D array of all different numbers of walkers of varying age groups and speeds.

The results of this graph in Figure 22 allows us to see that there is a large proportion of journeys being made by a single Age Group (<25). This is not surprising as the area being observed is on a University Campus, so most walkers are likely within that Age Group – in this case around 87%. This means that due to a lack of data in the older age groups, ensuring the behaviour seen empirically versus the model output would need to be verified. However, the results did show that out of the 867 total journeys taken during both observation sessions, 96% of the walkers were moving at

either a medium or fast speed. Another trend that can be identified from the results is that within the 25-50 age group, most of the journeys are recorded at a medium speed. The results show that the model should have a large preference for simulating moderate speed walkers that are in the lower age group.

4.3 Detouring versus non-detouring

The other main trend that needed to be looked for is how a walker's properties may vary depending on if they took the detour or not. Finding a relationship between these factors will allow the model to determine whether a walker is likely to take a detour based on the parameters it has. This ability would enable the walker to be unique as in the decision-making process when deciding on which path to take. When working with the observation results, you can produce the following graph about the walker's parameters and detour choices:

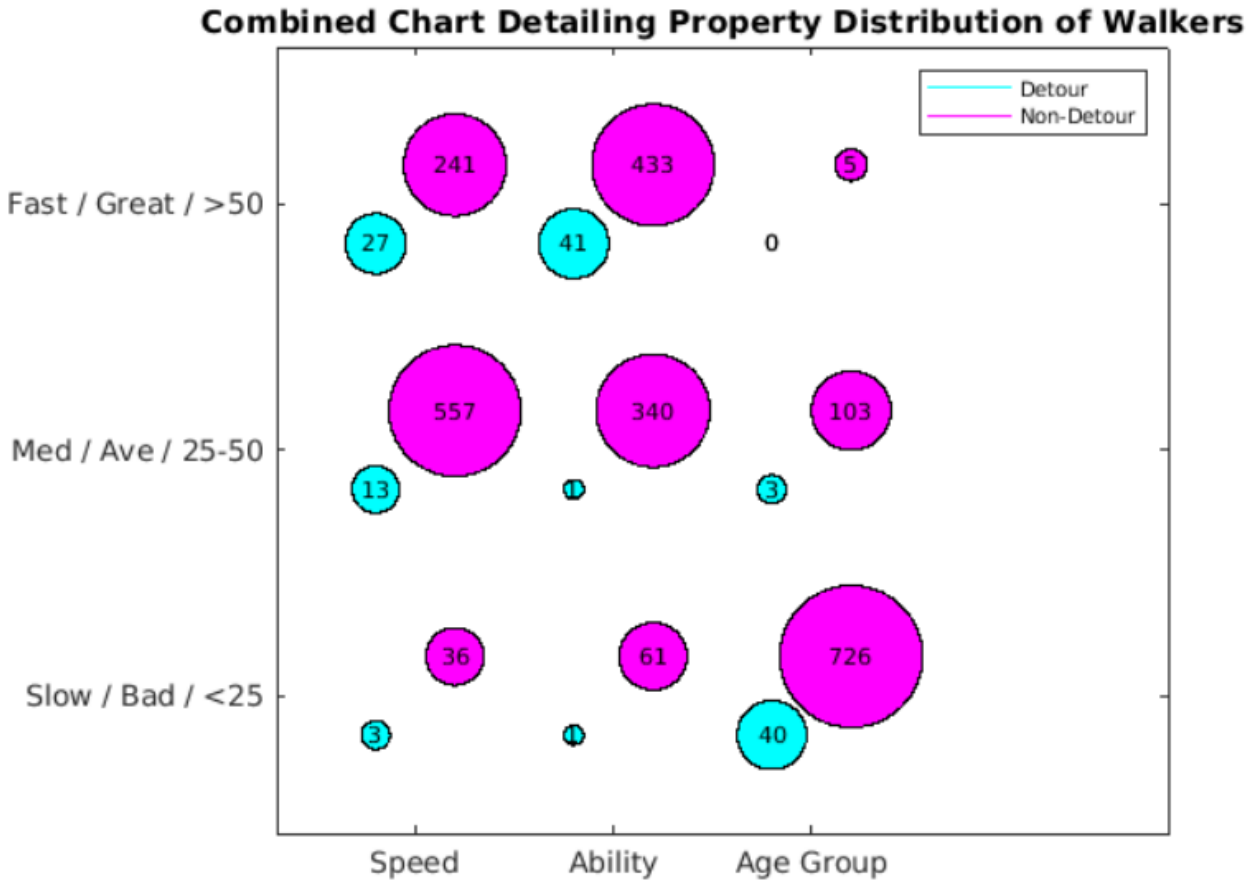

Figure 23: Bubble Graph for Walker parameters spread to see detoured walker parameter spread and non-detoured walker parameter spread. The two colours separate the walkers that detoured and the walkers that did not detour.

Detoured Walkers			
Parameter	1*	2*	3*
Speed	3	13	27
Ability	1	1	41
Age Group	40	3	0
			Total Detours: 43
Non-Detoured Walkers			
Parameter	1*	2*	3*
Speed	36	557	241
Ability	61	340	433
Age Group	726	103	5
			Total Non-Detours: 834

Table 7: Table detailing the bubbles values from Figure 23. Note the Asterisks (*) in the parameter row relate to the descriptors given in Table 3, and the values on the y-axis of the graph.

From the table, the total number of detours was only 43. This represented only 5% of all journeys. Despite this small share, there are enough data points to show a general trend for each of the walker's properties in finding the determining factors.

From Figure 23, quite a few developing trends can be observed. When focussing on the detoured walkers, the number of detours increases as the walker's speed increases. The walker's ability when deciding on whether to detour or not is almost exclusively limited to "Great". In terms of detouring depending on age, the results show that most detours occur within the "<25" age range, with very few in the middle age range.

When relating this to the non-detouring group which represents a much larger percentage of all journeys recorded, some slight differences between the spread of data are noticeable. Firstly, walker speed in the non-detouring group is centred around the medium speed, whereas in the detouring group it shifts towards a fast walker speed. This different distribution of speeds is could be a determining factor in whether the walker's detour or not. A similar effect can be seen within walking abilities where the non-walkers had a relatively even spread of journeys marked as "average" or "great", and the detoured walkers were almost exclusively "great".

Summarising the results from the study, we can assume that the changes to be made to the model need to reflect the following trends:

- Journeys "1,2 &3" are the main journeys being recorded, with a large portion of those in journey "1".
- Most walkers within the 25-50 age range have a moderate speed, and for <25 walkers, the ratio of speeds from slow/medium/fast fit approximately to the ratio 4:32:64% respectively.
- If a simulated walker were to detour, their parameters should be:
    - Age - Mostly <25, very few 25-50.
    - Ability – Almost exclusively "Great", very limited others.
    - Journey – Likely to be on journey "1", some on journeys "2" and "3".
    - Speed – likely to be fast or medium speed, unlikely slow.

4.4 Applying the Study Results

To apply the results from the study to the model, coding changes need to be made within the simulation. However, the map for the study location had to have the diagnostic tool check if there are many local maximums within the map. From the Figure below detailing the results from the diagnostic tool, there are only two areas of concern in the middle of the map in Figure 24. This is manageable as the simulation can track if any of the walker's stall and if they match the locations below, the scaling of the potentials can be edited to remove this.

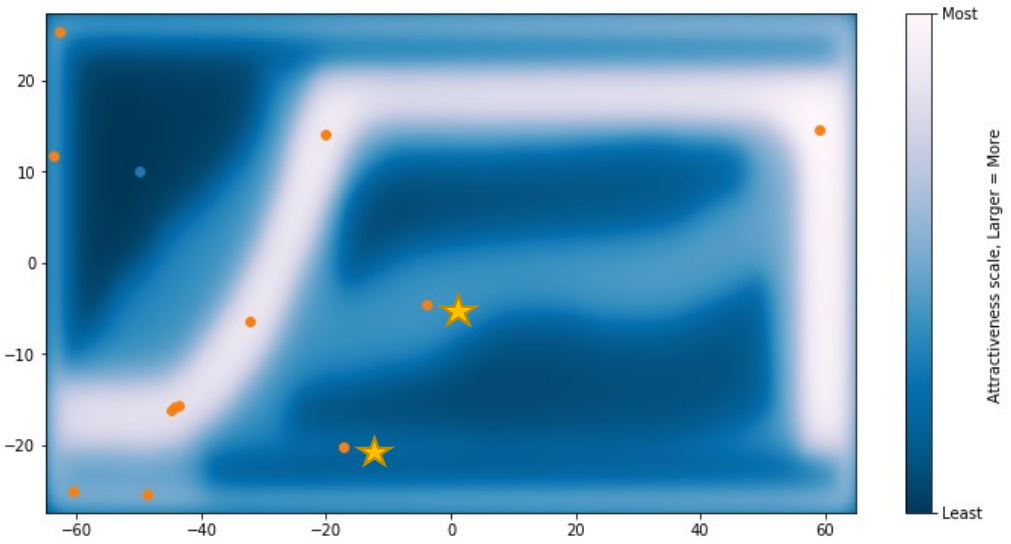

Figure 24: Diagnostic Tool output graph where the maximum points are orange circles, and the two maxima of interest are denoted by gold stars

The first change made to the model after checking the local maxima map was in the way the walker's path was chosen. Currently, a random number is generated up to the total number of routes then assigned to a given walker. This situation is not controllable, so changes need to be made for the simulation to correctly simulate the path's utilisation patterns as seen in the study. To do this, the code was changed where the walker's route is defined, and a new function was created.

A new function called "GenerateWalkerRoute" was created. It takes an input of a random integer between 0 and 99 and returns a value of either 1, 2, 3, 4 or 6, the

```
def GenerateWalkerRoute(routeRand):
    #Approx % of each route =   1: 57%    2: 7%    3: 33%    4: 2%    5: 0%     6: 1%
    if routeRand <57 : return 1
    elif routeRand <64 : return 2
    elif routeRand <97 : return 3
    elif routeRand <99 : return 4
    else : return 6
```

Figure 25: New function GenerateWalkerRoute, takes in a random integer and returns a random journey ID

routes that are used in the study. This can be implemented with the following logic:

The values in the Figure above correspond to the percentage chance that a walker takes a specific route on their journey. When implemented it will now mimic the walking patterns of the study in terms of approximately how many walkers move along each journey. This change also modifies the main running as the new value applied to the set_up_walker function is the returned value of the GenerateWalkerRoute function. This is seen below:

```
set_up_walker(GenerateWalkerRoute(np.random.randint(0,100)))
```

Figure 26: Code change for set_up_walker function, input is now the output of Figure 25

Following this implementation, the new parameters of Age and Walking Ability needed to be implemented to the model. To do this, new logic needs to be created to set up certain values that the simulation will use to run. The first step to achieve this was to set a parameter to base the others around. The parameter chosen for this is the Age of the walker, so the age of the walker must be determined.

To do this, the cumulative data that has been gathered from the study highlighted that approximately 87% of all walkers recorded were in the age group "<25", 12% in the age group "25-50" and 1% in the age group ">50". The code below in Figure 27

will return a value noting the age group of the walker when a random number from 0-999 is inputted. This recreates the trend shown with the age of the walkers with their relative percentages.

```
def GetWalkerAgeGroup(RandomAge):
    #Approx of each age = 1: 874    2: 120    3: 6
    if RandomAge < 874 : return 1
    elif RandomAge < 994 : return 2
    else : return 3
```

Figure 27: GetWalkerAgeGroup Function: Gets a random number from 0-999 and returns a random age group for the next walker. 0-999 to allow a resolution that makes the approximate distribution of ages.

After the age has been generated for the walker, the next step is to give it its unique speed and ability parameters. The first step for this was to use the same cumulative technique from the study data to define probabilities a walker will have each trait. This can be seen in the code below:

```
#Parameters for Walker Types
#Format   = [Slow, Medium, Fast] Cumulative Percentages
GroupSpeed0 = [4,67,100]
GroupSpeed25 = [4,92,100]
GroupSpeed50 = [40,80,100]

#Format = [Bad, Average, Great] Cumulative Percentages
GroupAbil0 = [8,45,100]
GroupAbil25 = [2,58,100]
GroupAbil50 = [0,60,100]

GroupParameters = [GroupSpeed0,GroupSpeed25,GroupSpeed50,GroupAbil0,GroupAbil25,GroupAbil50]
#print(GroupParameters)
```

Figure 28: New Code additions to specify what the proportions of values are for speed and ability depending on the age group of the walker. In this case, 0,25 and 50 represent the three age groups <25, 25-50 and >50 respectively.

The code above generates an array "GroupParameters" containing all the probabilities for the speed and ability parameters, using the age of the walker as the identifier. If the stated value of the walker's age is "1", the associated speed and ability probabilities are located within the first and fourth values of the "GroupParameters" array. They are split into three as there are three different age groups.

These two associated arrays are then sent into a final function called "GetWalkerRandomParameters". This functions job is to turn the arrays relating the study data into physical values that can then be used in the calculation of the

walker's path. To convert these values, they follow the same checking process as the age and route parameters where a random number is generated and compared to its proportions within the study data.

This can be seen implemented in the code below:

```
def GetWalkerRandomParameters(SpeedGroupChance, AbilityGroupChance):
    global WalkerSpeed,WalkerAbility
    Rand = np.random.randint(0,100, size = 2)
    #Array of 2 Rands 1 for speed, 1 for ability
    if Rand[0] < SpeedGroupChance[0]: WalkerSpeed = 0.8
    elif Rand[0] < SpeedGroupChance[1]: WalkerSpeed = 1
    else : WalkerSpeed = 1.2

    #ABILITY CALCULATING
    if Rand[1] < AbilityGroupChance[0]: WalkerAbility = 0.8
    elif Rand[1] < AbilityGroupChance[1]: WalkerAbility = 1
    else : WalkerAbility = 1.2
    return WalkerSpeed, WalkerAbility
```

Figure 29: The main function to determine the values of each parameter for a random walker, this step finalises the individualising of the walkers, all parameters are now random upon simulation.

Once these parameters have been calculated, all the required parameters of the study have now been generated into the code. At this point, the parameters now need to be placed in the right locations to correctly dictate the way the walker decides their walking route.

The routine that simulates the walkers has been modified to the version below, now generating a new walkers' parameters each cycle:

```
for i in range(0,50):
    calc_tr_new()
    intens[:]=0.
    for j in range(0,200):
        set_up_walker(GenerateWalkerRoute(np.random.randint(0,100))-1)   # -1 due to indexing of arrays
        #Above generates a new walker with a random route assigned
        Age = GetWalkerAgeGroup(np.random.randint(0,1000))-1             # -1 due to indexing of arrays
        #Above Returns the Age Group of the New Walker
        GetWalkerRandomParameters(GroupParameters[Age],GroupParameters[Age+3])
        #Gets the New Walkers random speed and ability parameters
        setup_potentials(WalkerAbility,WalkerSpeed)
        """print('Walker Age:', Age,'  WalkerSpeed:', WalkerSpeed,'  WalkerAbility', WalkerAbility, '  ', start,
              " -> ", dest, calc_path(WalkerSpeed))"""
        #print(i, start," -> ", dest, calc_path(WalkerSpeed))
    update_ground()
```

Figure 30: New logic for simulation of walkers, considers new parameters defined in the Figures above.

The final changes that were made to the model were to the calculation of the path, and the setting up of the potentials now that there were new parameters to account for. Within the setup_potentials procedure, both new parameters have been implemented then scaled with a noise term, to allow for a walker to be able to

detour with the same parameters. This matches what would be seen empirically. Also, the magic number from the initial model has been replaced with a variable describing its purpose within the code.

The code for this change can be seen below:

```
GradScaleFactor = 0.007
def setup_potentials(WalkerAbility,WalkerSpeed):
    WalkerSpeed += np.random.randint(-8,2)/1000
    WalkerAbility += np.random.randint(-8,2)/1000

    #print(WalkerAbility)
    global grad,desdirx,desdiry,dest
    grad=GradScaleFactor*np.array(np.gradient(TrailPotential)) * WalkerAbility * WalkerSpeed# Magic Number 0
    #grad=0.002*np.array(np.gradient(TrailPotential)) ORIGINAL

    #print (dest)
#Destination potential
    DestinationPotential=1.5 * -np.sqrt((dest[0]-x[:,None])**2+(dest[1]-y[None,:])**2)
```

Figure 31: Modified Setup_potentials function to account for new WalkerAbility and WalkerSpeed parameters. Using a noise term for each variable allow for each walker to get slightly different values for their parameters even if initially given the same slow/medium/fast speed parameter.

To account for the new walker speed parameter, the path calculation function has had the walker speed terms introduced to it. Doing this allows the model to apply a flat inertia increase to the walker. This change can reduce the amount of time it would take a walker to go from start to end destinations.

4.5 Final System Modelling

A small-scale simulation was run to test if the newly applied changes would result in a different simulation of the old model. A test simulation consisted of 10,000 walkers (50 cycles of 200 walkers) for both the old and new model, and the results of the simulation can be seen in Figure 32 on the following page.

It's possible to see within the new model map that over time new paths are beginning to form within the grassed area. This is not observed within the old model's map as the walkers are unable to detect any potential detour. This is due to the old model not being able to vary the walkers' parameters.

The small white paths that are beginning to evolve through the central section of the plot in Figure 32 show the detours off the pre-existing paths in the map. It is possible that if the simulation were run over a long period to simulate long-term footfall, these lines could continue to get closer together until a single desire path forms.

This behaviour would then fully match what was experienced from the observation, so you can assume that the model is behaving similarly to empirical evidence. Applying these results to a real-world model would mean the simulation can predict where paths might form given a map of the local ground, potentially saving in costs related to future work that may be required to create new paths.

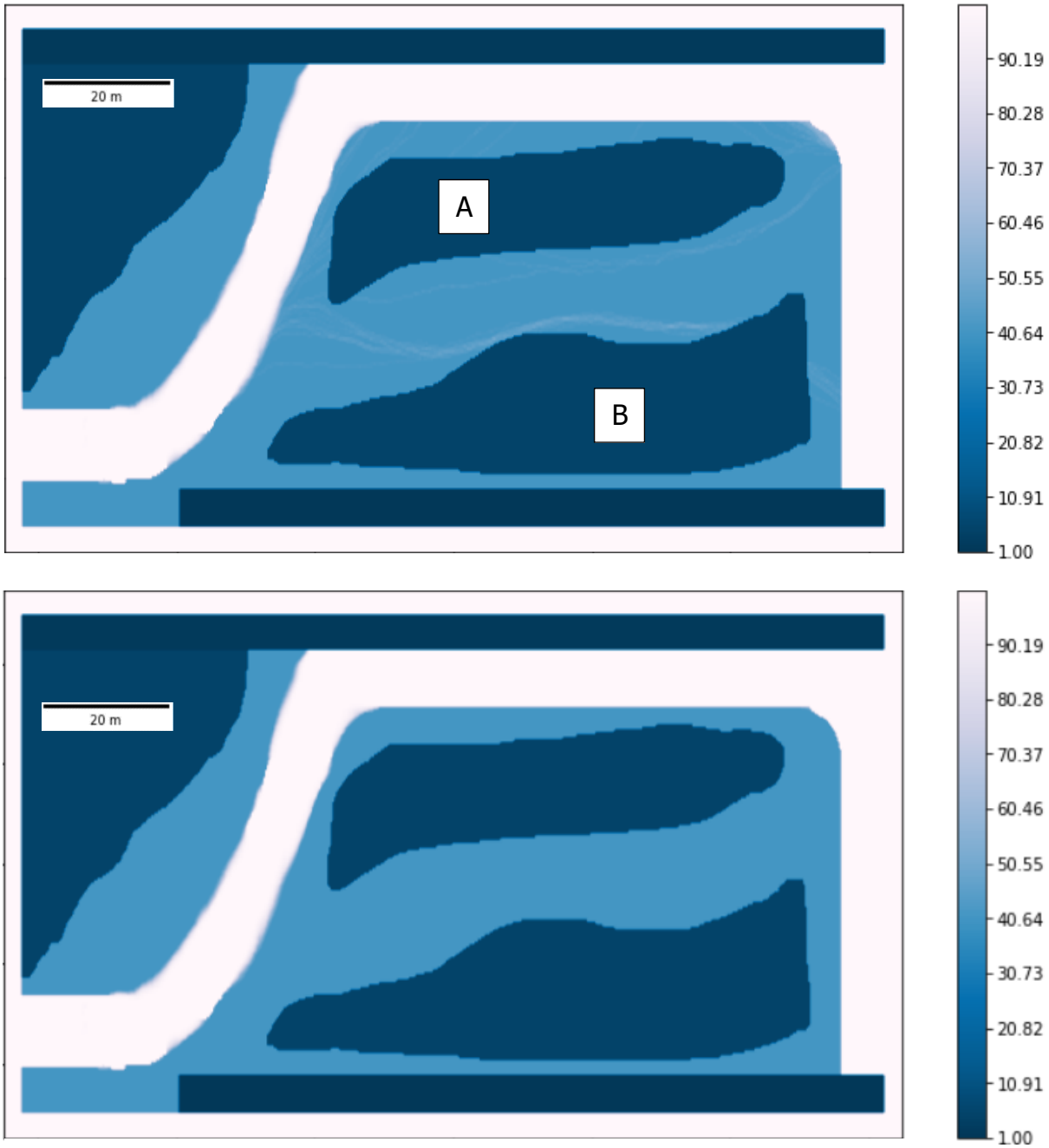

Figure 32: Plotted paths after 10000 walker simulation for both the extended model (top) and original model code (bottom). The white lines crossing along the middle signify walkers over time causing the potential in that area to increase. If this is left over periods of time, these faint lines can become bolder, potentially leading to a path forming between the two dark sections labelled A and B. This behaviour also shows that not all walkers are taking the detour. This mimics what is seen empirically. Scale bar in top left corner of each image relating image size relative to the actual study area size.

The extended model's code can be found in Appendix B in full and on GitHub. [1]

5. Conclusion

This project aimed to identify if modifying the Active Walker Model by Helbing et al. [2] by introducing new parameters to individualise the walkers, caused the model to produce more realistic results. This work was completed by performing a pedestrian traffic observation to develop new parameters for the model to use.

Analysis of this study data, when applied to the model, led to the detection of trends relating to the walkers of the study and the walker's attributes. These trends were then implemented into the model [2] to individualise the walkers. The results from the model confirmed that over time, similar walking patterns to those observed from the study were beginning to form.

Further research is needed to confirm the environmental conditions required for a detour to occur. Due to the limited scope of the project, data could only be captured during the winter months in a cold climate, so it was not possible to gather a complete set of data. It is likely that during the summer months of the year, the ground is more likely to be drier, potentially increasing the attractiveness of non-pathed surfaces. This would mean the relevance of the trail potential would then have to change to reflect this. However, the data that has been captured within this study was enough to create and test parameters across a moderately well sized set of utilisation patterns.

As a potential continuation of this research, it is possible to implement these changes into the modified model within the Mountain Trail Formation [12]. This could further increase the similarities between empirical evidence and a simulated model. This implementation could involve combining the new simulation logic from [12] in restricting the walker's movement across large elevation changes with the newly variable walkers to create an even more integrated model.

[1] GitHub Repository: https://github.com/HarryG18/DesirePathFormation

6. References

[1] K. Kohlstedt, "Least Resistance: How Desire Paths Can Lead to Better Design," https://99percentinvisible.org/, 25 01 2016. [Online]. Available: https://99percentinvisible.org/article/least-resistance-desire-paths-can-lead-better-design/. [Accessed 9 January 2020].

[2] D. Helbing, F. Schweitzer, J Keltsch, P. Molnar, "Active Walker Model for the Formation of Human and Animal Trail Systems," Physical Review E, 1997.

[3] N. Munier, "Risk Management for Engineering Projects," in *Risk Assessment and Analysis*, Switzerland, Springer, Cham, 2014, pp. 113-167.

[4] M. Schenk, "Untersuchungen zum Fußgängerverhalten," University of Stuttgart, Stuttgart, 1995.

[5] E. Dorato, G. Lobosco, "Designing Desire. A Parametric Approach to the Planning of Landscape Paths," 23 March 2017. [Online]. Available: http://convergencias.esart.ipcb.pt/?p=article&id=271. [Accessed 10 November 2019].

[6] F. Lamiraux, J.P. Laumond, M. Campana, "A simple path optimization method for motion planning," 23 October 2015. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01137844v2. [Accessed 18 January 2020].

[7] M. Turner, "Supply Path Optimisation: the Future of Programmatic in the UK," DQ&A, 9th January 2020. [Online]. Available: https://dqna.com/supply-path-optimisation-uk/. [Accessed 4 February 2020].

[8] H. O. Pollak, E. N. Gilbert, "Steiner Minimal Trees," *SIAM Journal on Applied Mathematics,* vol. 16, no. 1, pp. 1-29, 2006.

[9] K. Mehlhorn and P. Sanders, "Chapter 10: Shortest Paths," in *Algorithms and Data Structures: The Basic Toolbox.*, Springer-Verlag Berlin Heidelberg, 2008, pp. 191-216.

[10] K. Mehlhorn, P.Sanders, "Chapter 11: Minimum Spanning Trees," in *Algorithms and Data Structures: The Basic Toolbox*, Springer-Verlag Berlin Heidelberg, 2008, pp. 217-233.

[11] T. M. Gureckis, R. L. Goldstone, "Collective Behavior," *Topics in Cognitive Science,* vol. 1, no. 3, pp. 412-438, 2019.

[12] J. Hague, S. J. Gilks, "Mountain trail formation and the active walker model," *International Journal of Modern Physics C,* 2012.

[13] C. Ankowitsch, "trails," Ankowitsch, 14 August 2010. [Online]. Available: http://www.ankowitsch.de/2010/08/trampelpfade/. [Accessed 23 February 2020].

[14] P. Molnar, I. J. Farkas, K Bolay, D. Helbing, "Self-organizing pedestrian movement," *Environment and Planning B Planning and Design,* vol. 28, pp. 361-383, 2001.

[15] L. Nichols, "Social desire paths: a new theoretical concept to increase the usability of social science research in society.," *Theory and Society,* vol. 43, no. 6, 2014.

[16] J. Jessurun, H. J. P. Timmermans, J. Dijkstra, "A Multiagent cellular automata model of pedestrian movement, Pedestrian and Evacuation Dynamics," pp. 173-181, 2001.

[17] D. J. Kaup, Neal M. Finkelstein, Taras I. Lakoba, "Modifications of the Helbing-Molnár-Farkas-Vicsek Social Force Model for Pedestrian Evolution," *Simulation,* vol. 81, no. 5, pp. 339-352, 2005.

[18] P. K. Baran, Y. Zhai, "Urban Park Pathway Design Characteristics and Senior Walking Behavior," *Urban Forestry & Urban Greening,* vol. 21, pp. 60-73, 2016.

[19] K. Browne, B. Whitaker, Parks for People, Seeley, 1971.

[20] W. Cunningham, "Magic Number," 8 May 2013. [Online]. Available:

https://wiki.c2.com/?MagicNumber. [Accessed 26 February 2020].

[21] The SciPy Community, "Multi-dimensional image processing (scipy.ndimage)," The SciPy Community, 19 December 2019. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/ndimage.html. [Accessed 23 February 2020].

[22] unutbu, "How to find the local minima of a smooth multidimensional array in NumPy efficiently?," Stack Overflow, 21 October 2010. [Online]. Available: https://stackoverflow.com/questions/3986345/. [Accessed 7 January 2020].

[23] Warwick University, "The Slate," Warwick University, 20.

[24] D. C. Blanchard, "Steiner Minimum Tree Example," WikiCommons, 2014.

[25] Google Maps, "Virginia Tech (satellite)," Google Maps, 2020. [Online]. Available: https://www.google.com/maps/place/Virginia+Polytechnic+Institute+and+State+University/@37.2283886,-80.4256054,17z/. [Accessed 23 February 2020].

[26] Google Maps, "University of Warwick - Coventry," 2020. [Online]. Available: https://www.google.com/maps/place/University+of+Warwick/@52.3808275,-1.5667727,162m/data=!3m1!1e3!4m5!3m4!1s0x48774ac696d53ee5:0xaa928d75708b2b54!8m2!3d52.3792525!4d-1.5614704. [Accessed 27 February 2020].

[27] Google Maps, "University of Warwick - Coventry (Satellite)," 2020. [Online]. Available: https://www.google.com/maps/place/University+of+Warwick/@52.3808275,-1.5667727,19z/data=!4m5!3m4!1s0x48774ac696d53ee5:0xaa928d75708b2b54!8m2!3d52.3792525!4d-1.5614704. [Accessed 27 February 2020].

[28] E. Gumble, ES327 project report, School of Engineering, University of Warwick (2016)

[29] H. Brown, ES327 project report, School of Engineering, University of Warwick (2017)

Appendix A: Ethical Approval Letter

PRIVATE
H. Grace
School of Engineering
University of Warwick
Coventry
CV4 7AL

Supervisor: P. Brommer

10 December 2019

Dear H. Grace,

Study Title and Eng-Ethics Reference: *Desire Path Formation,* REGO-2019-ENG-020

Thank you for submitting the above-named project to the University of Warwick, Eng-Ethics, a sub-committee of the Biomedical and Scientific Research Ethics Committee for research ethical review.

I am pleased to advise that research ethical approval is granted.

In undertaking your study, you are required to comply with the University of Warwick's *Research Data Management Policy*, details of which may be found on the Research and Impact Services' webpages, under "Codes of Practice & Policies" » "Research Code of Practice" » "Data & Records" » "Research Data Management Policy", at:
http://www2.warwick.ac.uk/services/ris/research_integrity/code_of_practice_and_policies/research_code_of_practice/datacollection_retention/research_data_mgt_policy

You are also required to comply with the University of Warwick's Information Classification and Handling Procedure, details of which may be found on the University's Governance webpages, under "Governance" » "Information Security" » "Information Classification and Handling Procedure", at:
http://www2.warwick.ac.uk/services/gov/informationsecurity/handling.

Investigators should familiarise themselves with the classifications of information defined therein, and the requirements for the storage and transportation of information within the different classifications:

Information Classifications:
http://www2.warwick.ac.uk/services/gov/informationsecurity/handling/classifications
Handling Electronic Information:
http://www2.warwick.ac.uk/services/gov/informationsecurity/handling/electronic/
Handling Paper or other media
http://www2.warwick.ac.uk/services/gov/informationsecurity/handling/paper/.

School of Engineering
University of Warwick
Coventry CV4 7AL UK
Tel: +44(0) 24 765 23523
www.warwick.ac.uk

Appendix B: Code Listing

```
#Authors: Eli Gumble, Peter Brommer, Harry Brown
#Modified by: Harry Grace
#Initialisation
import matplotlib.pyplot as plt
import numpy as np
np.set_printoptions(threshold=np.inf)
from scipy.integrate import simps
from scipy import signal as sg
from scipy.interpolate import RectBivariateSpline as ReBiSpline
from numpy import ma
from matplotlib import colors, ticker, cm
from random import choice
import scipy.ndimage.filters as filters
import scipy.ndimage.morphology as morphology
import timeit
from PIL import Image
get_ipython().run_line_magic('matplotlib', 'inline')


# Read grids from image
im = Image.open("StudyArea.bmp")
Base = np.array(im)

scale = 0.25 #m per pixel
Nx = Base[:,0,0].size #N appears to be resolution
Ny = Base[0,:,0].size
xmin=-scale*0.5*(Nx-1)
xmax=scale*0.5*(Nx-1)
ymin=-scale*0.5*(Ny-1)
ymax=scale*0.5*(Ny-1)
x = np.linspace(xmin, xmax, Nx) # This is defining the axes and full space
y = np.linspace(ymin, ymax, Ny)
Y, X= np.meshgrid(y, x)
TrailPotential = np.zeros((Nx,Ny))
DestinationPotential=np.zeros((Nx,Ny))
Weight=np.zeros((Nx,Ny))
intens=np.zeros((Nx,Ny))
q_alpha=np.zeros((Nx,Ny))
expdist=np.zeros((2*Nx-1,2*Ny-1))
dest=np.zeros((2))
start=np.zeros((2))
grad=np.zeros((2,Nx,Ny))
vel=np.asarray([0.,0.])
pos=np.asarray([0.,0.])
#desdirx=ReBiSpline(x,y,grad[0,:,:],s=2)
#desdiry=ReBiSpline(x,y,grad[1,:,:],s=2)
intens[:]=0.

#print(route)
#parameters
t_track=50.
dt=0.1
tau=5.
isigma=3./5.
conv_thresh=10.e-4
precision=1.**2 #distance to target.
eps=0.025 #random motion contribution, same for all

#Show Map
plt.imshow(Base)
```

```
61.
62. #Create blue channel colours
63. z = np.zeros((Nx,Ny))
64. g_max=np.zeros((Nx,Ny))
65. g_nat=np.zeros((Nx,Ny))
66. g_nat=np.maximum(np.ones_like(g_nat),np.float64(Base[:,:,0]))
67. g_max=np.maximum(np.ones_like(g_max),np.float64(Base[:,:,1]))
68. z=g_nat
69.
70. # Trails (start and end point) For current Map
71. route = np.array([[[-53.,-17.],[ 60., 14.]],   #Journey 1
72.                  [[-53.,-17.],[ 60.,-17.]],   #Journey 2
73.                  [[ 60., 14.],[-53.,-17.]],   #Journey 3
74.                  [[ 60., 14.],[ 60.,-17.]],   #Journey 4
75.                  [[ 60.,-17.],[-53.,-17.]],   #Journey 5
76.                  [[ 60.,-17.],[ 60., 14.]]])  #Journey 6
77.
78.
79. #Setup weight matrix, here trapezoid rule.
80. Weight[:,:]=1
81. Weight[1:-1,:]=2
82. Weight[:,1:-1]=2
83. Weight[1:-1,1:-1]=4
84. Weight*=0.25*((x[-1]-x[0])/(Nx-1))*((y[-1]-y[0])/(Ny-1))
85.
86. # Setup distance matrix
87. for xi in range(1,Nx+1):
88.     for yi in range(1,Ny+1):
89.
90.         expdist[xi-1,yi-1]=np.exp(-isigma*np.sqrt((x[Nx-xi]-xmin)**2+(y[Ny-yi]-
    ymin)**2))
91.         expdist[-xi,-yi]  = expdist[xi-1,yi-1]
92.         expdist[-xi,yi-1] = expdist[xi-1,yi-1]
93.         expdist[xi-1,-yi] = expdist[xi-1,yi-1]
94.
95. # find index range > conv_thresh
96. subexpdist=expdist[(expdist>conv_thresh).any(1)]
97. subexpdist=subexpdist[:, np.any(subexpdist>conv_thresh, axis=0)]
98. #subexpdist=subexpdist[:,np.any(subexpdist>conv_thresh, axis=0)]
99.
100.
101.     def calc_tr_new():
102.         TrailPotential[:,:]=sg.convolve2d(z[:,:]*Weight[:,:],subexpdist[:,:],mode="
    same")
103.
104.     timeit.timeit(calc_tr_new,number=1)
105.
106.     plt.imshow(np.absolute(TrailPotential), cmap='PuBu_r')
107.
108.
109.     # Defines a Plot to show the smoothing of the supplied map to represent the res
    pective potentials of the ground, the larger
110.     # the potentials, the more attractive the ground is to the walker
111.     plt.figure(figsize = (12,6))
112.     cs = plt.contourf(X, Y, TrailPotential, levels=np.linspace(TrailPotential.min()
    ,TrailPotential.max(),1000),cmap='PuBu_r')
113.     cbar = plt.colorbar()
114.
115.     plt.scatter(track[0:1999,0],track[0:1999,1])
116.     plt.scatter(start[0], start [1])
117.     plt.scatter(dest[0], dest[1])
118.     plt.show()
119.
120.
121.     #set up walker
122.     def set_up_walker(route_id):
```

```
123.         global vel,pos,track,intens,dest,start,route
124.         #start
125.         start=np.array(route[route_id,0,:])
126.         #dest=(random.choice(ends))
127.         dest=np.array(route[route_id,1,:])
128.         vel=np.array([0.,0.])#set empty velocity
129.         pos=np.array(start)
130.         #print (pos)
131.         track=np.zeros((2000,2))
132.
133.
134.     #Calculate gradients eq 19
135.     #Trail gradient
136.     def setup_potentials():
137.         WalkerAbility += np.random.randint(-8,2)/1000
138.         WalkerAbility += np.random.randint(-8,2)/1000
139.         global grad,desdirx,desdiry,dest
140.         grad=0.007*np.array(np.gradient(TrailPotential)) * WalkerAbility * WalkerSp
    eed# Magic Number 0
141.         #grad=0.002*np.array(np.gradient(TrailPotential)) ORIGINAL
142.
143.         #Destination potential
144.         DestinationPotential=1.5 * -np.sqrt((dest[0]-x[:,None])**2+(dest[1]-
    y[None,:])**2)
145.         #Combine gradients
146.         grad+= np.array(np.gradient(DestinationPotential)[:]) # Magic Number 1
147.         #Normalise
148.         #grad[:,:,:]/=(np.sqrt(grad[0,:,:]**2+grad[1,:,:]**2))
149.         desdirx=ReBiSpline(x,y,grad[0,:,:],s=2)
150.         desdiry=ReBiSpline(x,y,grad[1,:,:],s=2)
151.
152.     ##PLOTTING
153.     #Plot the direction
154.     Directions = ['W', 'SW' , 'S' ,'SE', 'E' , 'NE' ,'N', 'NW' ,'W'] # Array of Dir
    ections
155.     DirectionAngles = [-180, -135, -90, -
    45, 0, 45, 90, 135, 180] # Array of Angles representing Directions
156.
157.     scgrad=np.arctan2(grad[1],grad[0]) * 180/np.pi #convert Rads to Degrees
158.     levels = np.linspace(-180, 180, 360) # -
    180 to 180 = 360 increments of 1 degree
159.     plt.figure(figsize=(12,6))
160.     cs = plt.contourf(X, Y,scgrad, levels = levels, cmap='hsv')
161.     #Define Both Colourbars
162.     cbar = plt.colorbar(cs, ticks = DirectionAngles)# Set colourbar to be active wi
    th ticks at each angle increment of direction
163.     cbar2 = plt.colorbar(cs)
164.     cbar.ax.set_yticklabels(Directions) # Set Lables of Colour Bar to Directions of
     Travel
165.     cbar.set_label("Direction to Travel") # Label ColourBar Axis
166.
167.     plt.scatter(track[0:1999,0],track[0:1999,1])
168.     #Plot Start and end Co-ords of specific walker
169.     #plt.scatter(start[0], start [1])
170.     plt.scatter(dest[0], dest[1])
171.     plt.show()
172.
173.
174.     def calc_path():
175.         global pos,vel,intens,track,dest,dvel,tau
176.         i=0
177.         hist=10
178.         samp=10
179.         avpos=np.zeros((2,hist))
180.         #Setup While loop to run until either the walker reaches the destination or
     the walker has passed 2400 movement cycles to
```

```
        #attempt to get there 2400 fits any journey in the study area
        while (np.dot(pos-dest,pos-dest)>precision and i<2400):

            #if (i%samp==0): avpos[:,(i%hist)//samp]=pos[:] #ORIGINAL
            if (i%samp==0): avpos[:,(i%(hist*samp))//samp]=pos[:] #FIXED

            pos+=dt*vel

            gradmagnitude=max(0.0001,np.sqrt(desdirx(pos[0],pos[1])**2+desdiry(pos[
0],pos[1])**2))
            xi=np.array(np.random.normal(0,1,2))

            vel[0]+= -
1/tau*vel[0] + (1/tau) * desdirx(pos[0],pos[1])/gradmagnitude+np.sqrt(2.*eps/tau)*xi
[0]
            vel[1]+= -
1/tau*vel[1] + (1/tau) * desdiry(pos[0],pos[1])/gradmagnitude+np.sqrt(2.*eps/tau)*xi
[1]
            #vel[0]+=-
1/tau*vel[0] + (dvel/tau)*desdirx(pos[0],pos[1])/gradmagnitude+np.sqrt(2.*eps/tau)*x
i[0]
            #vel[1]+=-
1/tau*vel[1] + (dvel/tau)*desdiry(pos[0],pos[1])/gradmagnitude+np.sqrt(2.*eps/tau)*x
i[1]

            #Set the current position of the walker into the track array for the cu
rrent iteration
            track[i,:]=pos[:]
            intens[int((pos[0]-xmin)*(Nx-1)/(xmax-xmin)),int((pos[1]-ymin)*(Ny-
1)/(ymax-ymin))]+=1.
            i+=1
            if (i%(hist*samp)==0):
                meanpos=np.mean(avpos,axis=1)
                if (np.dot(pos-meanpos,pos-meanpos)<precision):
                    print ("Stalled progress ",pos,meanpos,vel, dest)
                    break
        if (i==2000): print ("Missed goal ",dest,pos)
        return i

    # Calculate Q_alpha (strength of markings) eq 15
    def update_ground():
        global q_alpha,intens,z,g_max,t_track,g_nat
        q_alpha=intens*(1.-z/g_max)
    # Time evolution of ground potential
        #zdiff=(1./t_track)*(g_nat-z)+q_alpha
        z+=(1./t_track)*(g_nat-z)+q_alpha
        #cs = plt.contourf(X, Y, zdiff, cmap=cm.PuBu_r)
        #cbar = plt.colorbar()
        #plt.show
        #z[140:160,45:75]


    def plot_path():
        plt.figure(figsize=(12,6))
        plt.contourf(X, Y, z, levels=np.linspace(z.min(),z.max(),1000),cmap='PuBu_r
')
        plt.colorbar()
        #plt.scatter(track[0:1999,0],track[0:1999,1],1)
        plt.show(block=False)


    def GenerateWalkerRoute(routeRand):
        #Approx % of each route =   1: 57%    2: 7%    3: 33%    4: 2%    5: 0%
    6: 1%
        if routeRand <57 : return 1
        elif routeRand <64 : return 2
```

```
234.         elif routeRand <97 : return 3
235.         elif routeRand <99 : return 4
236.         else : return 6
237.
238.
239.     def GetWalkerAgeGroup(RandomAge):
240.         #Approx of each age = 1: 874    2: 120    3: 6
241.         if RandomAge < 874 : return 1
242.         elif RandomAge < 994 : return 2
243.         else : return 3
244.
245.
246.     #Parameters for Walker Types
247.     #Format   = [Slow, Medium, Fast] Cummulative Percentages
248.     GroupSpeed0 = [4,67,100]
249.     GroupSpeed25 = [4,92,100]
250.     GroupSpeed50 = [40,80,100]
251.
252.     #Format = [Bad, Average, Great] Cummulative Percentages
253.     GroupAbil0 = [8,45,100]
254.     GroupAbil25 = [2,58,100]
255.     GroupAbil50 = [0,60,100]
256.
257.     GroupParameters = [GroupSpeed0,GroupSpeed25,GroupSpeed50,GroupAbil0,GroupAbil25
    ,GroupAbil50]
258.     #print(GroupParameters)
259.
260.
261.     def GetWalkerRandomParameters(SpeedGroupChance, AbilityGroupChance):
262.         global WalkerSpeed,WalkerAbility
263.         Rand = np.random.randint(0,100, size = 2)
264.         #Array of 2 Rands 1 for speed, 1 for ability
265.         if Rand[0] < SpeedGroupChance[0]: WalkerSpeed = 0.8
266.         elif Rand[0] < SpeedGroupChance[1]: WalkerSpeed = 1
267.         else : WalkerSpeed = 1.2
268.
269.         #ABILITY CALCULATING
270.         if Rand[1] < AbilityGroupChance[0]: WalkerAbility = 0.8
271.         elif Rand[1] < AbilityGroupChance[1]: WalkerAbility = 1
272.         else : WalkerAbility = 1.2
273.         return WalkerSpeed, WalkerAbility
274.
275.
276.     for i in range(0,4):
277.         calc_tr_new()
278.         intens[:]=0.
279.         for j in range(0,40):
280.             set_up_walker(np.random.randint(0,len(route)))
281.             setup_potentials()
282.             print (i, start," -> ", dest, pos, calc_path())
283.         update_ground()
284.
285.     plot_path()
286.
287.     def detect_local_maxima(arr):
288.         # https://stackoverflow.com/questions/3684484/peak-detection-in-a-2d-
    array/3689710#3689710
289.         # http://www.scipy.org/doc/api_docs/SciPy.ndimage.morphology.html#generate_
    binary_structure
290.         neighborhood = morphology.generate_binary_structure(len(arr.shape),2) # def
    ine an connected neighborhood
291.
292.         # apply the local minimum filter; all locations of minimum value
293.         # in their neighborhood are set to 1
294.         # http://www.scipy.org/doc/api_docs/SciPy.ndimage.filters.html#minimum_filt
    er
```

```
295.         local_max = (filters.maximum_filter(arr, footprint=neighborhood)==arr)
296.
297.         # local_max is a mask that contains the peaks we are looking for, but also
    the background.
298.         # In order to isolate the peaks we must remove the background from the mask
    .
299.         background = (arr==0) # create the mask of the background
300.
301.         # a little technicality: we must erode the background in order to successfu
    lly subtract it from local_min,
302.         # otherwise a line will appear along the background border (artifact of the
     local minimum filter)
303.         # http://www.scipy.org/doc/api_docs/SciPy.ndimage.morphology.html#binary_er
    osion
304.         eroded_background = morphology.binary_erosion(background, structure=neighbo
    rhood, border_value=1)
305.
306.         # we obtain the final mask, containing only peaks, by removing the backgrou
    nd from the local_min mask
307.         detected_maxima = local_max ^ eroded_background
308.         return np.where(detected_maxima)
309.
310.
311.     def plot_potentials():
312.         global dest, PlotTickNo
313.         TotalPot = np.zeros((Nx,Ny)) # Define Blank Arrays of Map Size
314.         TotalPot -= 0.8*np.sqrt((dest[0]-x[:,None])**2+(dest[1]-
    y[None,:])**2) # Combine Equ from paper
315.         TotalPot += 0.4*TrailPotential # Combine Equ from paper
316.         #0.8 & 0.4 are scaling potentials for the function
317.
318.         Maximums=detect_local_maxima(TotalPot) # Run Detection Function to return a
    rray of Maximums
319.
320.         PlotTicks = ['Least','Most']; # Text for Labels for Colourbar
321.         PlotTickNo = [TotalPot.min(),TotalPot.max()]; # Setup array of Min and Max
    values
322.         plt.figure(figsize=(12,6)) # Set Figure Size to 12in x 6in
323.         # Plot Potential Map
324.         ColourDepth = int((PlotTickNo[1]-
    PlotTickNo[0])) % 1000 # Create Scale For Pot Map with max value of 1000,
325.         #Scales Range of PlotTickNo by 10
326.         cs = plt.contourf(X, Y, TotalPot, levels=np.linspace(PlotTickNo[0],PlotTick
    No[1],ColourDepth),cmap='PuBu_r') # Plot Pot Map
327.         cbar = plt.colorbar(cs, ticks = PlotTickNo) # Set Colour bar to only have 2
     values
328.         cbar.ax.set_yticklabels(PlotTicks) # Set Lables of Colour Bar to Directions
     of Travel
329.         cbar.set_label("Attractiveness scale, Larger = More ") # Label ColourBar Ax
    is
330.         #print(Maximums) #Can Print maximums if you want to see them in array form

331.         plt.scatter(-50,10)
332.         plt.scatter(x[Maximums[0]],y[Maximums[1]]) # Plot onto the graph the points
     in Maximums Array
333.         plt.show # Show Graph
334.
335.
336.     plot_potentials()
```