1 Nomenclature

Table 1: List of Mathematical Symbols

Symbol	Description
C^i	Cartesian coordinates at time step i, $C^i = (x^t, y^i, z^i)$, C^{goal} represents goal coordinates
R	Cumulative reward, $R = \sum r$
r^t	Reward at time step t, $r^0 = 0$
D^i	Distance to goal at time step i, $D^i = \|C^{goal} - C^i\|$
D^i_n	Normalised distance to goal at time step i, $D^i_n = \|C^{goal} - C^i\| \div \|C^{goal} - C^0\|$

Table 2: List of Shortforms

Shortform	Full
lf	Left front
lh	Left hind
rf	Right front
rh	Right hind
haa	Hip Abduction Adduction
hfe	Hip Flexion Extension
kfe	Knee Flexion Extension

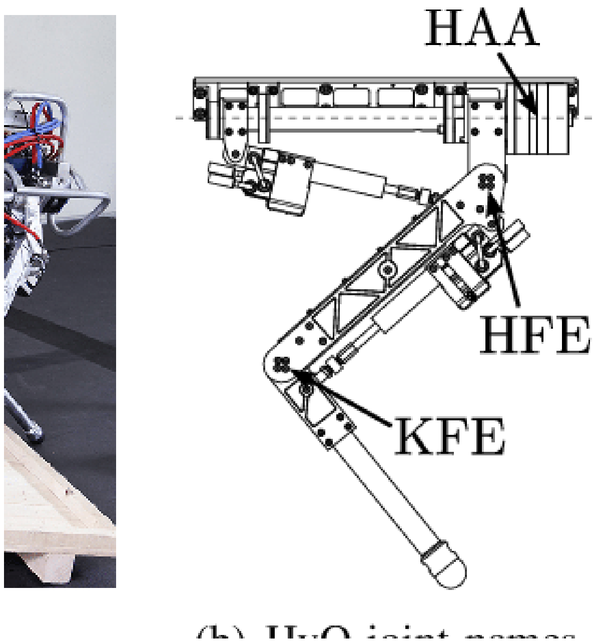

Figure 1: Joint Naming

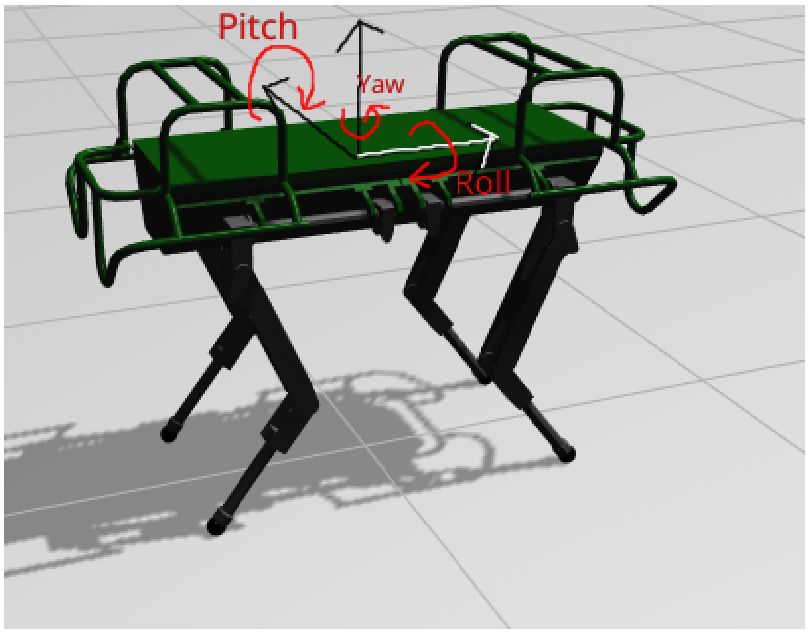

Figure 2: HyQ Rotation

2 User Variables

This section describes the environment variables that can be set by the user.

Symbol	Variable Name	Default	Description
k_{base}	base_rew_scaling	No impl	Base reward scaling constant
k_{exp}	exp_rew_scaling	5.0	Exponential reward scaling constant
k_{norm}	norm_rew_scaling	No impl	Normalised reward scaling constant
r_{reach}	reach_target_bonus_reward	0.0	Bonus reward upon reaching target
ϵ_{reach}	accepted_error	0.001	Accepted error limits for the goal (metres). See section 6.1
ϵ_{joint}	accepted_dist_to_bounds	0.01	Allowable distance to joint limits (radians). See section 6.2

3 Observations

Num	Observation	Min	Max
0	lf_haa_angle	-1.22173	+0.436332
1	lf_hfe_angle	-0.872665	+1.22173
2	lf_kfe_angle	-2.44346	-0.349066
3	lh_haa_angle	-1.22173	+0.436332
4	lh_hfe_angle	-1.22173	+0.872665
5	lh_kfe_angle	+0.349066	+2.44346
6	lf_haa_angle	-1.22173	+0.436332
7	rf_hfe_angle	-0.872665	+1.22173
8	rf_kfe_angle	-2.44346	-0.349066
9	rh_haa_angle	-1.22173	+0.436332
10	rh_hfe_angle	-1.22173	+0.872665
11	rh_kfe_angle	+0.349066	+2.44346
12	lf_haa_speed	-12	+12
13	lf_hfe_speed	-12	+12
14	lf_kfe_speed	-12	+12
15	lh_haa_speed	-12	+12
16	lh_hfe_speed	-12	+12
17	lh_kfe_speed	-12	+12
18	rf_haa_speed	-12	+12
19	rf_hfe_speed	-12	+12
20	rf_kfe_speed	-12	+12
21	rh_haa_speed	-12	+12
22	rh_hfe_speed	-12	+12
23	rh_kfe_speed	-12	+12
24	lf_contact_flag	0	1
25	lh_contact_flag	0	1
26	rf_contact_flag	0	1
27	rh_contact_flag	0	1
28	trunk_contact_flag	0	1
29	trunk_rotation_w	-1	1
30	trunk_rotation_x	-1	1
31	trunk_rotation_y	-1	1
32	trunk_rotation_z	-1	1
33*	trunk_accel_x	-50	50
34*	trunk_accel_y	-50	50
35*	trunk_accel_z	-50	50
36*	trunk_position_x	-10	10
37*	trunk_position_y	-10	10
38*	trunk_position_z	0	1

*Actual range unknown, only estimated

4 Reward

Reward, r is given by the sum of 4 components, r_b, r_n, r_e and r_{reach} which are base reward, normalised reward, exponential reward, and reward upon reaching respectively.

$$r = r_b + r_n + r_e + r_{reach} \tag{1}$$

4.1 Base Reward, r_b

At time t, Base reward, r_b, is calculated using changes in cartesian distance

$$r_b^t = |C^t - C^{t-1}| \tag{2}$$

4.2 Normalised Reward, r_n

Reward is normalised with the aim of making the robot getting the same reward when reaching different goal coordinates. The normalisation approach is described as follows.

1. Determine the magnitude of the changes between initial and goal coordinates, $K = |C^{goal} - C^0|$
2. Calculate normalised reward by dividing base reward with the magnitude K

Therefore,

$$r_n^t = r_b^t \div K = \frac{|C^t - C^{t-1}|}{|C^{goal} - C^0|} \tag{3}$$

4.2.1 Plots

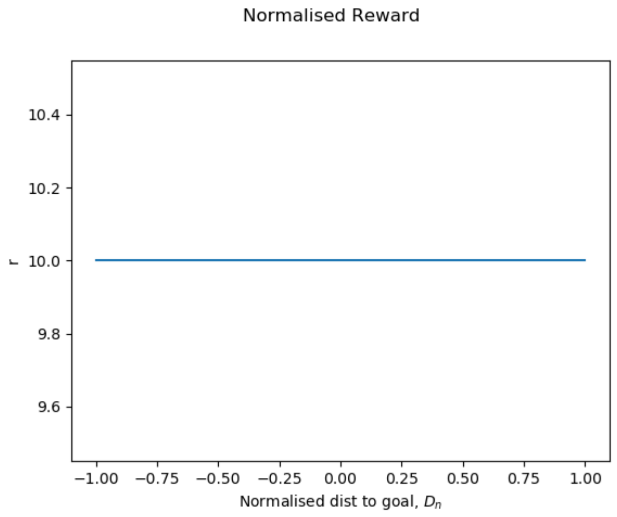

Figure 3: Plot of time step reward, r_n, scale=10

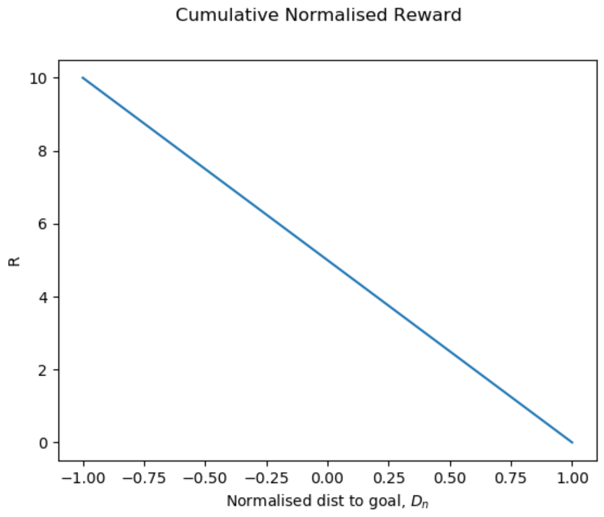

Figure 4: Plot of cumulative reward, R_n, scale=10

4.3 Exponential Reward, r_e

This is to provide additional incentive for the robot to complete the final bit of the task, by providing more reward towards the end. An example of such problem is when the robot is doing only 80% of the task but never 100%, even after prolonged periods of training.

4.3.1 Derivation

Since the x-variable is distance to goal, D_n, it can be a bit inconvenient because we want the maximum value of R_e to be when D_n is close to 0. As such, we transform the x-variable from D_n to some arbitrary variable y, such that $y = 1 - D_n$.

This will produce the results we want, as we get maximum reward when $D_n = 0$, and a reward of 0 when $D_n = 1$, which represents the starting position.

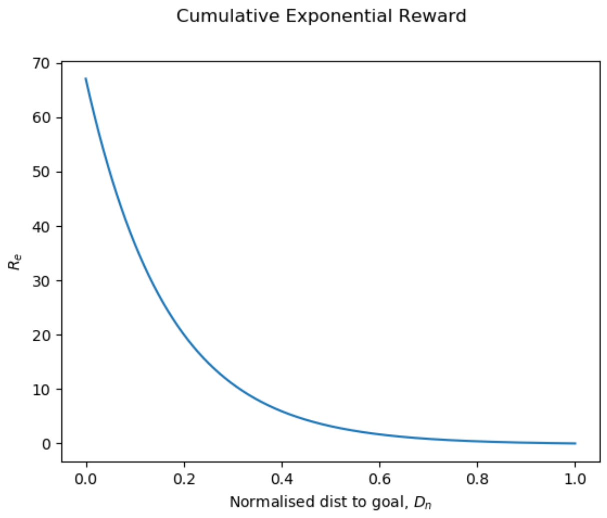

Figure 5: Plot of cumulative exponential reward, R_e, k_{exp}=6, $D_n = [0, 1.0]$

Based on Figure 5, we can see that the if $D_n > 1$, the rewards will only become insignificant and never going negative.

However, we want to penalise the robot for moving away from the goal, which meant that we need to give negative rewards for $D_n > 1$. As such, R_e is made into a piecewise function, with a linear negative function at $D_n > 1$. A further modification is made to the negative region function such that even if the robot keeps moving away from the goal, the reward will not get too negative. This is by introducing another piecewise linear function at $D_n > 9$ with a much lower scaling.

Therefore, our formula for R_e is given by:

$$R_e(D_n) = \begin{cases} \frac{1}{k_{exp}} e^{k_{exp}(1-D_n)} & 0 \leq D_n \leq 1 \\ 5(1 - D_n) & 1 < D_n \leq 9 \\ 0.5(1 - D_n) & 9 < D_n \end{cases} \tag{4}$$

And the formula of r_e is given by:

$$r_e^t = R_e(D_n^t) - R_e(D_n^{t-1}) \tag{5}$$

An illustration of the piecewise function of R_e is shown below.

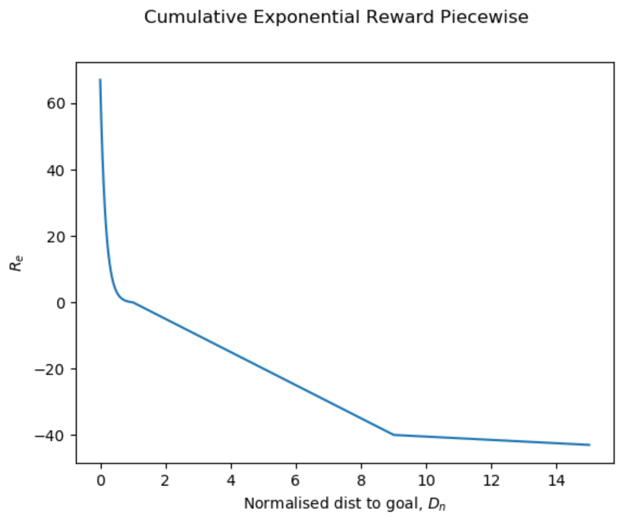

Figure 6: Plot of full cumulative exponential reward, R_e, k_{exp}=6, $D_n = [0, 1.0]$

4.4 Reach Reward, r_{reach}

This is user defined, and is added onto the reward before any penalties are applied. This means that penalty multipliers will be applied to this value.

The reward is provided upon reaching, where the definition of reaching is defined in Section 6.1 below.

5 Penalties

Penalties are handled after all rewards are calculated. Penalty multipliers are applied before fixed penalties are applied.

5.1 Orientation Penalty

Refer to 2 for roll, pitch and yaw definitions.

5.1.1 Yaw

Since the current task is going in a straight line, any orientation that is not in a straight line is penalised.

This is achieved by multiplying the rewards obtained at the current time step by a multiplier between 0 and 1. Since there might be some minor deviations from the yaw angle even during normal walking, we make it such that for some angle range there will be no penalty, and the penalties will only apply after the orientation exceeds the range. This limit is currently set to $\pm 2.5°$ from the desired yaw angle of 0.

After the yaw angle exceeds the limit, the penalty multiplier is $cos(\alpha)$, where α is the yaw angle.

5.1.2 Pitch

Since we are also trying to not allow the robot to tilt forwards or backwards too much, we also give some penalty for exceeding a specified pitch angle limit. The angle limit is currently set to $\pm 2.5°$ from the desired pitch angle of 0.

After the pitch angle exceeds the limit, the penalty multiplier is $cos(\beta)$, where β is the pitch angle.

5.2 Height Penalty

TBD

5.3 Reach Joint Limits

This is user defined, and is subtracted from the reward after the penalty multipliers are applied.

The penalty is given out when any joint reaches its joint limit. The definition of reaching joint limit is described in Section 6.2.

5.4 Fall

This is user defined, and is subtracted from the reward after the penalty multipliers are applied.

The penalty is given out when the robot falls to the ground. The definition of falling is described in Section 6.3.

6 Termination

There are 4 termination conditions, namely

1. Reach goal
2. Reach joint limits
3. Fall (body touching ground)
4. Timeout

6.1 Reach Goal

At the end of each time step, the coordinates of the centre of the robot is checked against the goal coordinates. If the robot's current centre coordinates is within a cube with with side length = ϵ_{reach} and centre located in the goal position, the episode is considered done.

Upon satisfying this condition, the bonus reward for reaching mentioned in Section 4.4 will be added.

6.2 Reach Joint Limits

At the end of each time step the each joint angle is checked against the maximum and minimum of that joint. If the joint angle is within the allowable range of the limits, then the joint limit is considered reached, i.e. ***False*** when $\theta^i_{min} < |\theta^i - \epsilon_{joint}| < \theta^i_{max}$, ***True*** otherwise, where

1. θ^i_{min} is the lower bound of the joint limits of joint i, *radians*
2. θ^i is the current joint angle of joint i, *radians*
3. ϵ_{joint} is the user defined parameter describing the allowable range to the joint limits, *radians*
4. θ^i_{max} is the upper bound of the joint limits of joint i, *radians*

Upon satisfying this condition, the penalty for reaching joint limits mentioned in Section 5.3 will be given out.

6.3 Fall

At the end of each time step, contact between the body and the ground is checked. If during that time step contact was made between the body and the ground, the robot is considered fallen and the episode is considered done.

If 10 simulator steps were made per gym step, then for all 10 simulator steps contact will be checked, not just the final step

Upon satisfying this condition, the penalty for falling mentioned in Section 5.4 will be given out.

6.4 Timeout

If the number of time steps elapsed in the episode is greater than or equal to the max time steps specified by the environment, it is considered done. This means that if t_{max} is set to 1, at the end of the first step it will be considered done and there will be no second step.

True when $t \geq t_{max}$, ***False*** otherwise, where t is current time step, t_{max} is user defined maximum time steps.

No penalty or reward is given for satisfying this condition.