ENGINEERING TRIPOS PART IIB

CCN: 5715D 4M20: ROBOTICS

ROBOTIC SIMULATION

1 Abstract

This report begins by calculating joint angles as a function of end effector locations. The centre of mass is calculated as a function of joint angles, and the optimal choice for the length L_0 of the feet is considered. Simplifying constraints are introduced which jointly specify a unique and valid trajectory to navigate a given terrain. A cubic equation for motion is derived, based on such constraints. Kinetic and potential energies are calculated and used to justify a static approximation to total torque. Instantaneous power in the motors is calculated, and used to calculate total energy consumption and efficiency.

2 Calculating joint angles

One of the most important first steps in the robotic simulation is finding a geometric relationship between the positions of the links and the angles that those links make with each other; for example, later on, these joint-angles are numerically differentiated to find angular velocities for each joint, which are multiplied by motor torques to find instantaneous power in each of the motors. Firstly, the angles are calculated as a function of the coordinates of the feet, assuming that the feet constantly remain horizontal (this and other simplifying constraints are discussed more thoroughly in Section 4). As shown in Figure 1, we denote by α the distance between the two outermost motors, which can be easily calculated using Pythagoras' theorem:

$$\alpha = \sqrt{(x_5 - x_1)^2 + (y_5 - y_1)^2}$$

Next, by considering the sum of the angles within the isosceles triangle formed by the three motors, we relate θ_2 to β (where these quantities are most easily described by their locations within Figure 1):

$$\beta = \frac{\theta_2}{2}$$

Next, by considering the right-angled triangle formed by one half of the isosceles triangle:

$$L \cos\left(\frac{\theta_2}{2}\right) = \frac{\alpha}{2}$$

$$\Rightarrow \theta_2 = 2\,\mathrm{acos}\left(\frac{\alpha}{2L}\right)$$

By considering the vertical component of the line whose length is α, we obtain a value for θ_1; since we constrain both feet to be horizontal, θ_3 is found trivially using θ_1 and θ_2:

$$\alpha \cos\left(\theta_1 + \frac{\theta_2}{2}\right) = y_5 - y_1$$

$$\Rightarrow \theta_1 = \mathrm{acos}\left(\frac{y_5 - y_1}{\alpha}\right) - \frac{\theta_2}{2}$$

$$\theta_3 = 180 - \theta_1 - \theta_2$$

These results can be implemented using Python code (as a method within a `TwoLegRobot` class, discussed more in Section 4) as follows (noting that Python uses 'zero-indexing', so `x[0]` refers to x_1):

```
def affectors_to_angles(self, x, y, theta):
    # Calculate distance between end affectors:
    alpha = np.sqrt((x[4] - x[0])**2 + (y[4] - y[0])**2)
    # Calculate angles:
    theta[1] = 2 * acosd(alpha / (2*self.L))
    theta[0] = acosd((y[4] - y[0]) / alpha) - theta[1] / 2
    theta[2] = 180 - theta[0] - theta[1]
    return theta
```

For example, if $(x_0, y_0) = (0, 0)$ and $(x_5, y_5) = (0.15, 0.05)m$, then $(\theta_1, \theta_2, \theta_3) = (33.80, 75.54, 70.66)$ degrees; if instead $(x_0, y_0) = (0, 0)$ and $(x_5, y_5) = (0.12, 0.00)m$, then $(\theta_1, \theta_2, \theta_3) = (36.87, 106.26, 36.87)$ degrees.

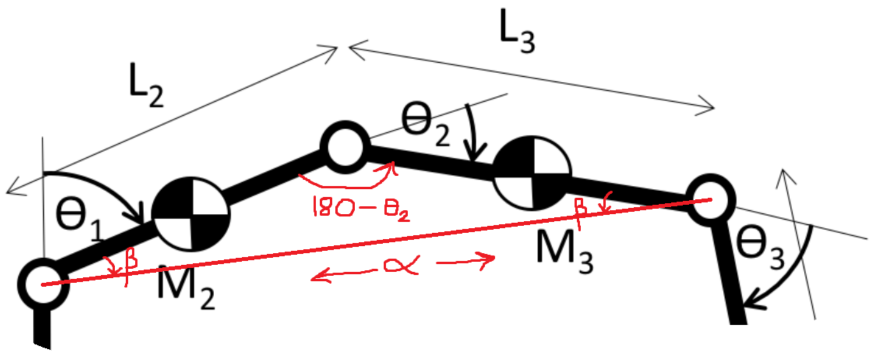


Figure 1: A geometrical picture of positions of the links and the angles that those links make with each other

3 Centre of mass

Next, we consider the centre of mass of the robot; this is important to consider when using 'static locomotion' (IE assuming that the robot is moving sufficiently slowly to assume that the robots mechanics can be modelled with statics), in order to guarantee that the robot won't topple over as a result of the vertical projection of its centre of mass being outside the convex hull of the robot's contact points with the ground. In general, for a body containing N discrete points with masses $\{m_n\}_{n=1}^{N}$ and horizontal positions $\{x_{m_n}\}_{n=1}^{N}$, the horizontal coordinate $\bar{x}$ of the centre of mass is the point that satisfies:

$$\sum_{n=1}^{N} m_n \bar{x} = \sum_{n=1}^{N} m_n x_{m_n}$$

$$\Rightarrow \bar{x} = \frac{\sum_{n=1}^{N} m_n x_{m_n}}{\sum_{n=1}^{N} m_n}$$

For the robot shown in Figure 2, we can express $\{x_{m_n}\}_{n=1}^{4}$ in terms of $(\theta_1, \theta_2, \theta_3)$ and x_1 as follows, assuming that foot 1 is horizontal:

$$x_{m_1} = x_1$$

$$x_{m_2} = x_1 + \frac{1}{2}L\sin(\theta_1)$$

$$x_{m_3} = x_1 + L\sin(\theta_1) + \frac{1}{2}L\sin(\theta_1 + \theta_2)$$

$$x_{m_4} = x_1 + L\sin(\theta_1) + L\sin(\theta_1 + \theta_2) + \frac{1}{2}L\sin(\theta_1 + \theta_2 + \theta_3)$$

Substituting these expressions into the formula for $\bar{x}$, and assuming $m_1 = m_2 = m_3 = m_4$ we obtain:

$$\bar{x} = x_1 + \frac{L}{8}(5\sin(\theta_1) + 3\sin(\theta_1 + \theta_2) + \sin(\theta_1 + \theta_2 + \theta_3))$$

Similarly, we can derive the vertical coordinate $\bar{y}$ of the centre of mass as:

$$\bar{y} = y_1 + \frac{L}{8}(7 + 5\cos(\theta_1) + 3\cos(\theta_1 + \theta_2) + \cos(\theta_1 + \theta_2 + \theta_3))$$

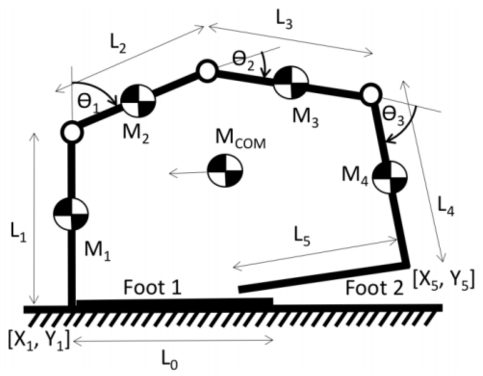


Figure 2: Illustration of the locations of the centres of mass of the robot's individual links

Next, we consider the constraints on the length L_0 of foot 1 in order to ensure that the robot will not topple over during motion; ignoring the constraint that both feet must remain horizontal during motion, $\bar{x}$ is maximised when $(\theta_1, \theta_2, \theta_3) = (90, 0, 0) \Rightarrow \bar{x} = x_1 + \frac{9}{8}L$; however, this position is not considered useful or realistic during locomotion, and reinstating the constraint that the feet must remain horizontal, it is found that $\bar{x}$ is maximised when $(\theta_1, \theta_2, \theta_3) = (90, 0, 90) \Rightarrow \bar{x} = x_1 + L$. Therefore, if we set $L_0 = L$ (plus possibly some tolerance), then we can guarantee that the robot doesn't topple over during motion (assuming that both feet remain horizontal, there are no external forces apart from gravity, and the robot does not move fast enough to topple over due to a sudden change in momentum); assuming that neither foot can pass through the 'ankle' just above the other foot, the maximum stride length is then also equal to L, and the maximum stride is equivalent to the robot moving between the two positions shown below in Figure 3.

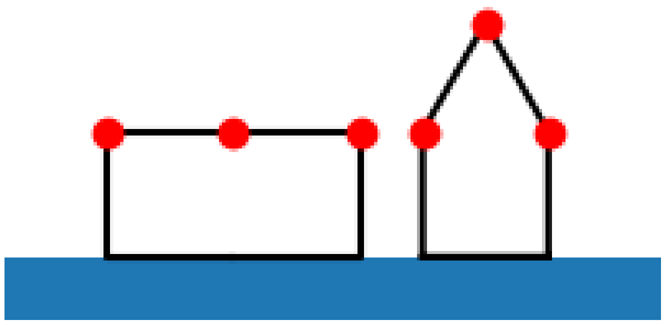


Figure 3: Illustration of the maximum stride length of a walking robot with foot length $L_0 = L$

Setting the foot length L_0 to be too large has two major disadvantages: firstly the maximum stride length is reduced, because the two feet are unable to get too close together, and secondly the assumption that the feet are light will become inaccurate, and the weight of the feet will require more work to be done by the system, reducing energy efficiency and increasing the maximum torque required from the motors.

4 Navigation

In this section, we consider the robot as it traverses the terrain shown below in Figure 4, including trajectories of joint angles, minimum torque for each of the three motors, duration required to complete the task, and energy efficiency. Without the simplifying constraints mentioned in earlier sections, this is a very under-constrained problem; there are many valid trajectories that finish with the robot reaching the goal, and due to the extreme non-linearity of the problem, it is difficult to find the 'optimal' trajectory; instead, we impose a set of principled simplifying constraints, which jointly specify a unique, valid trajectory for the given terrain, given a small set of parameters.

First of all, we assume that it is realistic to allow step changes in acceleration, but unrealistic to allow step changes in velocity, since this would require infinite torque in the motors. In order to make sure that the velocities vary smoothly to zeroth order (IE without assuming that its gradients are smooth as well), we consider only motions in which the velocity profiles of the end effectors vary quadratically, with zero initial and final velocity, corresponding to cubic position trajectories for the end effectors (we derive the equations for those trajectories below). We also assume that the mean speed of the robot is constant (meaning that the time taken for each cubic position trajectory is proportional to the total distance travelled by the end effector during the motion)[1], and that all of the motions made by the end effectors are either strictly horizontal or strictly vertical (which makes it significantly easier to navigate the step without collision). Finally, we assume that each step taken by the robot consists of lifting one of the feet up vertically by a distance y_{lift}, translating that foot horizontally by a distance x_{step}, and then lowering that foot vertically back down to the floor by a distance y_{lift} (except for when the robot approaches the step, in which case it needs to lift up its foot vertically by a distance $y_{lift} + h$, where h is the height of the step). Together, these simplifying constraints lead to a unique and valid trajectory for the robot, subject to parameters which need to be chosen (x_{step}, y_{lift}, and the mean speed of the cubic motion trajectories).

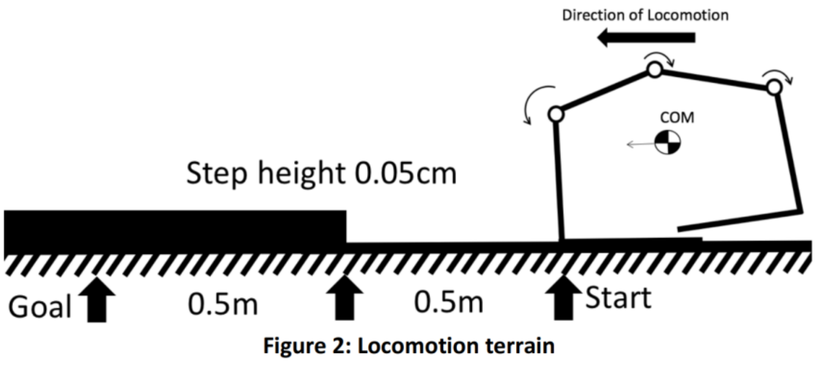


Figure 4: Illustration of the terrain which the robot must navigate

Cubic motion trajectories

In this sub-section we derive the equation for the cubic position trajectory of the end effectors; while foot 1 is clamped to the floor, we consider a quadratic velocity profile for the vertical component of the

[1] An interesting alternative would be to consider a fixed initial/maximum absolute acceleration (the two are equivalent)

velocity of the end effector, $\dot{y}_5(t)$, which occurs for a time-period of length T, subject to the following constraints:

$$\dot{y}_5(t) = at^2 + bt + c \qquad \int_0^T \dot{y}_5(t)\,dt = y_{lift} \qquad \dot{y}_5(0) = \dot{y}_5(T) = 0$$

The constraint of zero initial and final velocities implies:

$$c = 0$$

$$b = -aT$$

$$\Rightarrow \dot{y}_5(t) = at(t - T)$$

The constraint on the total distance travelled during the motion implies:

$$\frac{1}{3}aT^3 - \frac{1}{2}aT^3 = y_{lift}$$

$$\Rightarrow a = -\frac{6y_{lift}}{T^3}$$

We therefore obtain the following equations for the velocity and position of the end effector during the motion:

$$\dot{y}_5(t) = \frac{6y_{lift}}{T^3}t(T - t)$$

$$y_5(t) = \frac{y_{lift}}{T^3}t^2(3T - 2t) + y_5(0)$$

After choosing a number N of time steps to simulate, this trajectory can be implemented as a Python function using the following 3 lines of code:

```python
def cubic_motion_trajectory(start_pos, distance, T, N):
    t = np.linspace(0, T, N)
    return distance*(t**2)*(3*T - 2*t)/(T**3) + start_pos
```

With this function implemented, it is possible to create the robot class and simulate as it navigates the terrain; the full 186 lines of code used by the `TwoLegRobot` class are too lengthy to include in this report, however the three major high-level methods are included below:

```python
def take_horizontal_step(self, y_lift, x_step):
    self.lift_left(y_lift)
    self.step_left(x_step)
    self.lift_left(-y_lift)
    self.lift_right(y_lift)
    self.step_right(x_step)
    self.lift_right(-y_lift)

def walk_distance(self, distance):
    x_init = self.x[0, -1]
    # Keep taking full steps until a full step would be too much:
    while self.x[0, -1] + self.x_step - x_init > distance:
        self.take_horizontal_step(self.y_lift, self.x_step)
    # Take a final step of the right length:
    final_step_length = distance + x_init - self.x[0, -1]
    self.take_horizontal_step(self.y_lift, final_step_length)
```

```
def navigate_step(self, step_height):
    self.lift_left(step_height + self.y_lift)
    self.step_left(self.x_step)
    self.lift_left(-self.y_lift)
    self.lift_right(step_height + self.y_lift)
    self.step_right(self.x_step)
    self.lift_right(-self.y_lift)
```

Finally, the navigation of the terrain can be simulated using the following five lines of code from within a main script:

```
r = TwoLegRobot()
r.walk_distance(-0.49)
r.navigate_step(0.05)
remaining_distance = -1.0 - r.x[0, -1]
r.walk_distance(remaining_distance)
```

A visual depiction of the robot navigating this terrain is shown below in Figure 5:

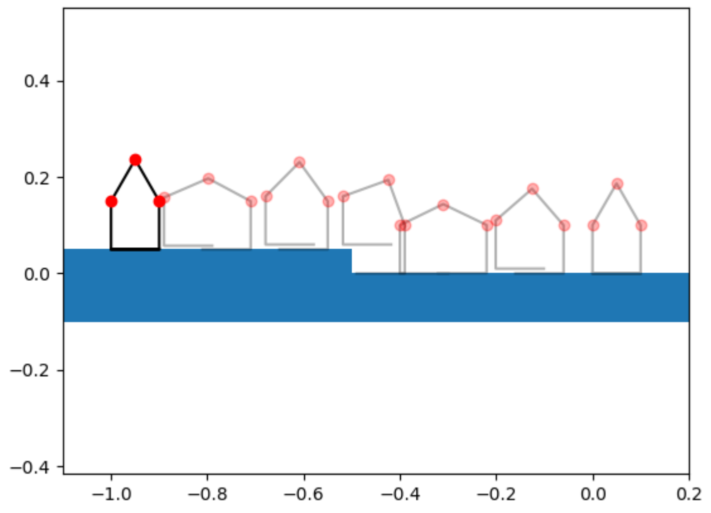
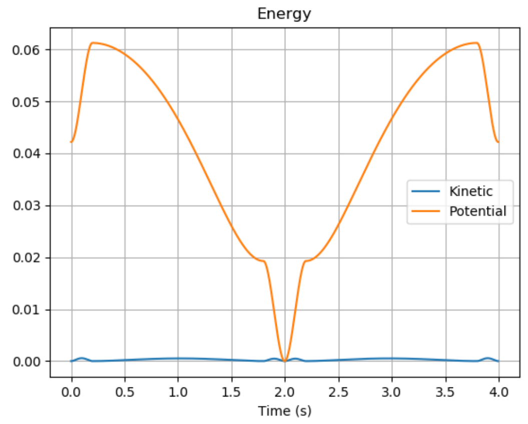


Figure 5: Illustration of the simulated robot navigating the terrain

Figures 7 and 8 in the Appendix show trajectories for the vertical, horizontal and angular position, velocity, and acceleration (found using numerical differentiation), both for taking a single step and for navigating the entire terrain respectively, using $y_{lift} = 0.01m$, $x_{step} = -0.08m$ and speed = 0.05m/s, corresponding to a single step with one foot taking two seconds; we assume this to be a reasonable time period per step for a robot with low-power motors using static locomotion. With these parameter settings, the total time required for the robot to reach the goal after climbing the step is 53.119 s.

Energy, torque and power

We begin this section by calculating the instantaneous kinetic and (gravitational) potential energies for the robot throughout the navigation task; as well as being simple to calculate, observing these quantities for different parameter settings serves as partial justification for the approximations used later when calculating torque. Kinetic energy at each time instance is found by squaring and summing the velocity components of the centre of each link, and scaling by half the mass of each link; relative potential energy is found by summing the vertical position of the centre of each link, and scaling by its mass and by gravitational acceleration. Figure 6 below shows the energy profiles for a single step, for mean effector speeds of 0.05m/s (left; considered originally) and 0.5m/s (right); as shown in the plots, for the mean effector speeds being considered, the kinetic energy is negligible compared to the variations in potential energy due to gravity; this serves as part of the justification for the approximations used for calculating motor torque.



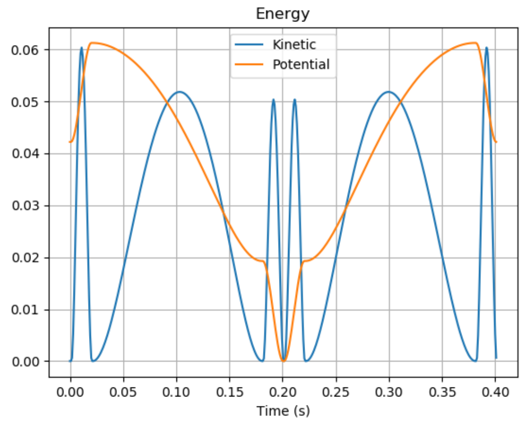


Figure 6: The effect of mean effector speed on kinetic energy relative to potential energy. Left: mean effector speed = 0.05m/s; right: mean effector speed = 0.5m/s (right). Units are in Joules

Ideally, motor torque would be calculated by deriving an equation of motion for the system (possibly in the form of the Euler-Lagrange equation), and solving for the torques in terms of the quantities that are already known (EG positions, velocities and accelerations); however, since the kinetic energy of the robot during its motion is negligible compared to the potential energy for the mean effector speeds that are considered realistic, we instead use a 'static' approximation to the torque, using purely moments of gravitational force about motor positions; we also justify this using an order-of-magnitude approximation to the 'dynamic' torques that would occur throughout the motion as a result of angular accelerations and moments of inertia as follows: as can be seen in the 'Joint angular accelerations' sub-plot of Figure 7, the absolute angular acceleration barely surpasses 15 rad s^{-2}; we use this value below to compare order-of-magnitude calculations for static vs dynamic torque:

Static torque:
$$\tau \propto mgL$$
$$m = 0.1kg$$
$$g = 9.81ms^{-1}$$
$$L = 0.1m$$
$$\Rightarrow \tau \propto 0.0981$$

Dynamic torque:
$$\tau \propto m\ddot{\theta}L^2$$
$$m = 0.1kg$$
$$|\ddot{\theta}| < 15 \text{ rad } s^{-2}$$
$$L = 0.1m$$
$$\Rightarrow \tau \propto 0.015$$

As shown above, based on these rough calculations, the static torque is close to an order of magnitude greater than the dynamic torques for the parameters in question, therefore we approximate torque entirely by static contributions due to gravity. By considering free body diagrams of the robot, it is straightforward to show that when foot 1 is clamped to the ground, the torques in the three motors are given by the following expressions (where a clockwise-positive sign convention is used):

$$\tau_1 = \frac{5}{2}mg\left(x_2 - \frac{2}{5}x_3 - \frac{3}{5}x_4\right) \qquad \tau_2 = \frac{3}{2}mg(x_3 - x_4) \qquad \tau_3 = 0$$

Similarly, when foot 1 is clamped to the ground, the torques in the three motors are given by the following expressions:

$$\tau_1 = 0 \qquad \tau_2 = \frac{3}{2}mg(x_3 - x_2) \qquad \tau_3 = \frac{5}{2}mg\left(x_4 - \frac{2}{5}x_3 - \frac{3}{5}x_2\right)$$

Trajectories for the static torques are shown in Figures 7 and 8 in the Appendix; as a result of the assumption of static torque, these values do not change as the mean effector speed is varied, which is clearly unrealistic at higher speeds. For the parameters considered above, the maximum absolute torque throughout the navigation task in each of the three motors is 0.363, 0.147, and 0.353 Nm respectively.

Next, we calculate instantaneous power in the motors, by multiplying torque by angular velocity; there is one slight complication, which is that in order to receive correct and sensible trajectories for instantaneous power, it is necessary to flip the sign of the angular velocity when foot 2 is clamped and foot 1 is moving; this is because although the sign convention remains consistent (clockwise-positive), in this case the link which is considered stationary and the link which is considered rotating relative to that stationary link are exchanged, and so in order to maintain a consistent sign-convention, it is necessary to flip the sign of the angular velocity. This can be achieved easily using the following six lines of code shown below:

```python
def instantaneous_power(torque, thetadot, foot1_clamped, foot2_clamped):
    # Indices when foot 1 is not clamped and foot 2 is clamped:
    inds = ~foot1_clamped & foot2_clamped
    # Create array which is flipped when foot 1 is not clamped and foot 2 is
    # clamped and is otherwise the same:
    thetadot_flipped = np.zeros(thetadot.shape)
    thetadot_flipped[:, inds] = -thetadot[:, inds]
    thetadot_flipped[:, ~inds] = thetadot[:, ~inds]
    return torque * thetadot_flipped
```

Finally, we calculate total energy consumed by the motors by clipping the negative instantaneous powers to zero (these negative instantaneous powers occur EG when a link is rotating downwards in the direction of gravity), and integrating the resulting trajectory with respect to time; negative instantaneous powers are clipped because we assume that the robot has no built-in capacity to use the work done by gravity to recharge its batteries when moving in the direction of gravitational acceleration. For the parameters considered above, the total energy consumed by the motors during the navigation task is 4.644 J, assuming perfectly efficient motors; for reference, the change in gravitational potential energy from start to finish is equal to 0.196 J, corresponding to an energy efficiency of 4.225%. This value is sensitive to the parameters used by the robot; for example, if y_{lift} is reduced from 0.01 to 0.005, the energy efficiency increases to 4.592%, which demonstrates the general effect of small changes in motor trajectories on overall performance metrics. However, the mean speed of each motion does not affect the energy efficiency in this model, due to the assumption of static torque, which is clearly unrealistic.

5 Summary

This report began in Section 2 by calculating joint angles as a function of end effector locations and implementing this in Python. In Section 3, the centre of mass was calculated as a function of joint angles, and this was used to consider the optimal choice for the length L_0 of the feet, and the consequences of choosing a length which is too large. Section 4 focused on the robot navigating a given terrain (including a step to be climbed), for which simplifying constraints were introduced which jointly specified a unique and valid trajectory, based on choices for parameters. This required an appropriate cubic equation for motion being derived, with zero initial and final velocity, and specified travel distance and time period. Next, kinetic and potential energies were calculated and compared for different choices of parameters, and served as the basis for a static approximation to total torque, along with order-of-magnitude calculations using calculated angular accelerations. After stating the equations for static torque, instantaneous power in the motors was calculated, and this instantaneous power was integrated to calculate total energy consumption and efficiency by considering the total change in gravitational potential energy. Finally, it was remarked that small changes in motor trajectories can have a noticeable impact on overall performance metrics.

6 Appendix

Trajectories for a single step

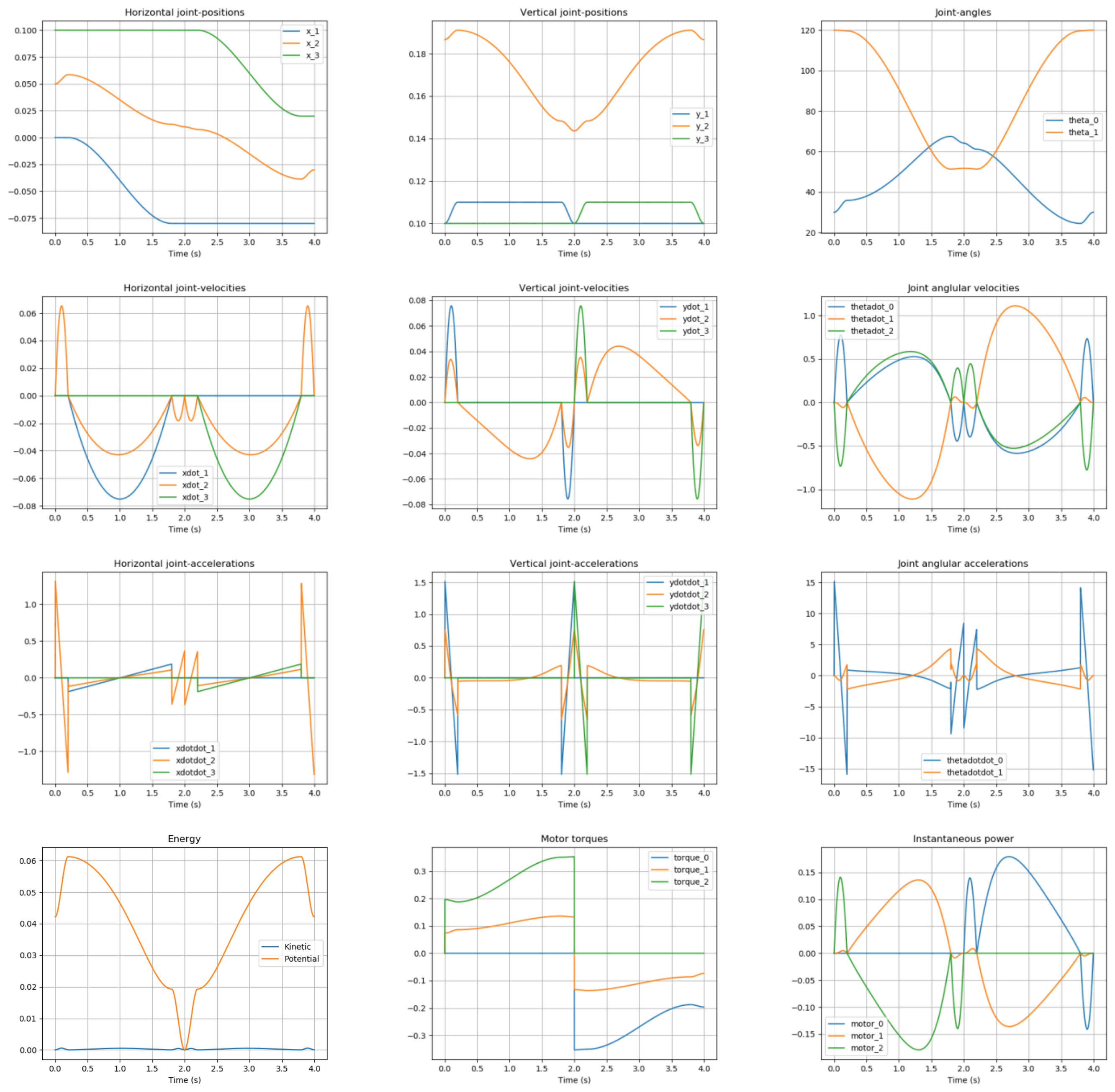


Figure 7: Trajectories for the vertical, horizontal and angular position, velocity, and acceleration (found using numerical differentiation), energy, torque, and power, for a single step. Angles are measured in degrees, but angular velocities and accelerations are measured in radians per second and radians per second per second, respectively; all other units are SI units

Trajectories for navigating the full terrain

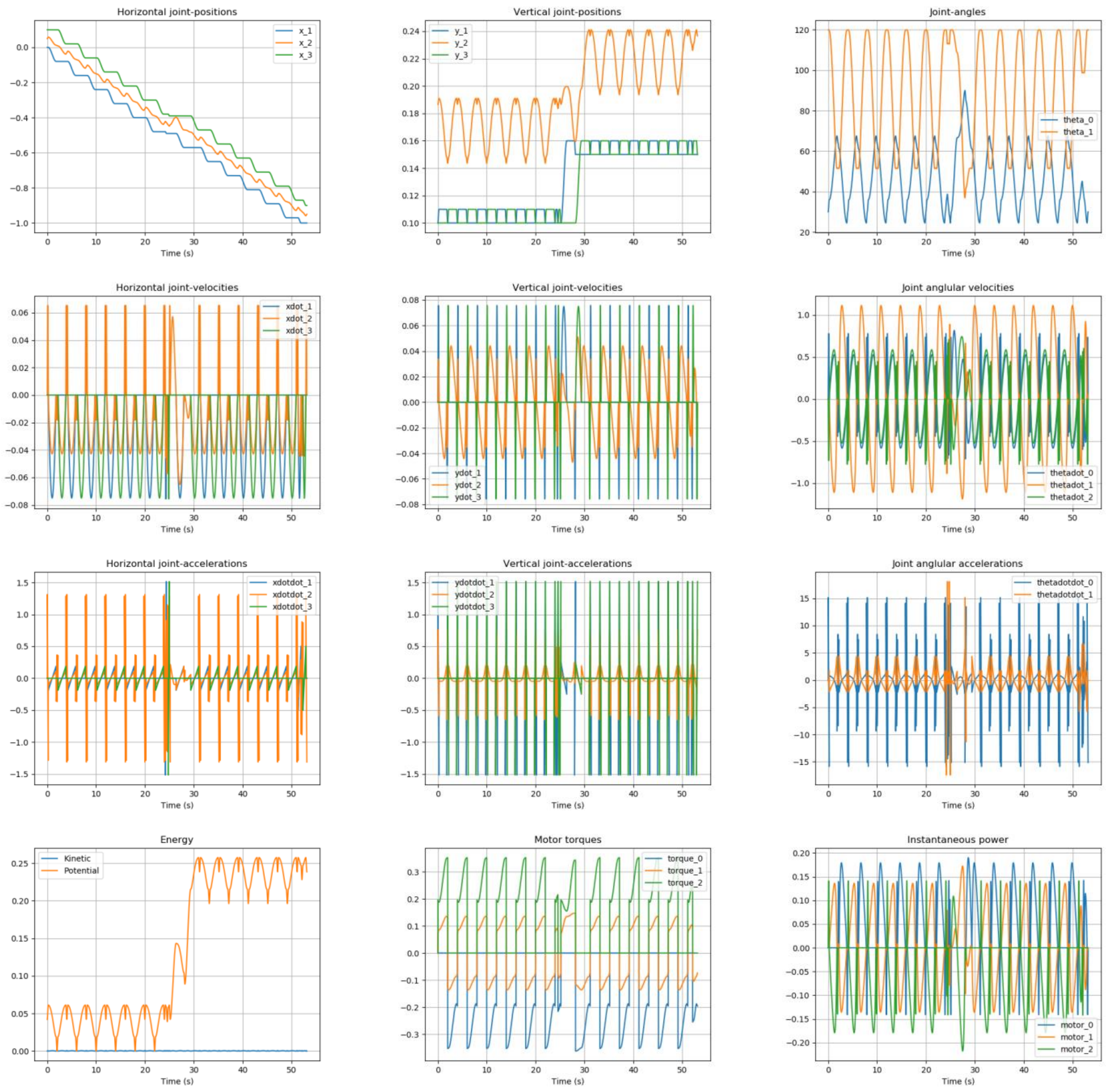


Figure 8: Trajectories for the vertical, horizontal and angular position, velocity, and acceleration (found using numerical differentiation), energy, torque, and power, for navigating the entire terrain of Figure 4. Angles are measured in degrees, but angular velocities and accelerations are measured in radians per second and radians per second per second, respectively; all other units are SI units