UNIVERSITY OF CAMBRIDGE

MPhil in Machine Learning and, Machine Intelligence 2018-2019

Module Coursework Feedback

Module Title: Robotics

Module Code: 4M20

Candidate Number: F606F

Coursework Number: 2

I confirm that this piece of work is my own unaided effort and adheres to the Department of Engineering's guidelines on plagiarism. √

Date Marked: Click here to enter a date. Marker's Name(s): Click here to enter text.

Marker's Comments:

This piece of work has been completed to the following standard *(Please circle as appropriate)*:

	Distinction			Pass			Fail (C+ - marginal fail)		
Overall assessment (circle grade)	**Outstanding**	**A+**	**A**	**A-**	**B+**	**B**	**C+**	**C**	**Unsatisfactory**
Guideline mark (%)	90-100	80-89	75-79	70-74	65-69	60-64	**55-59**	**50-54**	**0-49**
Penalties	**10% of mark for each day, or part day, late (Sunday excluded).**								

The assignment grades are given **for information only**; results are provisional and are subject to confirmation at the Final Examiners Meeting and by the Department of Engineering Degree Committee.

Question 1

Given that the Arduino control scales the possible achievable angles of the motor from 0 to 1, a function that can transform the angle input into the unit output is needed, which allows us to control the motor angle accurately and precisely.

It is found that the motor has around 180° accessible angle range, the outer ring which connects the arm with the motor output is adjusted to achieve the designed angle trajectories which facilitate the motion.

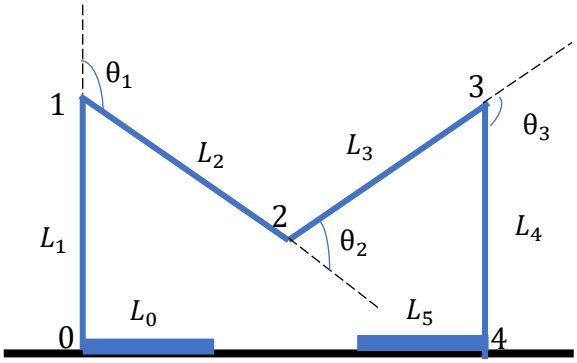


Figure 1: The robot architecture.

Consider the material provided, the length of arm is $0.14m(L_1, L_2, L_3, L_4)$ and the length of the foot is $0.2m(L_0, L_5)$. Again, the inverse kinematic is used to compute the joint angles from rear (4) and front (0) feet locations.

When we want to move the rear foot, we assume the front foot is fixed on the ground and set it as the fixed reference point $(0,0)$ so that the range of x for the rear foot is $(0.2,0.28)$m.

$$\begin{pmatrix}\theta_1\\ \theta_2\\ \theta_3\end{pmatrix} = \begin{pmatrix}\pi - \theta_2 - \theta_3\\ cos^{-1}\left(\frac{x^2+y^2}{2L^2} - 1\right)\\ sin^{-1}\left(\frac{x}{L} + \frac{s_2}{1+c_2}\frac{y}{L}\right)\end{pmatrix}$$

, where θ_i the joint angle between arms L_i and L_{i+1} , and L the length of the arm.

When moving the front foot, the rear foot is assumed to be fixed and is set as the reference point $(0,0)$, so the range of x for the front foot is $(-0.28, -0.2)m$.

$$\begin{pmatrix}\theta_1\\ \theta_2\\ \theta_3\end{pmatrix} = \begin{pmatrix}sin^{-1}\frac{1}{2}\left(\frac{-x}{L} + \frac{s_2}{1+c_2}\frac{y}{L}\right)\\ cos^{-1}\left(\frac{x^2+y^2}{2L^2} - 1\right)\\ \pi - \theta_1 - \theta_2\end{pmatrix}$$

The maximum separation between the front and rear feet is $0.28m$, this corresponds to the angles $(\theta_1, \theta_2, \theta_3) = (45.6^{\circ}, 88.8^{\circ}, 45.6^{\circ})$, and the minimum separation is $0.2m$ which corresponds to the angles $(\theta_1, \theta_2, \theta_3) = (90^{\circ}, 0^{\circ}, 90^{\circ})$. Ideally, we also wish to raise the rear foot forwards by $(0.28 - 0.2)/2 = 0.04m$ and upwards by $0.05m$ which is the height of the stair or the obstacle from the maximum separation pose. Similarly, for the front foot, what differs is it is raised from the minimum separation pose. Their corresponding angles are $(49.3^{\circ}, 57.8^{\circ}, 72.3^{\circ})$ and $(72.3^{\circ}, 57.8^{\circ}, 49.3^{\circ})$.

So, we calibrate the motors in the way that the motor 1 has the angle range $(\theta_{1,min}, \theta_{1,max}) = (-147^{\circ}, -5^{\circ})$, the motor 2 has angle range of $(-70^{\circ}, 90^{\circ})$ and the motor 3 has the range of $(-50^{\circ}, 125^{\circ})$. And using the equation below, the angle input for each motor will be converted to the unit output,

$$Unit_i = \frac{\theta_i - \theta_{i,min}}{\theta_{i,max} - \theta_{i,min}}$$

, where θ_i the input angle of motor i, $\theta_{i,min}$ the minimum calibrated angle of motor i, $\theta_{i,max}$ the maximum calibrated angle of motor i and $Unit_i$ the output unit for the motor i readable by the Arduino system.

The following matlab functions are then written to compute the units automatically,

```
function unit = angle_to_unit(angle, mode)
if mode == 1
    unit = (angle-147)/(-5-147);
elseif mode == 2
    unit = (angle+70)/(90+70);
elseif mode == 3
    unit = (angle+50)/(125+50);
end
end
```

And execute the command to drive the motor via the Arduino system,

```
function action(an1,an2,an3,svf,svm,svr,second)
un1in = angle_to_unit(an1,1);
un2in = angle_to_unit(an2,2);
un3in = angle_to_unit(an3,3);
writePosition(svf, un1in);
writePosition(svm, un2in);
writePosition(svr, un3in);
pause(second);
end
```

However, this approach of using the inverse kinematics to control the motors is not working, since the influence of gravity and the maximum torque output of the motor are not considered. The motors cannot reach the angles as expected.

To explain this, the theory of moment of inertia is used. Assume that the total weight lifted by a motor i is the sum of the weights of other motors only (excluding the arm weights), each motor is approximated as a cube and has the moment of inertia of the cube.

Therefore, the rotational inertia for the motor 1 is given by,

$$I_1 = \mathrm{M}\left(\frac{1}{3}s^2 + L^2 + 2\left(Lcos\frac{\theta_2}{2}\right)^2\right)$$

, where I_1 the moment inertia lifted by motor 1, M the mass of each motor, s the length of squared motor and L the arm length.

By considering the geometry, the minimum torque acting by the weight depends on the joint angles,

$$\tau_{1,min} = MgL\left(sin\theta_1 + 2sin\left(\theta_1 + \frac{\theta_2}{2}\right)cos\frac{\theta_2}{2}\right)$$

Similarly, for the motor 2, the rotation inertia and the minimum torque are given by,

$$I_2 = M\left(\frac{1}{3}s^2 + 2L^2\right)$$

$$\tau_{2,min} = MgL(sin\theta_1 + sin\theta_3)$$

For the motor 3, the rotation inertia and the minimum torque are given by,

$$I_3 = M\left(\frac{1}{3}s^2 + L^2 + 2\left(Lcos\frac{\theta_2}{2}\right)^2\right)$$

$$\tau_{3,min} = MgL\left(sin\theta_3 + 2sin\left(\theta_3 + \frac{\theta_2}{2}\right)cos\frac{\theta_2}{2}\right)$$

The angular acceleration of the motor i is given by,

$$\alpha_i = \frac{\tau_i - \tau_{i,min}}{I_i}$$

, where τ_i the output torque of the motor.

Whether the motor can drive the weights to the instructed angle and execute the action depends on its previous pose. As it will affect the angular acceleration, if $\alpha = 0$, the motor cannot lift at that angle but can only stay at that angle. This explains why sometimes the motors cannot execute certain actions under certain poses.

Especially when the motherboard is attached on the 3rd arm, the motor 1 cannot lift the rear foot with any pose because the motherboard is far away from motor 1. Even if the motherboard is attached on the 2nd arm which is closer to the motor 1, but now it is far away from motor 3 so the robot cannot lift its front foot. So, a prepared pose is needed to lift the rear foot as expected.

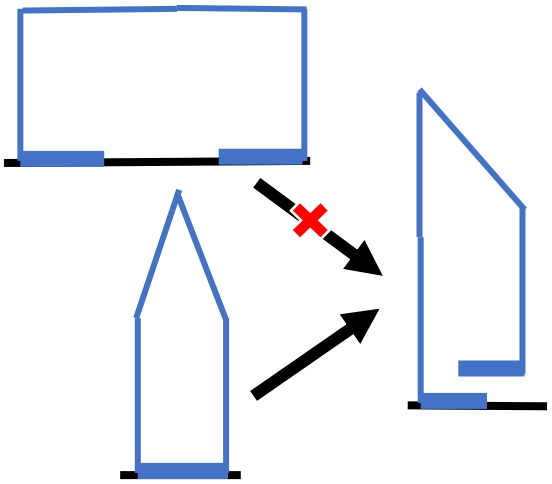


Figure 2: It shows how the robot needs a prepared pose to execute another move.

Since the feet are not modified, there is another problem caused by the screws which make the feet not flat and induce motion instability.

Moreover, the robot is constructed with non-identical arms which are not in the same length. Thus, the inverse kinematics equations with equal arm length assumption cannot estimate the accurate joint angles.

The wires connected to the motherboard on the robot provide extra inertia to the robot. If the wires are from the travelling direction of the robot, the robot can move forwards more easily. But if the wires are opposite the travelling direction, they pull the robot backwards, the robot moves forwards more slowly.

Due to the influence of weight, the non-identical arm lengths, the wires and the non-flat feet, the robot was found to oscillate a lot during the execution.

To overcome these problems, we have developed a Graphical User Interface with matlab to fine tune the executed angles calculated by the inverse kinematics. And we have tried to strategize the motion with higher tolerance of uncertainties.

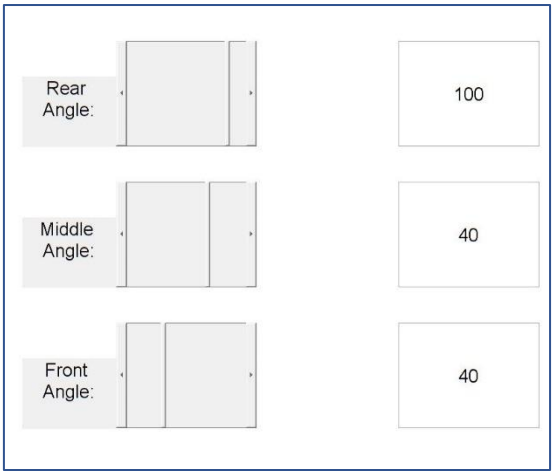


Figure 3: Graphical User Interface (GUI) which can control the joint angles of the robot.

For the locomotion Problem 1, our strategy is that the robot first walks towards the stair, when it is in front of the stair, it performs a backflip motion. The arm then extends to pass the goal.

This set of motions has the advantage of high tolerance of motion instability. It does not require precise and accurate control of angles to walk on the stair one by one to reach the goal. The backflip motion is also quicker because it uses the gravity to pull itself down to the upper level of staircase.

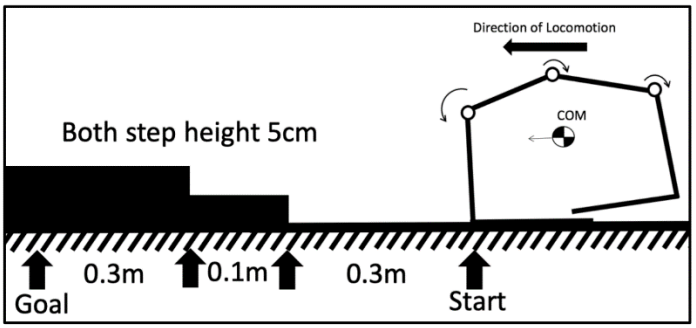


Figure 4: Locomotion Problem 1.

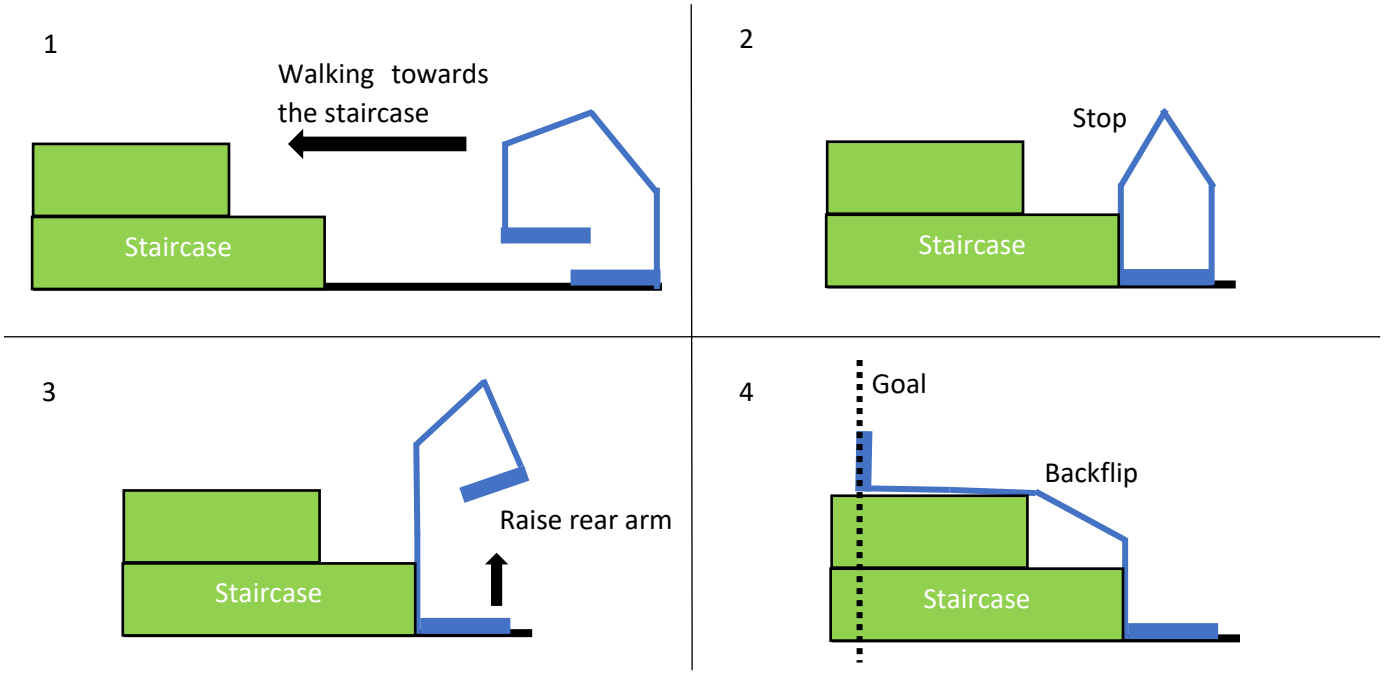


Figure 5: Strategy for locomotion problem 1 applied to reach the goal.

For the locomotion Problem 2, our strategy is that the robot first walks towards the obstacle. Then it uses its front arm to grab the obstacle and pulls itself up to the obstacle. The rear foot kicks the obstacle and the robot crosses the obstacle. Finally, it walks towards the goal to finish the task.

In this problem, one of the biggest challenges is to walk over the obstacle without losing balance. But since the robot is self-oscillating a lot, it may deviate from the obstacle direction and does not allow it to climb up and down the obstacle smoothly and accurately to avoid losing balance. Thus, our strategy to grab the obstacle and pull can help to instantly repositioning the robot walking direction. And since the robot feet are long, the climb up and down motion are difficult. Therefore, the strategy to get over it is to use the front foot to hit the ground which provides a upward force that lift up the rear foot on the obstacle. This motion is repeated twice to move the rear foot further on the obstacle. Finally, the rear foot kicks backward to force the robot to go down the obstacle.

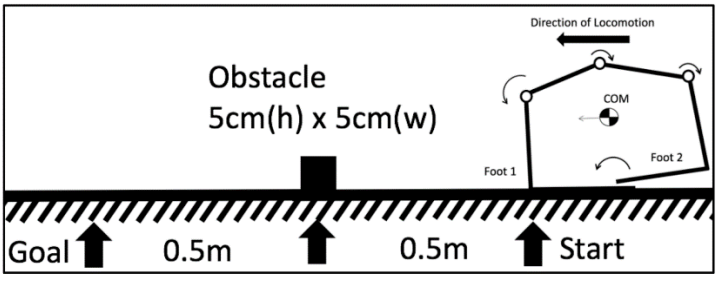


Figure 6: Locomotion Problem 2.

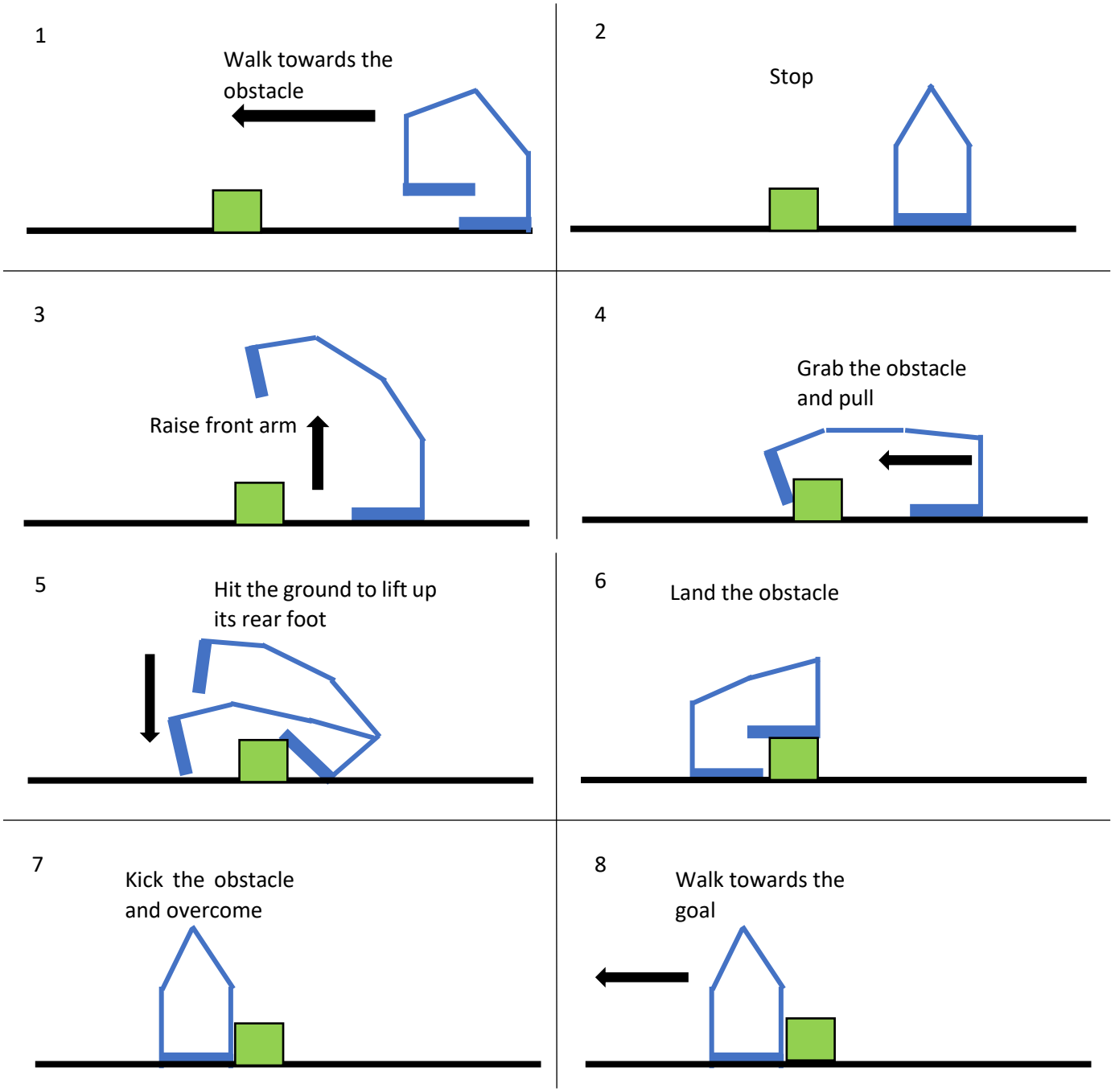


Figure 7: Strategy for Locomotion Problem 2 applied to reach the goal.

Question 2 (run s1.m and s2.m)

After watching the demonstration of group 8, I believe their technique in climbing over the stair (locomotion problem 1) can also be used to climb up and down the obstacle (locomotion problem 2) with only kinematic control if the robot has a good balance.

I choose to change my strategy that replicates the trajectory of the group 8 in locomotion problem 1 since it can be simulated directly by the kinematic method to avoid the complex dynamic control. Their arm length is $0.14m$ and foot length is $0.1m$ (Not $0.2m$).

I simulate their strategy used in locomotion problem 1 to solve locomotion problem 2. But since they were also using the dynamic method to overcome the obstacle, then I will only compare the walking motion data from them in locomotion problem 2.

In my simulation, the joint angles' trajectories for 1st locomotion problem are,

$$5\left[\begin{pmatrix}34.8^\circ\\110.3^\circ\\34.8^\circ\end{pmatrix}\rightarrow\begin{pmatrix}8.8^\circ\\120.3^\circ\\50.9^\circ\end{pmatrix}\rightarrow\begin{pmatrix}20.9^\circ\\138.2^\circ\\20.9^\circ\end{pmatrix}\rightarrow\begin{pmatrix}50.9^\circ\\120.3^\circ\\8.8^\circ\end{pmatrix}\right]\rightarrow\begin{pmatrix}34.8^\circ\\110.3^\circ\\34.8^\circ\end{pmatrix}$$

The set of joint angles allow the robot to walk from the starting location to the front of the stair.

In comparison to the 1st walking data from group 8,

$$(15|true:10)\left[\begin{pmatrix}30^\circ\\95^\circ\\30^\circ\end{pmatrix}\rightarrow^{\begin{pmatrix}10^\circ\\95^\circ\\30^\circ\end{pmatrix}}\begin{pmatrix}10^\circ\\100^\circ\\30^\circ\end{pmatrix}\rightarrow\begin{pmatrix}20^\circ\\110^\circ\\20^\circ\end{pmatrix}\rightarrow\begin{pmatrix}30^\circ\\100^\circ\\10^\circ\end{pmatrix}\rightarrow^{\begin{pmatrix}30^\circ\\95^\circ\\10^\circ\end{pmatrix}}\right]\rightarrow\begin{pmatrix}30^\circ\\95^\circ\\30^\circ\end{pmatrix}$$

The predicted data is quite similar. But it takes more steps in reality (=10) than in simulation (=5). The predicted angle is generally greater than that in the reality since group 8 takes a smaller step to keep the balance, but my simulation does not have the gravity involved so the robot in the simulation is taking a bigger step.

There are some adjustment angles presented in the superscript of the arrow, they are used to stabilize the motion.

When climbing over stairs, the predicted trajectory is,

$$\left[\begin{pmatrix}34.8^\circ\\110.3^\circ\\34.8^\circ\end{pmatrix}\rightarrow\begin{pmatrix}8.8^\circ\\120.3^\circ\\50.9^\circ\end{pmatrix}\rightarrow\begin{pmatrix}20.9^\circ\\138.2^\circ\\20.9^\circ\end{pmatrix}\right]\rightarrow 2\left[\begin{pmatrix}50.1^\circ\\132.9^\circ\\-3.0\end{pmatrix}\rightarrow\begin{pmatrix}61.5^\circ\\85.2^\circ\\33.4^\circ\end{pmatrix}\rightarrow\begin{pmatrix}45.6^\circ\\88.8^\circ\\45.6^\circ\end{pmatrix}\rightarrow\begin{pmatrix}20.9^\circ\\138.2^\circ\\20.9^\circ\end{pmatrix}\right]$$

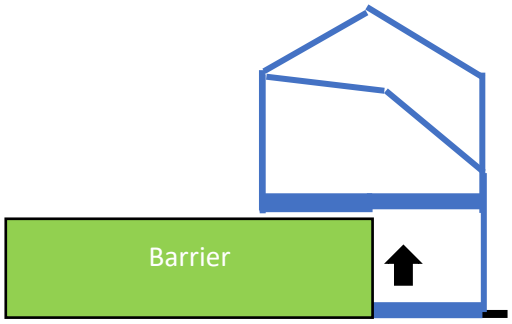
The angle in the reality is,

$$\left[\begin{pmatrix}30^\circ\\95^\circ\\30^\circ\end{pmatrix}\rightarrow^{\begin{pmatrix}10^\circ\\95^\circ\\30^\circ\end{pmatrix}}\begin{pmatrix}10^\circ\\100^\circ\\30^\circ\end{pmatrix}\rightarrow\begin{pmatrix}20^\circ\\110^\circ\\20^\circ\end{pmatrix}\right]$$

$$\rightarrow 2\left[\begin{pmatrix}40^\circ\\110^\circ\\-10^\circ\end{pmatrix}\rightarrow\begin{pmatrix}60^\circ\\75^\circ\\-10^\circ\end{pmatrix}\begin{pmatrix}60^\circ\\60^\circ\\40^\circ\end{pmatrix}\begin{pmatrix}55^\circ\\55^\circ\\30^\circ\end{pmatrix}\begin{pmatrix}50^\circ\\65^\circ\\20^\circ\end{pmatrix}\begin{pmatrix}50^\circ\\73^\circ\\10^\circ\end{pmatrix}\begin{pmatrix}35^\circ\\85^\circ\\-5^\circ\end{pmatrix}\begin{pmatrix}35^\circ\\95^\circ\\-20^\circ\end{pmatrix}\begin{pmatrix}35^\circ\\100^\circ\\-30^\circ\end{pmatrix}\begin{pmatrix}30^\circ\\110^\circ\\-45^\circ\end{pmatrix}\begin{pmatrix}30^\circ\\120^\circ\\-45^\circ\end{pmatrix}\begin{pmatrix}20^\circ\\120^\circ\\-35^\circ\end{pmatrix}\begin{pmatrix}10^\circ\\120^\circ\\-20^\circ\end{pmatrix}\begin{pmatrix}-10^\circ\\120^\circ\\0^\circ\end{pmatrix}\begin{pmatrix}-10^\circ\\110^\circ\\20^\circ\end{pmatrix}\rightarrow\begin{pmatrix}20.9^\circ\\138.2^\circ\\20.9^\circ\end{pmatrix}\right]$$

Group 8 climbs up the stair successfully due to many adjustments angles to keep the balance. In Figure 8, it shows how differently the robot climbs up the stair in simulation and in reality.

In simulation,



In reality,

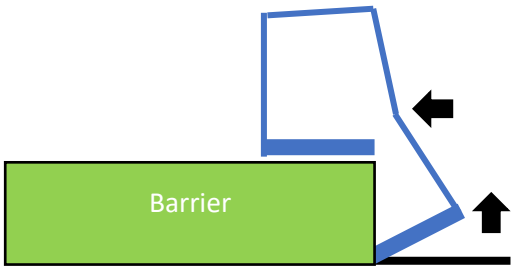


Figure 8: Comparison of simulation and reality performance of climbing the stair

Lastly, walking towards the goal, the predicted angles are,

$$5\left[\begin{pmatrix}20.9^\circ\\138.2^\circ\\20.9^\circ\end{pmatrix}\rightarrow\begin{pmatrix}50.5^\circ\\122.5^\circ\\6.9^\circ\end{pmatrix}\rightarrow\begin{pmatrix}32.4^\circ\\115.2^\circ\\32.4^\circ\end{pmatrix}\rightarrow\begin{pmatrix}6.9^\circ\\122.5^\circ\\50.5^\circ\end{pmatrix}\right]\rightarrow\left[\begin{pmatrix}20.9^\circ\\138.2^\circ\\20.9^\circ\end{pmatrix}\rightarrow\begin{pmatrix}50.5^\circ\\122.5^\circ\\6.9^\circ\end{pmatrix}\rightarrow\begin{pmatrix}32.4^\circ\\115.2^\circ\\32.4^\circ\end{pmatrix}\right]$$

The walking data gives us,

$$4\left[\begin{pmatrix}20^\circ\\110^\circ\\20^\circ\end{pmatrix}\rightarrow\begin{pmatrix}30^\circ\\100^\circ\\10^\circ\end{pmatrix}\rightarrow\begin{pmatrix}30^\circ\\95^\circ\\10^\circ\end{pmatrix}\rightarrow\begin{pmatrix}30^\circ\\95^\circ\\30^\circ\end{pmatrix}\rightarrow\begin{pmatrix}10^\circ\\95^\circ\\30^\circ\end{pmatrix}\rightarrow\begin{pmatrix}10^\circ\\100^\circ\\30^\circ\end{pmatrix}\rightarrow\begin{pmatrix}20^\circ\\110^\circ\\20^\circ\end{pmatrix}\right]$$

The robot only takes 4 steps in reality but 5 steps in simulation. The simulation is more accurate on the upper level of the staircase this time because there is a friction difference between the floor and the upper level of the staircase.

The predicted joint angles' trajectories for 2nd locomotion problem with the simulation is,

When walking to the front of the obstacle,

$$10\left[\begin{pmatrix}32.4^\circ\\115.2^\circ\\32.4^\circ\end{pmatrix}\rightarrow\begin{pmatrix}6.9^\circ\\122.5^\circ\\50.5^\circ\end{pmatrix}\rightarrow\begin{pmatrix}20.9^\circ\\138.2^\circ\\20.9^\circ\end{pmatrix}\rightarrow\begin{pmatrix}50.5^\circ\\122.5^\circ\\6.9^\circ\end{pmatrix}\right]\rightarrow\begin{pmatrix}32.4^\circ\\115.2^\circ\\32.4^\circ\end{pmatrix}$$

When climbing over the obstacle, (use kinematic method to get over)

$$\left[\begin{pmatrix}6.9^\circ\\122.5^\circ\\50.5^\circ\end{pmatrix}\rightarrow\begin{pmatrix}20.9^\circ\\138.2^\circ\\20.9^\circ\end{pmatrix}\right]\rightarrow\left[\begin{pmatrix}50.1^\circ\\132.9^\circ\\-3.0^\circ\end{pmatrix}\rightarrow\begin{pmatrix}61.5^\circ\\85.2^\circ\\33.4^\circ\end{pmatrix}\rightarrow\begin{pmatrix}45.6^\circ\\88.8^\circ\\45.6^\circ\end{pmatrix}\rightarrow\begin{pmatrix}20.9^\circ\\138.2^\circ\\20.9^\circ\end{pmatrix}\right]$$

When climbing down the obstacle,

$$\left[\begin{pmatrix}32.4^\circ\\115.2^\circ\\32.4^\circ\end{pmatrix}\rightarrow\begin{pmatrix}15{,}9^\circ\\111.2^\circ\\52.8^\circ\end{pmatrix}\rightarrow\begin{pmatrix}-3.0^\circ\\132.9^\circ\\50.1^\circ\end{pmatrix}\rightarrow\begin{pmatrix}20.9^\circ\\138.2^\circ\\20.9^\circ\end{pmatrix}\right]$$

Finally, walking towards the goal,

$$5\left[\begin{pmatrix}20.9^\circ\\138.2^\circ\\20.9^\circ\end{pmatrix}\rightarrow\begin{pmatrix}51.7^\circ\\115.9^\circ\\12.4^\circ\end{pmatrix}\rightarrow\begin{pmatrix}40.0^\circ\\100.0^\circ\\40.0^\circ\end{pmatrix}\rightarrow\begin{pmatrix}12.4^\circ\\115.9^\circ\\51.7^\circ\end{pmatrix}\right]\rightarrow\left[\begin{pmatrix}20.9^\circ\\138.2^\circ\\20.9^\circ\end{pmatrix}\rightarrow\begin{pmatrix}51.7^\circ\\115.9^\circ\\12.4^\circ\end{pmatrix}\rightarrow\begin{pmatrix}40.0^\circ\\100.0^\circ\\40.0^\circ\end{pmatrix}\right]$$

From their data,

For walking,

$$(15|true:10)\left[\begin{pmatrix}30^\circ\\95^\circ\\30^\circ\end{pmatrix}\rightarrow\begin{pmatrix}10^\circ\\95^\circ\\30^\circ\end{pmatrix}\rightarrow\begin{pmatrix}10^\circ\\100^\circ\\30^\circ\end{pmatrix}\rightarrow\begin{pmatrix}20^\circ\\110^\circ\\20^\circ\end{pmatrix}\rightarrow\begin{pmatrix}30^\circ\\100^\circ\\10^\circ\end{pmatrix}\rightarrow\begin{pmatrix}30^\circ\\95^\circ\\10^\circ\end{pmatrix}\right]\rightarrow\begin{pmatrix}30^\circ\\95^\circ\\30^\circ\end{pmatrix}$$

For the obstacle part, (use dynamic method to get over)

$$\begin{pmatrix}20^\circ\\110^\circ\\20^\circ\end{pmatrix}\rightarrow\begin{pmatrix}40^\circ\\110^\circ\\-10^\circ\end{pmatrix}\rightarrow\begin{pmatrix}60^\circ\\60^\circ\\-10^\circ\end{pmatrix}\rightarrow\begin{pmatrix}20^\circ\\60^\circ\\0^\circ\end{pmatrix}\rightarrow\begin{pmatrix}20^\circ\\60^\circ\\70^\circ\end{pmatrix}\rightarrow\begin{pmatrix}40^\circ\\60^\circ\\70^\circ\end{pmatrix}$$

$$\begin{pmatrix}90^\circ\\0^\circ\\90^\circ\end{pmatrix}\rightarrow\begin{pmatrix}70^\circ\\0^\circ\\80^\circ\end{pmatrix}\rightarrow\begin{pmatrix}70^\circ\\40^\circ\\80^\circ\end{pmatrix}\rightarrow\begin{pmatrix}20^\circ\\110^\circ\\20^\circ\end{pmatrix}$$

After the obstacle,

$$10\left[\begin{pmatrix}20^\circ\\110^\circ\\20^\circ\end{pmatrix}\rightarrow\begin{pmatrix}30^\circ\\100^\circ\\10^\circ\end{pmatrix}\rightarrow\begin{pmatrix}30^\circ\\95^\circ\\10^\circ\end{pmatrix}\rightarrow\begin{pmatrix}30^\circ\\95^\circ\\30^\circ\end{pmatrix}\rightarrow\begin{pmatrix}10^\circ\\95^\circ\\30^\circ\end{pmatrix}\rightarrow\begin{pmatrix}10^\circ\\100^\circ\\30^\circ\end{pmatrix}\rightarrow\begin{pmatrix}20^\circ\\110^\circ\\20^\circ\end{pmatrix}\right]$$

The inaccuracy is from not considering dynamic control. For example, during the demonstration, the robot usually vibrates whatever execution is made, this vibration induces small distance error that cannot be accounted by the kinematic simulation method. The friction and normal reaction of the mat, the obstacle and the staircase are also neglected during the simulation, which also cause the inaccuracy.

Also, the weight is not considered. In the demonstration, it is found that the robot cannot raise the arm attached with the motherboard.

Even, if we have simulated the friction, there is still a lot of uncertainties. Since the motion of the robot changes the environment settings, for instance, the fur direction of the mat, it can then change the friction constant, which causes inaccuracy again.

Question 3 (run s1.m, s2.m and s3.m)

Assume that the obstacle and the stair step have no friction, to overcome the terrain and reach the goal, the strategy of the planning algorithm can be set up as follows.

The planning algorithm involves two parts,

1. Walking
   The number of steps and step size needed for the robot to travel from the starting point to the obstacle or the staircase are firstly computed. The first part of this algorithm creates some possible step sizes from 0 to maximum travel distance physically achievable by the front foot with an increment of -0.01,

```
function [idealsize, idealno]= no_steps(start_x,end_x,max_step)
% All possible step distance
step_size = [];
step_no = [];
d = end_x-start_x;
count = 1;
for i = 0:-0.01:max_step
    step_size(count) = i;
    step_no(count) = d/i;
    count = count+1;
end
% Choose the step size with min no.
dim = size(step_no);
step_no_integer = [];
count2 = 1;
for j = 1:dim(2)
    if rem(step_no(j),1) == 0
        step_no_integer(count2) = step_no(j);
        count2 = count2+1;
    end
end
idealno = min(abs(step_no_integer));
ind = find(step_no==idealno);
idealsize = step_size(ind);

end
```

   In all the possible step sizes, choose the optimal step size which can satisfy the condition that the number of steps for this trajectory are minimized.
   To avoid the repetitive calculation of step size needed, the step size is reused after the robot climbs over the obstacle or the staircase.

   To generate the full joint angles trajectory, all the parameters calculated above are put into the function that can transform the step size and the number of steps into front (joint 0) and rear (joint 4) feet step locations, the moving foot index and three joint angles. This information is then stored into the matrix.
   So, the following matrix is in the format of,

Table 1 shows how the information matrix about the robot locomotion is formatted.

1	Front foot (joint 0) x-location
2	Rear foot (joint 4) x-location
3	Front foot (joint 0) y-location
4	Rear foot (joint 4) y-location
5	Moving foot index (indicates which foot is moving)
6	Joint angle 1
7	Joint angle 2
8	Joint angle 3

8x54 double

	1	2	3	4	5	6
1	0	0	0	-0.0300	-0.0600	-0.0600
2	0.1600	0.1300	0.1000	0.1000	0.1000	0.0700
3	0	0	0	0.0500	0	0
4	0	0.0500	0	0	0	0.0500
5	114	114	114	102	102	114
6	0.6082	0.1535	0.3652	0.8878	0.6082	0.1535
7	1.9251	2.1003	2.4112	2.1003	1.9251	2.1003
8	0.6082	0.8878	0.3652	0.1535	0.6082	0.8878
9						

2. Climbing

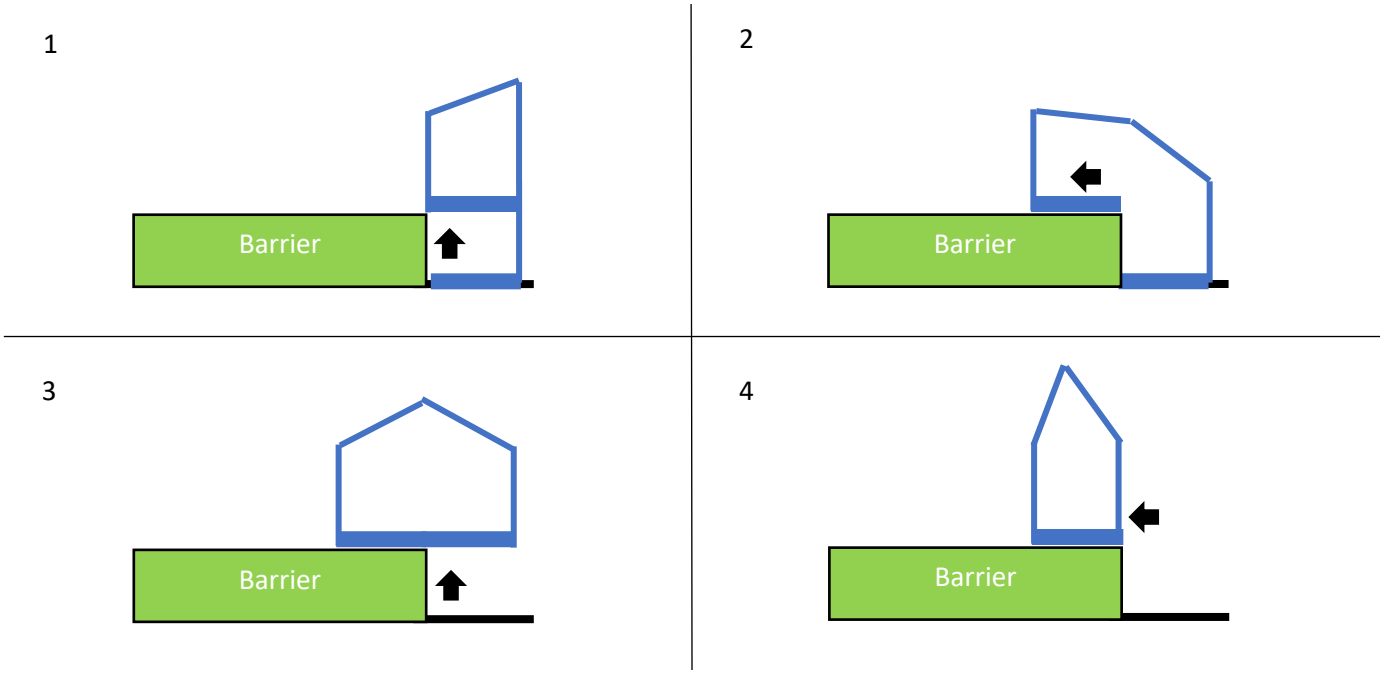


Figure 9: Strategy applied to climb any obstacle.

Once the robot has reached the obstacle or the staircase, the climbing part of the algorithm is activated. As shown in Figure 9, the foot which is close to the barrier is raised vertically upwards to the height of the barrier, and then moving forwards horizontally until the whole foot is fit on the barrier. Once the first foot has landed properly, the next foot is also raised vertically upwards to the height of the barrier, and then moving forwards horizontally until the whole robot is standing upon the barrier.

```
% Steps
movecoor2 = zeros(4,8);
feet2 = [];
for s = 1:8
    if s == 1
        movecoor2(:,s) = movecoor(:,dim(2));
    else
        movecoor2(:,s) = movecoor2(:,s-1);
    end
    if rem(s,2) == 0

        if rem(s,4) == 0
            feet2(s-1) = 'r';
            feet2(s)   = 'r';
            movecoor2(2,s) = movecoor2(2,s)-0.1; % 0.1 stair size
            movecoor2(3,s-3:s) = height*(s/4);
            movecoor2(4,s-1:s) = height*(s/4);
        else
            feet2(s-1) = 'f';
            feet2(s)   = 'f';
            movecoor2(1,s) = movecoor2(1,s)-0.1; % 0.1 stair size

        end
    end
end
movecoor2 = [movecoor2;feet2];
```

With this climbing part, it allows the robot to overcome any barrier.

This planning algorithm can successfully overcome the locomotion problems 1 and 2. To test its planning ability, the following terrain is set up. The new terrain consists of a 5 steps staircase. (run s3.m)

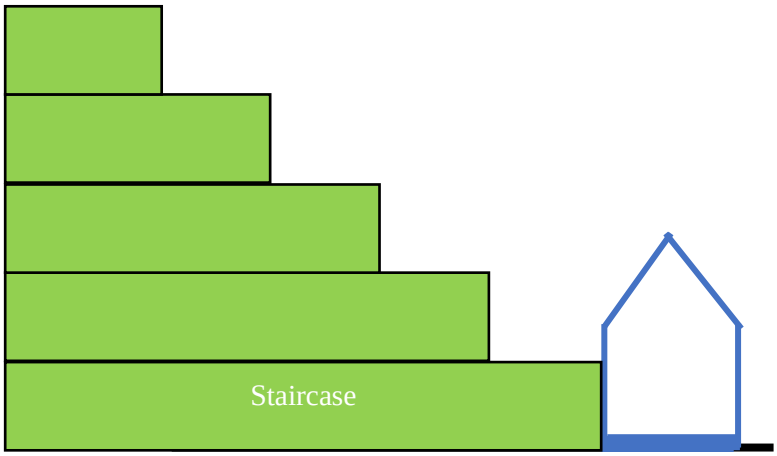


Figure 10: The terrain architecture and the robot initial position.

The planning algorithm should output the following joint angle trajectory automatically from the command window,

$$\begin{pmatrix} 20.9^{\circ} \\ 138.2^{\circ} \\ 20.9^{\circ} \end{pmatrix} \rightarrow 5\left[\begin{pmatrix} 50.1^{\circ} \\ 132.9^{\circ} \\ -3.0 \end{pmatrix} \rightarrow \begin{pmatrix} 61.5^{\circ} \\ 85.2^{\circ} \\ 33.4^{\circ} \end{pmatrix} \rightarrow \begin{pmatrix} 45.6^{\circ} \\ 88.8^{\circ} \\ 45.6^{\circ} \end{pmatrix} \rightarrow \begin{pmatrix} 20.9^{\circ} \\ 138.2^{\circ} \\ 20.9^{\circ} \end{pmatrix}\right]$$

Question 4 (run Q.m)

A breadth-first search (BFS) Q-learning algorithm is developed in this question to allow the simulated robot to learn locomotion of its foot (actually only the landing spots of the feet). But once the robot knows the landing spots, it is not difficult to recover the whole continuous moving tajectory of the joints locations and angles.

So the algorithm is described as follows.

1. Setting up possible states for both feet (1st,2nd and 3rd column of the matrix)

```
function [form,indices] = matrixform(xstate,ystate,frontx,rearx,reffx,refrx)
dimx= size(xstate); dimy = size(ystate);
form = zeros(dimx(2)*dimy(2),2);
for i = 1:dimx(2)*dimy(2)
    if rem(i,5) == 0
        form(i-4:i,1) = xstate(i/5);
        form(i-4:i,2) = ystate';
    end
end
form = [form;form];
leg = zeros(2*dimx(2)*dimy(2),1);
leg(1:dimx(2)*dimy(2)) = 'r'; leg(dimx(2)*dimy(2)+1:2*dimx(2)*dimy(2)) = 'f';
form = [form, leg];
form = reward_pos(form,reffx,refrx);
form = reward_loc(form,frontx,rearx,reffx,refrx);
form = sum_reward(form);
% select the greatest reward
indices = find(form(:,7)==max(form(:,7)));
form(indices,:)
end
```

To make the task computationally possible, I have set up a finite number of choices for each dimension for each foot to choose from. The number of choices is also large enough to allow the robot to overcome any obstacle in a trial-and-error manner.

For the x-axis, the possible states range from -0.18 to 0.18m with a step of +0.01m so the number of x-states is 37. For the y-axis, the possible states range from 0 to 0.1m with a step of +0.025m so the number of y-states is 5. And for each x state value, 5 possible y states are available. So, a total number of states available for each leg is 37x5 = 185.

As the result, in Figures 11 and 12, each circle represents each possible state in the 2d space. The red circles denote the front feet states, and the blue ones denote the rear feet states.

To give both feet to have equal opportunities to move, all possible states for both feet are put into the 1st and 2nd columns of the same matrix. And to distinguish whether the possible states belong to front or rear foot, the 3rd column is encoded with the feet information using ASCII standard. Ideally, whether the robot moves its front or rear feet in that decision state depends on the final reward system.

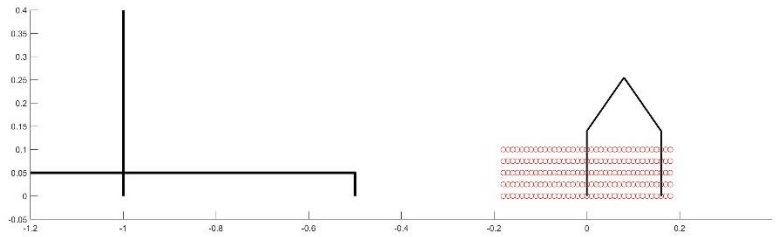


Figure 11: All possible states available for the front foot.

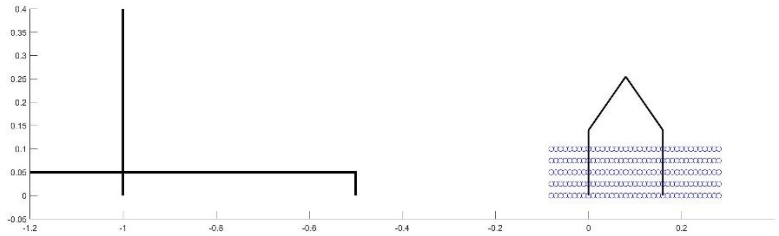


Figure 12: All possible states available for the rear foot.

2. Reward and Penalty for the robot pose (4th and 5th column of the matrix)

```
function reward = reward_pos(form,reffx,refrx)
dim = size(form);
reward = zeros(dim(1),1);
possible = zeros(dim(1),1);
for i = 1:dim(1)
    reward(i,1) = -10*form(i,1);

    if form(i,3) == 114
        ext = sqrt((reffx-(form(i,1)+refrx))^2+(0-(form(i,2)+0))^2);
        if or(ext > 0.28, reffx>(form(i,1)+refrx))
            possible(i,1) = 0;
            reward(i,1) = -10000;
        else
            possible(i,1) = 1;
        end
    elseif form(i,3) == 102
        ext = sqrt((form(i,1)+reffx-refrx)^2+(form(i,2)+0-(0))^2);
        if or(ext > 0.28, (reffx+form(i,1))>refrx)
            possible(i,1) = 0;
            reward(i,1) = -10000;
        else
            possible(i,1) = 1;
        end
    end
end
reward = [form,reward,possible];
end
```

It gives the rewards when the estimated pose in each possible state is physically acheivable by the robot, but gives the penalties when it does the opposite.

i. Maximum separation

Assume that the sum of the joint angles is 180° which allows the feet land perpedicularly on the ground, the maximum separation between the joints 0 and 4 is $0.28m$ which is the sum of L_2 and L_3.

The separation between the rear foot (joint 4) and the front foot (joint 0) is given by,

$$s = \sqrt{\left(x_{front} - x_{rear}\right)^2 + \left(y_{front} - y_{rear}\right)^2}$$

, where s the joints 0 and 4 separation, x_{front} the joint 0's x position, x_{rear} the joint 4's x position, y_{front} the joint 0's y position and y_{rear} the joint 4's position.

If $s > 0.28$, the following state will be penalized with the value of -10000.

ii. Rear foot cannot surpass Front foot

The condition $x_{front} \leq x_{rear}$ is set. If $x_{front} > x_{rear}$, the following state will be penalized with the value of -10000.

iii. Reward for the pose approaching the goal direction

Since the terrain and the goal are set on the negative side of the x-axis, the reward is set to be 10 times the opposite sign of the x state position. For example, if the front foot chooses to move towards $-0.18m$ relative to its initial landing spot, the goal is positioned at $-1m$, the reward will be 1.8.

The reward equation is given by,

$$R = -10(x_{state})$$

, where R the reward and x_{state} the possible states along x-axis ($linspace(-0.18,0.18,37)$).

3. Reward and Penalty for the robot location (6th column of the matrix)

```
function reward = reward_loc(form,frontx,rearx,reffx,refrx)
% Add platform info later
x_plt = -0.5; y_plt = 0.05;
%----------------------%
dim = size(form);
loc = zeros(dim(1),1);
for i = 1:dim(1)
    fx = frontx;
    rx = rearx;
    if form(i,3) == 114
        loc(i,1) = -1000*(form(i,1)+refrx-rx);
        if form(i,1)+refrx < x_plt
            if form(i,2)+0-0 < y_plt
                loc(i,1) = -10000;
            end
        end
    elseif form(i,3) == 102
        loc(i,1) = -1000*(form(i,1)+reffx-fx);
        if form(i,1)+reffx < x_plt
            if form(i,2)+0-0 < y_plt
                loc(i,1) = -10000;
            end
        end
    end
end
reward = [form,loc];
end
```

It gives the rewards when the robot is walking towards the goal direction and landing its feet on the platform, but gives the penalties when it does the opposite.

i. Relative to the original rear and front feet x location

Giving reward due to the motion towards the goal relative to the starting position allows the robot to figure out which direction it should go. And it is more important than the pose it choose to move, so the following reward is set,

$$R = -1000\left(x_i - x_{start,i}\right)$$

, where R the reward, x_i the current location of the foot i which is the sum of x_{state} and $x_{ref,i}$ the latest reference location of the foot i and $x_{start,i}$ the starting x location of the foot i.

It is important to note that the distance travelled along y-axis is not considered since our goal is to reach $x = -1m$. Later, the obstacle penalty will be introduced so that the robot will learn to lift the feet to the terrain.

ii. Relative to the stairs

Once one of the feet is stepped into the region where the terrain is positioned, the stair penalty is activated when the foot is below a certain height.

The robot is penalized by the value of -10000 when the landing spot of the foot is below the height of the terrain.

When the penalty is not set, the front leg chooses to land below the terrain as shown in Figure 13.

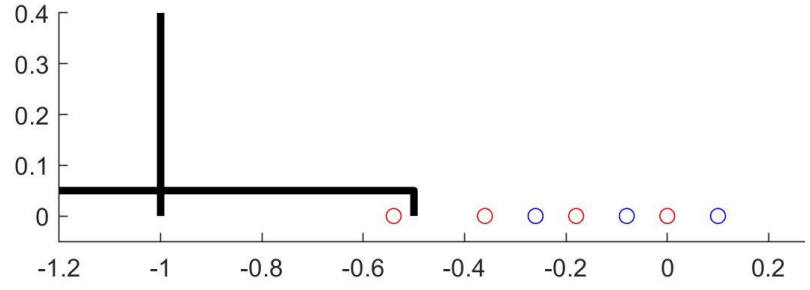


Figure 13: Wrong landing point chosen by the robot without stair penalty.

When the penalty is set, the front foot can land on the terrain as shown in Figure 14. Also, due to the pose penalty, the robot chooses to retreat its front foot to make the landing pose on the terrain physically achievable.

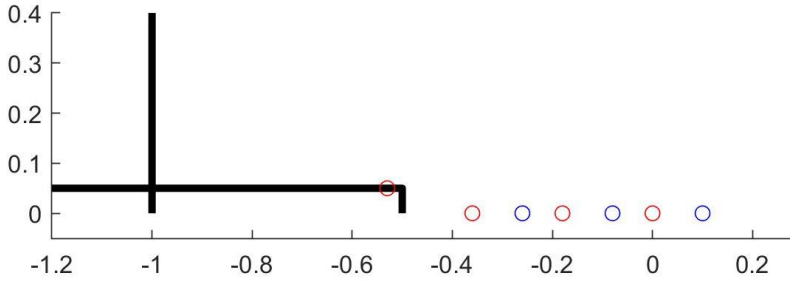


Figure 14: Sensible landing point chosen by the robot with stair penalty.

4. Sum all the rewards and penalties for each state (7th column of the matrix)
   The reward values from the 4th and 6th columns from the given state are summed up to compute the overall reward for that given state.
   So now for each descision state, the robot has a decision matrix shown in Table 2 that describes all possible step states it can take.

```
function sumr = sum_reward(form)
dim = size(form);
su = zeros(dim(1),1);
for i = 1:dim(1)
    if form(i,5) == 1
        su(i,1) = form(i,4)+form(i,6);
    elseif form(i,5) == 0
        su(i,1) = -10000;
    end
end
sumr = [form,su];
end
```

370x7 double

	1	2	3	4	5	6	7
182	0.1800	0.0250	114	-10000	0	-180.0000	-10000
183	0.1800	0.0500	114	-10000	0	-180.0000	-10000
184	0.1800	0.0750	114	-10000	0	-180.0000	-10000
185	0.1800	0.1000	114	-10000	0	-180.0000	-10000
186	-0.1800	0	102	1.8000	1	180	181.8000
187	-0.1800	0.0250	102	-10000	0	180	-10000
188	-0.1800	0.0500	102	-10000	0	180	-10000
189	-0.1800	0.0750	102	-10000	0	180	-10000

Table 2 shows how the matrix stores the information about all possible states and their rewards.

1	2	3	4	5	6	7
All possible x-states	All possible y-states	Rear leg 'r' denoted by ASCII (114)	Reward from the pose	Physicsally possible due to the pose? (Boolean)	Reward from the location	Sum of all reward (pose+location)
All possible x-states	All possible y-states	Front leg 'f' denoted by ASCII (102)	Reward from the pose	Physically possible due to the pose? (Boolean)	Reward from the location	Sum of all reward (pose+location)

The matrix above is created to store all the useful information (landing spots, reward) about the possible states for each decision state made by the robot.

5. Decision making based on the total reward
   The row indices with the maximum overall reward are obtained.

```
% select the greatest reward
indices = find(form(:,7)==max(form(:,7)));
form(indices,:)
end
```

Sometimes there are more than one states with the same x state having the maximum reward because the reward system is based on the x-axis trajectory. But since this Q-learning algorithm is used to predict the landing location of the feet only, the state with maximum reward and minimum landing height (which also concerns the environment settings) is chosen as the final decision.

```
%% Functions
% Summary for choices
function [reffx, refrx] = summary(state,indices,frontx,rearx)
choices = state(indices,:);
index_miny = find(choices(:,2) == min(choices(:,2)));
reffx = frontx; refrx = rearx;
if choices(index_miny, 3) == 114
    refrx = refrx + choices(index_miny,1);
elseif choices(index_miny,3) == 102
    reffx = reffx + choices(index_miny,1);
end

end
```

Result

Overall Trajectory of landing points is shown in Figure 15.

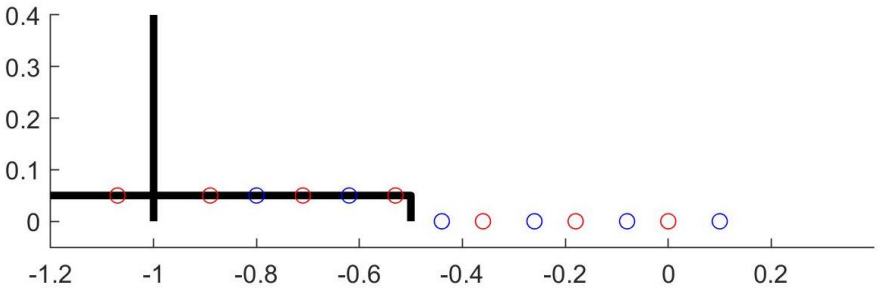


Figure 15: Overall feet landing spot trajectory. Red circles represent the front foot and blue circles represent the rear foot.

The number of trial-and-errors is given by,

$$N = N_{state} N_{step}$$

, where N the number of trial-and-errors, N_{state} the number of all possible states and N_{step} the number of steps made.

So, the number of trial-and-errors is $370(11) = 4070$.