EE587 INTRODUCTION TO ROBOTICS

FINAL PROJECT

Report

SEA URCHIN ROBOT

Name and Surname:
Alper Kayabaşı

Student ID:
2535961

CODE AND VIDEO IS AVAILABLE.

February 5, 2022

Contents

List of Tables

List of Figures

Kinematic Analysis of Sea-Urchin

Forward Kinematic of the Sea-Urchin Robot

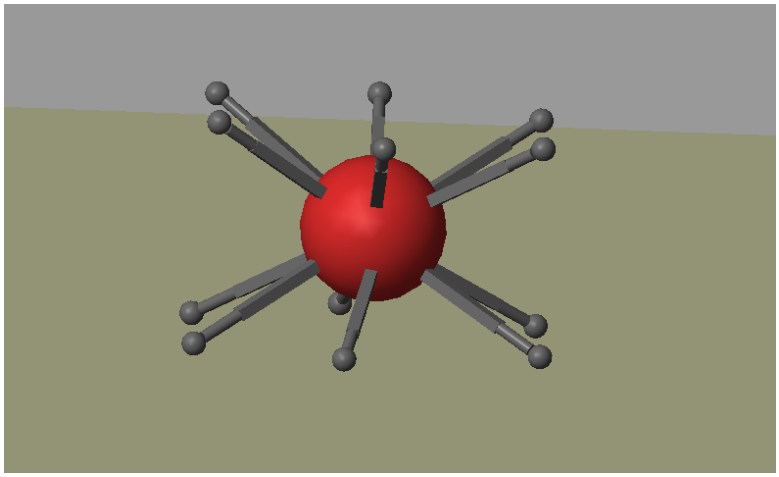


Figure 1.1: 3D Model of Robot

Sea-Urchin robot consists of spherical shell with radius r_s and 12 spikes that are connected to it. Each spike is constructed by rectangular prism, cylinder, and small sphere. Body has 6-DoF while spikes have 3-DoF. Although complete appearance of the robot is shown in Fig.1.1, one particular spike is taken into account in Fig.1.2 to understand mobility of robot and other legs are hidden for simplicity in Fig.1.2.

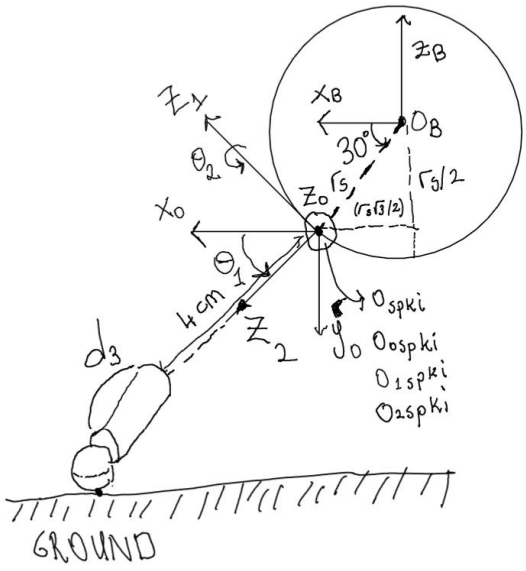


Figure 1.2: Front Profile of Robot

Cylinder parts of spike are capable of retracting and extending from the end of the rectangular prism and they can extend 4 cm maximally from the end of rectangular prism whose length equals to 4 cm. This prismatic degree of freedom is represented with d_3 in Fig.1.2. Spike can roll around z_1 axis tangent to spherical shell in Fig.1.2 so that urchin can move its spike

forward and backward. Spike also can roll around z_0 axis in Fig.1.2 that points out of page so that urchin can move its spike inward and outward. Furthermore, small sphere with radius of 0.5 cm in Fig.1.2 make contact with floor. Position of bottom spikes around sphere are clarified by using intersection of shell of sphere and auxiliary plane whose equation equals to $z_b = -r_s/2$. This intersection corresponds to circle and spikes pivot from points on circle where each connection point is located around circle at intervals of **60 degree**. Same configuration is valid for upper part except upper spikes are positioned on intersection of spherical shell and plane whose equation equals to $z_b = r_s/2$. Frames are assigned for the given geometry based on Denavit-Hartenberg (DH) convention. Entries of DH table given in 1.1 are parameterized for i^{th} spike. If d_i has negative sign, table corresponds to bottom spikes. Positive signed d_i implies upper spikes.

Table 1.1: Denavit-Hartenberg assignment for spk_i.

Link	d_i	a_i	θ_i	α_i
O_B - O_{spk_i}	$\pm\frac{r_s}{2}$	$\frac{r_s\sqrt{3}}{2}$	θ_{spk_i}	-90
$O_{spk_i} - O_{0spk_i}$	0	0	θ_{1spk_i}	90
$O_{0spk_i} - O_{1spk_i}$	0	0	θ_{2spk_i}	90
$O_{1spk_i} - O_{2spk_i}$	$0.04 + d_3$	0	0	0

Spikes are numbered starting from one to six for both bottom and upper part in counter clockwise direction. Bottom spike and upper spike with number i have constant θ_{spk_i} joint angle. Since upper and bottom spikes are horizontally symmetric with each other, there is no need to extra index. Spikes are called with upper and bottom to prevent confusion whenever they are referred. Angle constant for i^{th} leg is given in Equation 1.1. In majority of computation, reference frame are taken as O_{spk_i}; however, first row in Table 1.1 are used in construction of robot in simulation during placement of spikes.

$$\theta_{spk_i} = 60 \times (i-1) \; for \; i = 1, 2, 3, 4, 5, 6 \tag{1.1}$$

Forward kinematic of spikes are computed based on last three rows in Table 1.1 since trajectory of each spike are assigned from reference frame O_{spk_i} and derived forward kinematic equations are benefited in computation of inverse kinematic and Jacobian. Trajectory computation of spikes will be given in details after inverse kinematic derivation in next subsection. Derivation of forward kinematic are given below:

$$A^{Ospk_i}_{O_{0spk_i}} = \begin{bmatrix} cos(\theta_1) & 0 & sin(\theta_1) & 0 \\ sin(\theta_1) & 0 & -cos(\theta_1) & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A^{O_{O_{0spk_i}}}_{O_{1spk_i}} = \begin{bmatrix} cos(\theta_2) & 0 & sin(\theta_2) & 0 \\ sin(\theta_2) & 0 & -cos(\theta_2) & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
$$A^{O_{1spk_i}}_{O_{2spk_i}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_3 + 0.04 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.2}$$

Equations in 1.2 represent homogeneous transformation corresponding to last three row in Table 1.1. Upper index of homogeneous transformation are taken as reference frame while the bottom index expresses target frame. For simplicity, I drop bottom index spk_i which express transformation for concerned spike inside matrices in both Equation 1.2, 1.3, and 1.4.

$$A_{O_{1spk_i}}^{O_{spk_i}} = A_{O_{0spk_i}}^{O_{spk_i}} \times A_{O_{1spk_i}}^{O_{0spk_i}} = \begin{bmatrix} cos(\theta_2)cos(\theta_1) & sin(\theta_1) & cos(\theta_1)sin(\theta_2) & 0 \\ sin(\theta_1)cos(\theta_2) & -cos(\theta_1) & sin(\theta_1)sin(\theta_2) & 0 \\ sin(\theta_2) & 0 & -cos(\theta_2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.3}$$

$$A_{O_{2spk_i}}^{O_{spk_i}} = \begin{bmatrix} cos(\theta_2)cos(\theta_1) & sin(\theta_1) & cos(\theta_1)sin(\theta_2) & cos(\theta_1)sin(\theta_2)(d_3+0.04) \\ sin(\theta_1)cos(\theta_2) & -cos(\theta_1) & sin(\theta_1)sin(\theta_2) & sin(\theta_1)sin(\theta_2)(d_3+0.04) \\ sin(\theta_2) & 0 & -cos(\theta_2) & -cos(\theta_2)(d_3+0.04) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.4}$$

Inverse Kinematic of the Sea-Urchin Robot

Inverse kinematic is used to transform task-space trajectory into joint space trajectory in this work. Its computation is very important for control design in joint space. Derivation for forward kinematic are given for the end of cylinder in Fig.1.2. However, task space trajectories for spikes are planned for the point where small sphere belonging to spike touches ground. Therefore, position of end of cylinder is computed from contact point and inverse kinematic solution is performed based on derived forward kinematic equations. Let us define $\vec{x_e}$ as point where small sphere touches ground. Although sphere does not have to contact in one unique point with ground due to possible flexibility in real scenario, I **assume** that one unique point exists in every configuration.

$$\begin{aligned} \vec{c} =& \vec{x_e} - 0.005j - \hat{z_2} * 0.005 \\ j = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad & \hat{z_2} = \begin{bmatrix} cos(\theta_1)sin(\theta_2) \\ sin(\theta_1)sin(\theta_2) \\ -cos(\theta_2) \end{bmatrix} \end{aligned} \tag{1.5}$$

Under this assumption, end of cylinder are computed by using direction of cylinder, $\hat{z_2}$, that corresponds to first three elements of third column in Equation 1.4 and unit vector in y_0 direction as shown in Fig. 1.2. I obtain position vector for center of small sphere by moving in negative y_0 direction as much as radius of small sphere. After that, I reach position of end of cylinder, $\vec{c}$, by moving backward in $\hat{z_2}$ direction as much as radius of small sphere.

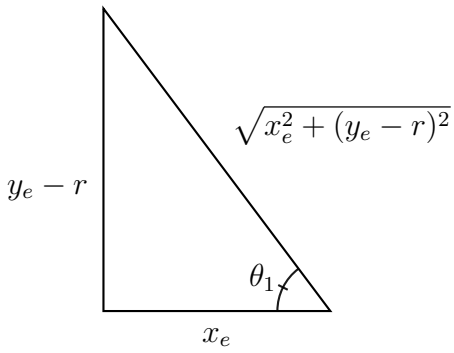
$$\vec{c} = \begin{bmatrix} x_e \\ y_e - 0.005 \\ z_e \end{bmatrix} - \begin{bmatrix} 0.005cos(\theta_1)sin(\theta_2) \\ 0.005sin(\theta_1)sin(\theta_2) \\ -0.005cos(\theta_2) \end{bmatrix} = \begin{bmatrix} x_e - 0.005cos(\theta_1)sin(\theta_2) \\ y_e - 0.005 - 0.005sin(\theta_1)\sin(\theta_2) \\ z_e + 0.005cos(\theta_2) \end{bmatrix} \tag{1.6}$$

First three element of last column of matrix in Equation 1.4 also represent position of end of cylinder and I equate Equation 1.6 to the first three element of last column in 1.4 so that I find required joint angle that satisfy desired position for contact point. Let **r** and **k** corresponds to radius of small sphere and length of rectangular prism so I can adjust radius of the sphere and length of the prism to make robot walk in balance empirically in construction phase.

$$\begin{aligned} x_e =& cos(\theta_1)sin(\theta_2)(r+k+d_3) \\ y_e - 0.005 =& sin(\theta_1)sin(\theta_2)(r+k+d_3) \\ z_e =& -cos(\theta_2)(r+k+d_3) \end{aligned} \tag{1.7}$$

Second equation is divided by first equation in Equation 1.7 to solve θ_1 angle.

$$\frac{y_e - r}{x_e} = \frac{sin(\theta_1)sin(\theta_2)(r+k+d_3)}{cos(\theta_1)sin(\theta_2)(r+k+d_3)} = tan(\theta_1) \quad \theta_1 = atan2\left(\frac{y_e - r}{x_e}\right) \tag{1.8}$$



Based on triangle given above, $\sin(\theta_1)$ can be found. This is substituted for $\sin(\theta_1)$ in equation 1.9 which is obtained by dividing second and third equality in Equation 1.7.

$$\begin{aligned} \frac{y_e - r}{z_e} &= \frac{sin(\theta_1)sin(\theta_2)}{-cos(\theta_2)} = \frac{(y_e - r)sin(\theta_2)}{-\sqrt{x_e^2 + (y_e - r)^2}cos(\theta_2)} \\ tan(\theta_2) &= \frac{sin(\theta_2)}{cos(\theta_2)} = \frac{-\sqrt{x_e^2 + (y_e - r)^2}}{z_e} \\ \theta_2 &= atan2\left(\frac{-\sqrt{x_e^2 + (y_e - r)^2}}{z_e}\right) \end{aligned} \tag{1.9}$$

Third equality in Equation 1.9 is divided by $-cos(\theta_2)$. After that, r and k are taken left of the equality to find d_3 as shown in Equation 1.10.

$$d_3 = \frac{-z_e}{cos(\theta_2)} - r - k \tag{1.10}$$

Matlab implementation of generic inverse kinematic function for spikes are given below in Fig. 1.3.

```
function [ang1,ang2,d3] = inverse_kinematic(fx,fy,fz,r,k)
ang1 = atan2(fy-r,fx);
ang2 = atan2(sqrt((fy-r)^2 + fx^2),-fz);
d3 = (-fz/cos(ang2)) -r -k;
end
% K corresponds to constant leg length of urchin
% R corresponds to amount of extension of spike after the leg part
```

Figure 1.3: Matlab function for inverse kinematic of spike

Trajectory Planning

Walking Trajectory

A gait is the order of manner of landing and lifting of spikes to provide a walking procedure. For spikes of robot, there are two phases: the stance phase and transfer phase. When walking, the order of changing between stance and transfer phase will define gait of the robot. During transfer phase, leg traverses from point 2 to point 1 through air as shown in Figure 2.1. Transfer phase consists of two phase that are numbered as 2 and 3 in Figure 2.1. Spike takes off to height H' in first phase numbered as 2 while it lands off initial point 1 in second phase numbered as 3. Transfer phase aims to make spikes ready for stance phase. In stance phase, spike constantly touches ground while it traverses from point 1 to point 2 so that body of urchin robot moves in opposite of stance direction. During stance phase velocity of tip of spike is constant in z-direction.

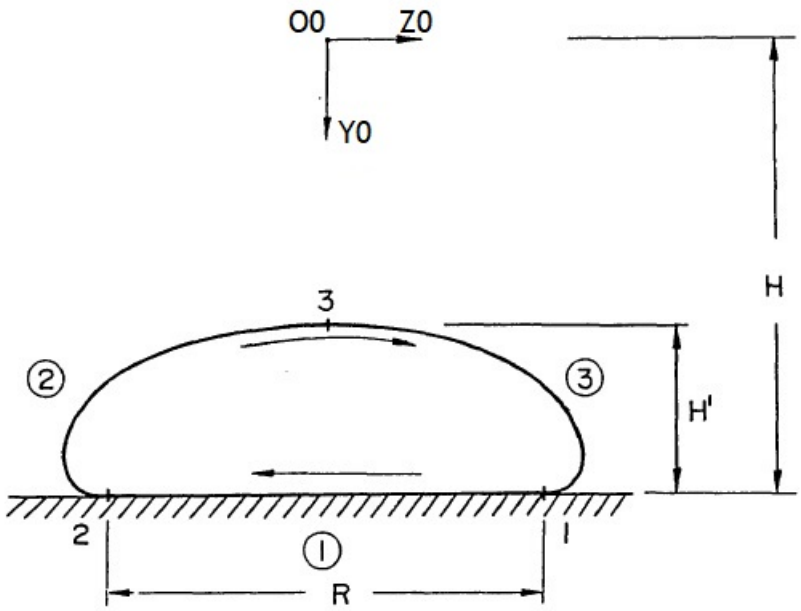


Figure 2.1: Transfer and Stance Phase [1]

Trajectory of the foot is defined by the following parameters:

- R = spike stroke length
- H = height of the spike trajectory
- H' = height of the connection point of spikes to spherical shell
- T (cycle period) = T_{stance} + T_t, where T_t corresponds to transfer phase time
- $T_t = T_h + T_f$, where T_h and T_f corresponds to take-off and land-off time of spike.

Table 2.1: Initial and final Condition for spike during stance phase

Coordinate of Spike	Position	Velocity	Acceleration
$z_e\,(t=0)$	$\frac{R}{2}$	$-\frac{R}{T_{stance}}$	0
$z_e\,(t=T_{stance})$	$\frac{-R}{2}$	$-\frac{R}{T_{stance}}$	0

Based on constant velocity assumption, there are desired initial and final conditions that are necessary to be satisfied during stance phase. Conditions in Table 2.1 are used to find trajectory in z direction during stance phase.

$$z_e(t) = \frac{R}{2} - Vt \quad y_e(t) = 3\sqrt{3}cm \quad x_e(t) = 9cm \tag{2.1}$$

Quintic polynomial trajectories [3] can satisfy desired initial and final position, velocity, and acceleration. Fifth order polynomial trajectory is capable of satisfying six boundary condition for initial and final part of transfer phase. Quintic polynomial trajectory is given in Equation 2.2.

$$z_e(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \tag{2.2}$$

Stance and transfer phase trajectory are solved by taking initial time 0 but they are concatenated to each other in implementation since position at $z_e(t = T_t)$ coincide with initial z_e position during stance phase.

Table 2.2: Initial and final condition for spike during transfer phase

Coordinate of Spike	Position	Velocity	Acceleration
$z_e\,(t=0)$	$\frac{-R}{2}$	0	0
$z_e\,(t=T_t)$	$\frac{+R}{2}$	0	0
$y_e\,(t=0)$	$3\sqrt{3}$ cm	0	0
$y_e\,(t=T_h)$	4.2 cm	2 cm/s	0
$y_e\,(t=T_f)$	$3\sqrt{3}cm$	0	0

$$\begin{aligned}
z_e(t=0) &= a_0 + a_1(0) + a_2(0)^2 + a_3(0)^3 + a_4(0)^4 + a_5(0)^5 \\
\dot{z_e}(t=0) &= a_1 + 2a_2(0) + 3a_3(0)^2 + 4a_4(0)^3 + 5a_5(0)^4 \\
\ddot{z_e}(t=0) &= 2a_2 + 6a_3(0) + 12a_4(0)^2 + 20a_5(0)^3 \\
z_e(t=T_t) &= a_0 + a_1(T_t) + a_2(T_t)^2 + a_3(T_t)^3 + a_4(T_t)^4 + a_5(T_t)^5 \\
\dot{z_e}(t=T_t) &= a_1 + 2a_2(T_t) + 3a_3(T_t)^2 + 4a_4(T_t)^3 + 5a_5(T_t)^4 \\
\ddot{z_e}(t=T_t) &= 2a_2 + 6a_3(T_t) + 12a_4(T_t)^2 + 20a_5(T_t)^3
\end{aligned} \tag{2.3}$$

Equations in 2.3 are converted into matrix form for efficient solution.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 1 & T_t & T_t^2 & T_t^3 & T_t^4 & T_t^5 \\ 0 & 1 & 2T_t & 3T_t^2 & 4T_t^3 & 5T_t^4 \\ 0 & 0 & 2 & 6T_t & 12T_t^2 & 20T_t^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} z_e(t=0) \\ \dot{z}_e(t=0) \\ \ddot{z}_e(t=0) \\ z_e(t=T_t) \\ \dot{z}_e(t=T_t) \\ \ddot{z}_e(t=T_t) \end{bmatrix} \tag{2.4}$$

Equations are solved in MatLab with given in Appendix. Computed trajectories are transformed into joint space via inverse kinematic as shown in 2.2.

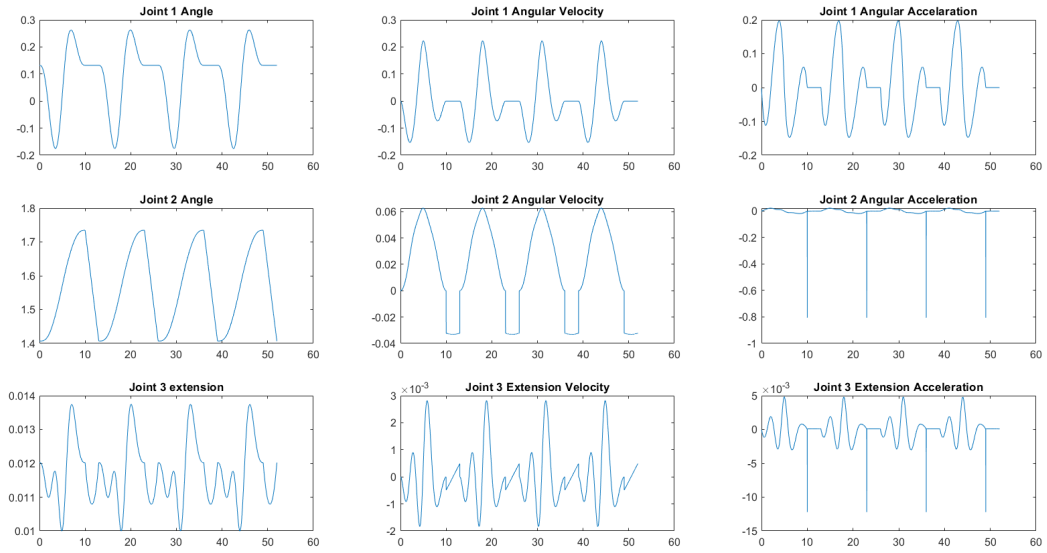


Figure 2.2: Joint space position,velocity, and acceleration

Derivation of Jacobian

Jacobian are necessary during force control and some of dynamic derivation. Whenever some force applies over tip of spike, it creates torque effect on joint which is governed by equation $\tau = J^T F$. It forms essential principle of force control.

Lower Part of Jacobian

Z directions in the derived kinematic equations are used to find jacobian for tip of spike since they constitutes rotation axis where spikes turn around. Lower part corresponds to angular sensitivity of tip of spike with respect to joint angle change so it relates joint space velocities to angular velocities. Third column of lower part of jacobian equals to $\mathbf{0_{3x1}}$ vector since last joint is prismatic.

$$\begin{gathered} z_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad z_1 = \begin{bmatrix} sin(\theta_1) \\ -cos(\theta_1) \\ 0 \end{bmatrix} \\ J_w = \begin{bmatrix} 0 & sin(\theta_1) & 0 \\ 0 & -cos(\theta_1) & 0 \\ 1 & 0 & 0 \end{bmatrix} \end{gathered} \tag{3.1}$$

Upper Part of Jacobian

Upper part of jacobian determines translational sensitivity of tip of spike with respect to joint angle change so it relates joint space velocities with translational velocity of end effector.

$$J_v = \begin{bmatrix} J_{v1} & J_{v2} & J_{v3} \end{bmatrix}$$

$$\begin{aligned} J_{v1} &= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \times \left(0.04 + \frac{d_3}{2}\right) \begin{bmatrix} cos(\theta_1)sin(\theta_2) \\ sin(\theta_1)sin(\theta_2) \\ -cos(\theta_2) \end{bmatrix} \\ J_{v1} &= \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \left(0.04 + \frac{d_3}{2}\right) \begin{bmatrix} cos(\theta_1)sin(\theta_2) \\ sin(\theta_1)sin(\theta_2) \\ -cos(\theta_2) \end{bmatrix} \\ J_{v1} &= \left(0.04 + \frac{d_3}{2}\right) \begin{bmatrix} -sin(\theta_1)sin(\theta_2) \\ cos(\theta_1)sin(\theta_2) \\ 0 \end{bmatrix} \\ J_{v2} &= \begin{bmatrix} sin(\theta_1) \\ -cos(\theta_1) \\ 0 \end{bmatrix} \times \left(0.04 + \frac{d_3}{2}\right) \begin{bmatrix} cos(\theta_1)sin(\theta_2) \\ sin(\theta_1)sin(\theta_2) \\ -cos(\theta_2) \end{bmatrix} \\ J_{v2} &= \begin{bmatrix} 0 & 0 & -cos(\theta_1) \\ 0 & 0 & -sin(\theta_1) \\ cos(\theta_1) & sin(\theta_1) & 0 \end{bmatrix} \left(0.04 + \frac{d_3}{2}\right) \begin{bmatrix} cos(\theta_1)sin(\theta_2) \\ sin(\theta_1)sin(\theta_2) \\ -cos(\theta_2) \end{bmatrix} \\ J_{v2} &= \left(0.04 + \frac{d_3}{2}\right) \begin{bmatrix} cos(\theta_1)cos(\theta_2) \\ sin(\theta_1)cos(\theta_2) \\ sin(\theta_2) \end{bmatrix} \\ J_{v3} &= \begin{bmatrix} cos(\theta_1)sin(\theta_2) \\ sin(\theta_1)sin(\theta_2) \\ -cos(\theta_2) \end{bmatrix} \end{aligned} \tag{3.2}$$

Dynamic of Sea-Urchin Robot

Spike dynamics for Walking

I use MATLAB symbolic toolbox to avoid erroneous computation by hand. I check formulation of Christoffel symbol, Jacobian assigned for center of mass of rectangular prism and cylinder over and over again in the script to prevent any mistake. I neglect existence of small sphere in equation of motion since its existence only aims to make contact with floor. During design, I design its mass and inertia of small sphere as low so that it has no considerable effect on equation of motions. Matlab script for equation of motion is given in Appendix. q_1, q_2, and q_3 corresponds to θ_1, θ_2, and d_3.

$$D(q) = \begin{bmatrix} 0.002q_3 sin(q_2)^2 + 8.4 \times 10^{-5} sin(q_2)^2 + 0.0125q_3^2 sin(q_2)^2 + 2 \times 10^{-15} & 0 & 0 \\ 0 & 0.0125q_3^2 + 0.002q_3 + 8.4 \times 10^{-5} & 0 \\ 0 & 0 & 0.05 \end{bmatrix} \tag{4.1}$$

$$C(q,\dot{q}) = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$$

$$\begin{aligned} c_{11} &= \dot{q_3}\left(0.0125q_3 sin(q_2)^2 + 0.001 sin(q_2)^2\right) + \dot{q_2}\left(0.0125 cos(q_2) sin(q_2) q_3^2 + 0.002 cos(q_2) sin(q_2) q_3 + 8.4 \times 10^{-5} cos(q_2) sin(q_2)\right) \\ c_{12} &= \dot{q_1} * sin(2q_2)\left(0.00625q_3^2 + 0.001q_3 + 4.2 \times 10^{-5}\right) \\ c_{13} &= \dot{q_1} sin(q_2)^2 \left(0.0125q_3 + 0.001\right) \\ c_{21} &= -2 \times 10^{-37} \dot{q_1} sin(2q_2)\left(3.125 \times 10^{34} q_3^2 + 5 \times 10^{33} q_3 + 2.1 \times 10^{32}\right) \\ c_{22} &= \dot{q_3}(0.0125q_3 + 0.001) \\ c_{23} &= \dot{q_2}(0.0125 * q_3 + 0.001) \\ c_{31} &= -10^{-34} * \dot{q_1} sin(q_2)^2 (1.25 \times 10^{32} q_3 + 10^{31}) \\ c_{32} &= -\dot{q_2}(0.0125q_3 + 0.001) \\ c_{33} &= 0 \end{aligned} \tag{4.2}$$

$$g(q) = \begin{bmatrix} 0.001961 cos(q_1) sin(q_2)(125.0q_3 + 11) \\ 0.001961 cos(q_2) sin(q_1)(125q_3 + 11) \\ 0.2452 sin(q_1) sin(q_2) \end{bmatrix} \tag{4.3}$$

$$\tau = D(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) \tag{4.4}$$

Spike Dynamics for Rolling

Appropriate spike should be chosen to perform kick for desired rotation. For example, back spike provides forward rolling. Therefore, single leg rolling dynamic are inspected. Determined leg extends to fulfill required rotation. F corresponds to applied force due to pressure based actuator.

$$P = mg(d+R)sin(\alpha)$$
$$K = \frac{1}{2}I\ddot{a}^2 + \frac{1}{2}m\ddot{a}^2(d+R)^2 + \frac{1}{2}m\ddot{d}^2$$
$$L = K - P = \frac{1}{2}I\ddot{a}^2 + \frac{1}{2}m\ddot{a}^2(d+R)^2 + \frac{1}{2}m\ddot{d}^2 - mg(d+R)sin(a)$$
$$\frac{\partial L}{\partial a} = -mg(d+R)cos(a) \quad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{a}}\right) = I\ddot{a} + m(\ddot{a}(d+R)^2 + 2\dot{a}\dot{d}(d+R))$$
$$\frac{\partial L}{\partial \dot{a}} = I\dot{a} + m\dot{a}(d+R)^2 \quad \frac{\partial L}{\partial d} = m\ddot{a}^2(d+R) - mgsin(a)$$
$$\frac{\partial L}{\partial \dot{d}} = m\dot{d} \quad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{d}}\right) = m\ddot{d}$$

$$I\ddot{a} + m(\ddot{a}(d+R)^2 + 2\dot{a}\dot{d}(d+R)) + mg(d+R)cos(a) = 0$$
$$m\ddot{d} - m\dot{a}^2(d+R) + mgsin(a) = F$$

$$\ddot{a} = \frac{-mg(d+R)cos(a) - 2m\dot{d}\dot{a}(d+R)}{I + m(d+R)^2}$$
$$\ddot{d} = \frac{F + m\dot{a}^2(d+R) - mgsin(a)}{m}$$

Figure 4.1: Rolling Scheme

Control Strategies

Stiffness Control

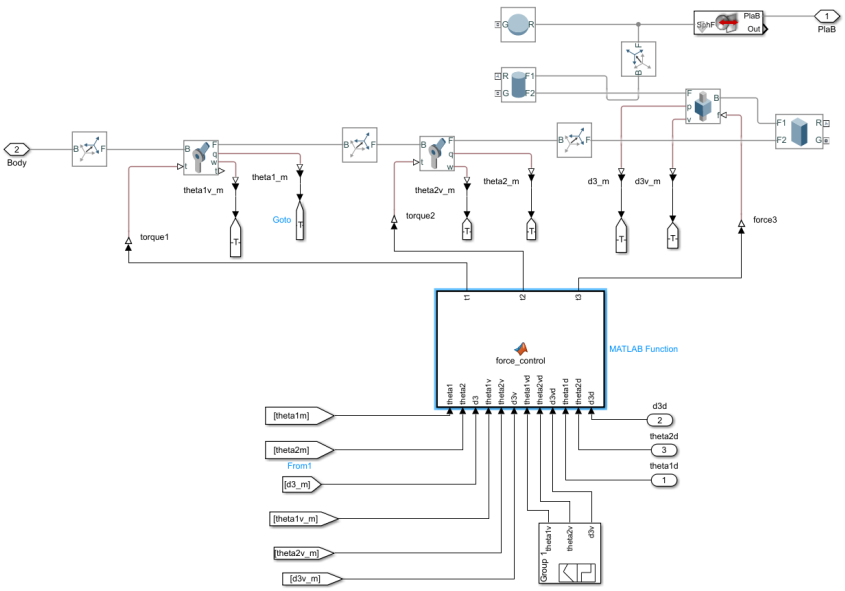


Figure 5.1: Simulink Implementation of Jacobian Force Control

Salisbury [2] method enables end-effector of manipulator to behave in desired cartesian stiffness. It controls cartesian stiffness by applying appropriate torques to joints. Control law is implemented in joint space with help of Jacobian. K_{px} in Equation 5.1 is 3×3 diagonal matrix that determines linear stiffness of manipulator. I aim to protect spikes from ruptures by choosing relatively small stiffness in y direction so that spikes comply with ground in y direction during stance phase.

$$\begin{aligned} F &= K_{px}\delta x \\ \delta x &= J(\theta)\delta\theta \\ F &= K_{px}J(\theta)\delta\theta \\ \tau &= J^T(\theta)F \\ \tau &= J^T(\theta)K_{px}J(\theta)\delta\theta \end{aligned} \tag{5.1}$$

$J^T(\theta)K_{px}J(\theta)$ term in Equation 5.1 expresses joint space stiffness matrix in similar manner to cartesian stiffness matrix K_{px}. Control torque satisfying desired stiffness is given in Equation 5.2.

$$\tau = J^T(\theta)K_{px}J(\theta)(\theta_d - \theta) + K_v(\theta_d - \theta) \tag{5.2}$$

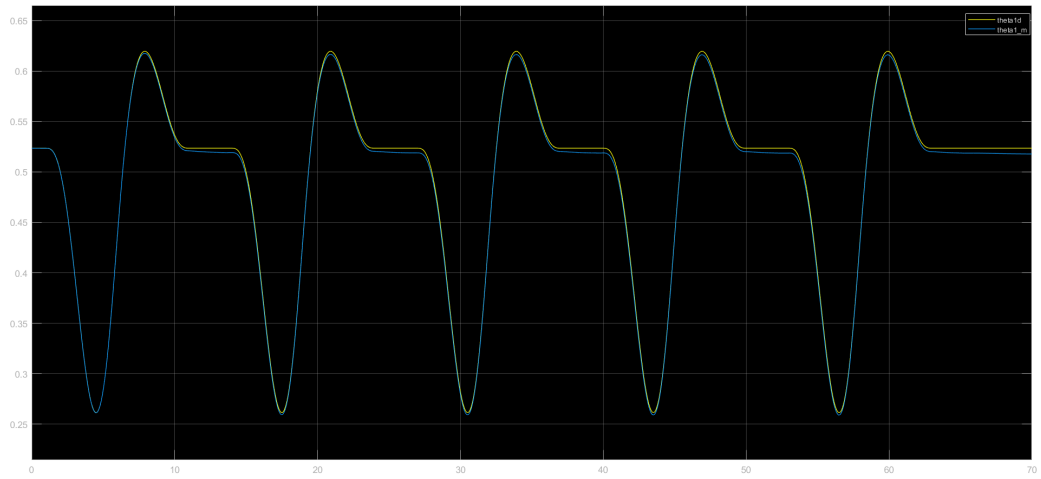


Figure 5.2: θ_1 measured and θ_1 desired comparison

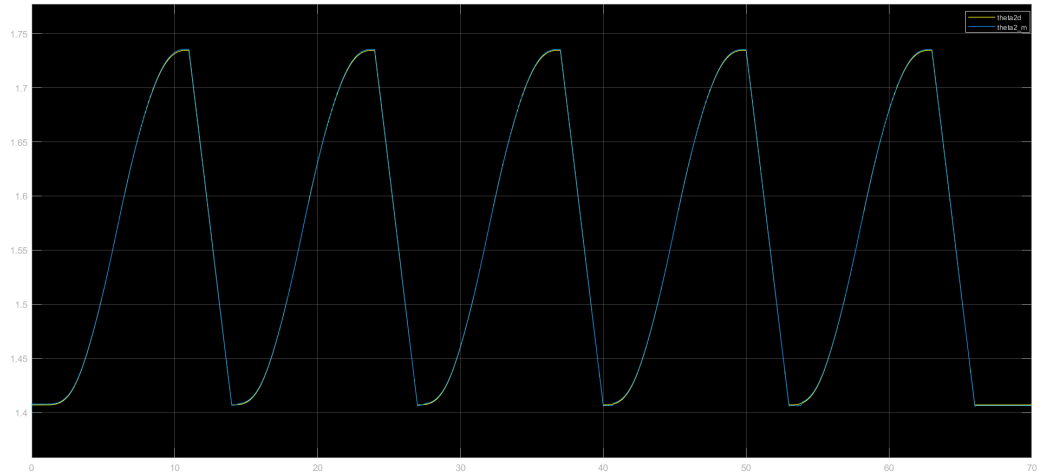


Figure 5.3: θ_2 measured and θ_2 desired comparison

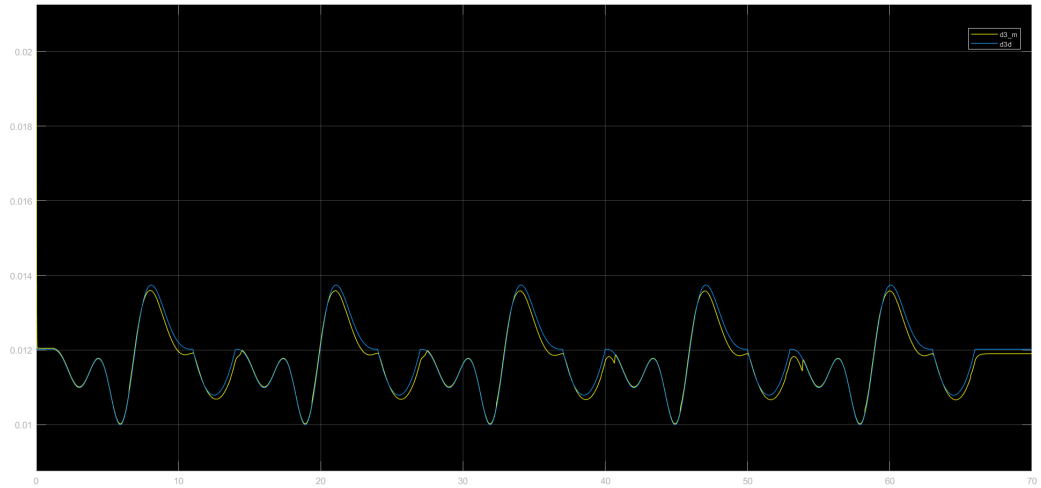


Figure 5.4: d_3 measured and d_3 desired comparison

PID Control

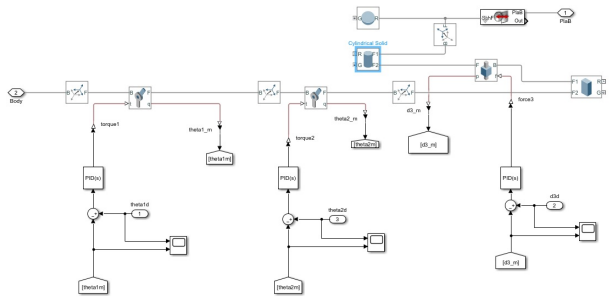


Figure 5.5: Simulink Implementation of PID Controller

Table 5.1: Gains of PID Controller.

Joint	P	I	D
θ_1	10	10	10
θ_2	10	10	10
d_3	10000	1000	100

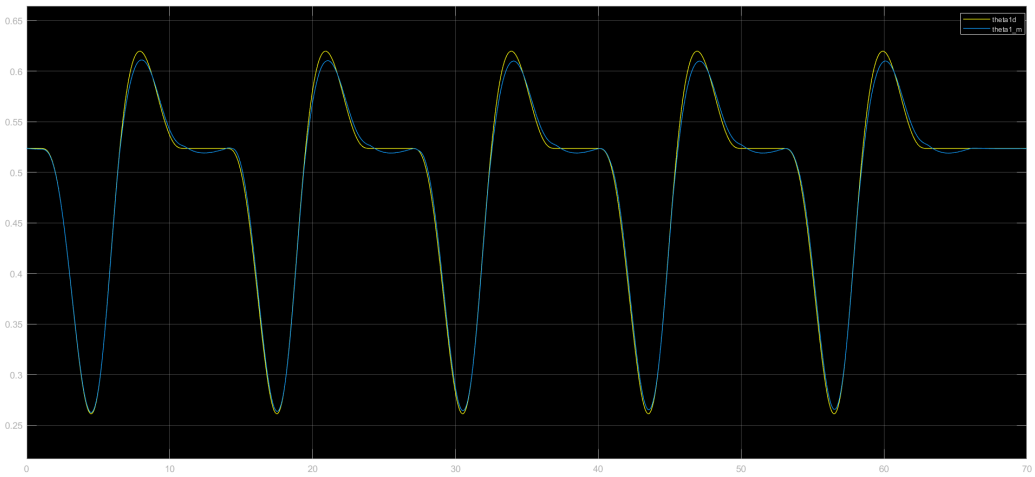


Figure 5.6: θ_1 measured and θ_1 desired comparison for PID Control.

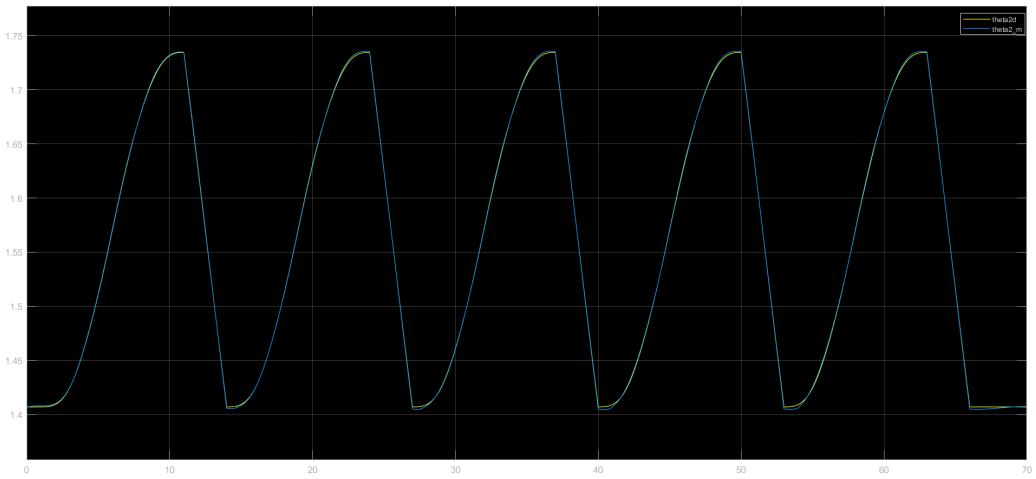


Figure 5.7: θ_2 measured and θ_2 desired comparison for PID Control.

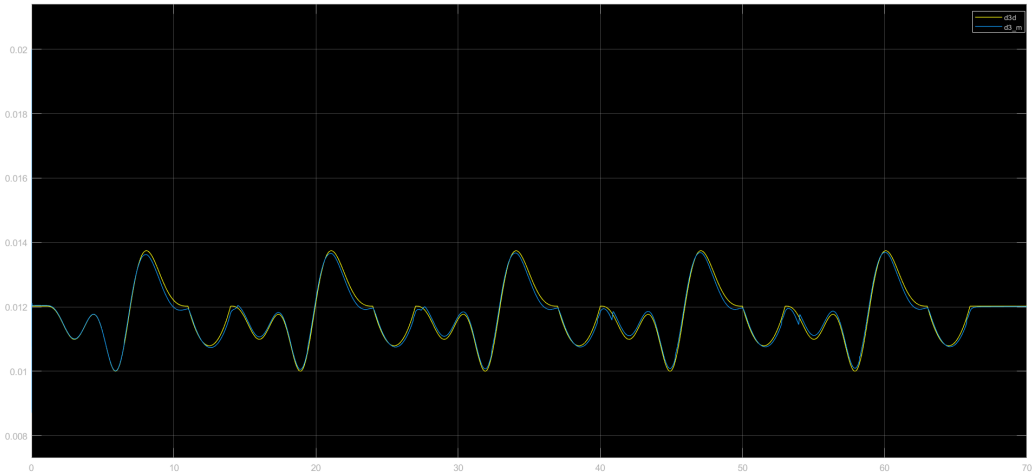


Figure 5.8: d_3 measured and d_3 desired comparison for PID Control.

PID controller aims to minimize error between desired joint angles and actual joint angles of actuator. Actual joint angles are measured by position sensors. After that, error is weighted by P gain called as proportional gain. This weighted difference is fed into actuator as torque input. I adjust proportional gain to obtain tolerable overshoot without comprising tracking speed. During control part, I wanted to tried two control scheme. I consider that force control technique provides compliance ability. At least, I hope it brings such ability. Since I possess joint space trajectory, I tried to found method which applies force control in joint space. I encountered with Salisbury's paper [2]. Based on his derivation, I implemented force control. I also applied PID control to compare with it as baseline. It empirically seems similar performance in terms of tracking. There are of course piece of compromise to comply with terrain in case of force control.

Conclusions

I encountered with a lot of challenges to implement this project and learnt a lot of things.

- I studied walking and rolling robot literature seriously. It almost corresponds to the biggest portion of time that I spent for project. I grasped walking and rolling phase and tried to make it sense. I read a lot of paper and thesis for walking and rolling robot design.
- Trajectories designed for operational space appears more rigorous than joint space trajectory. Since it brings full awareness of spike trajectory. As I learnt this fact, I began to research this type of trajectory. I found quintic trajectory that satisfies initial and final conditions and this match with my situation since I want smooth land-off to ground.
- Since trajectories are designed for operational space, it is required to convert them into joint space. Here, I applied derived inverse kinematic equation to obtain joint space trajectories.
- Dynamic equation seems intractable due to many equation so I learnt symbolic computation to get equations of motions. This takes my time but it worth to solve them in MatLab symbolic toolbox. I acquire ability of solve dynamic equations in software from this task.
- I had never experienced mechanical design tools before this project began. I choose simulink environment for mechanical stuffs. I almost design everything parametric so that I tuned the parameters such as inertia, masses, and length to construct realistic and practical mechanical structure for reasonable movement.

Appendix

Transfer and Stance Trajectory

```
% transfer phase trajectory
k=4e-2;
r=4e-2;
t0 = 0;
th = 5;
tf = 10;
foot_zi = -1.5e-2;
foot_zf = 1.5e-2;
foot_yi = 3*sqrt(3)*1e-2;
foot_yh = 4.2e-2;
foot_yf = 3*sqrt(3)*1e-2;
foot_x = 9e-2;

data_mtrx = [1 t0 t0^2 t0^3 t0^4 t0^5;0 1 2*t0 3*t0^2 4*t0^3 5*t0^4;
    0 0 2 6*t0 12*t0^2 20*t0^3;1 tf tf^2 tf^3 tf^4 tf^5;0 1 2*tf 3*tf^2 ...
    4*tf^3 5*tf^4;0 0 2 6*tf 12*tf^2 20*tf^3];
data_mtrxl = [1 t0 t0^2 t0^3 t0^4 t0^5;0 1 2*t0 3*t0^2 4*t0^3 5*t0^4;
    0 0 2 6*t0 12*t0^2 20*t0^3;1 th th^2 th^3 th^4 th^5;0 1 2*th 3*th^2 ...
    4*th^3 5*th^4;0 0 2 6*th 12*th^2 20*th^3];
data_mtrxs = [1 th th^2 th^3 th^4 th^5;0 1 2*th 3*th^2 4*th^3 5*th^4;
    0 0 2 6*th 12*th^2 20*th^3;1 tf tf^2 tf^3 tf^4 tf^5;0 1 2*tf 3*tf^2 ...
    4*tf^3 5*tf^4;0 0 2 6*tf 12*tf^2 20*tf^3];

A1 = data_mtrx \ [foot_zi;0;0;foot_zf;0;0];
Ayl = data_mtrxl \ [foot_yi;0;0;foot_yh;2e-2;0];
Ays = data_mtrxs \ [foot_yh;2e-2;0;foot_yf;0;0];
t = linspace(0,tf,500);
tl = t(1:250);
ts = t(251:500);
z1 = A1(1) + A1(2).*t + A1(3)*(t.^2) + A1(4)*(t.^3) + A1(5)*(t.^4) + A1(6)*(t.^5);
yl = Ayl(1) + Ayl(2).*tl + Ayl(3)*(tl.^2) + Ayl(4)*(tl.^3) + Ayl(5)*(tl.^4) + Ayl(6)*(tl.^5);
ys = Ays(1) + Ays(2).*ts + Ays(3)*(ts.^2) + Ays(4)*(ts.^3) + Ays(5)*(ts.^4) + Ays(6)*(ts.^5);


y = [yl ys];

theta1 = [];
theta2 = [];
d3 = [];


for i=1:500
    [theta_1,theta_2,d_3] = inverse_kinematic(foot_x,y(i),z1(i),r,k);
    theta1 = [theta1 theta_1];
    theta2 = [theta2 theta_2];
    d3 = [d3 d_3];
end




% Support Phase Trajectory

t_sup = linspace(0,3,500);
z_sup = foot_zf - 1e-2.*t_sup;

for i=1:500
    [theta_1,theta_2,d_3] = inverse_kinematic(foot_x,foot_yf,z_sup(i),r,k);

    theta1 = [theta1 theta_1];
    theta2 = [theta2 theta_2];
    d3 = [d3 d_3];
end


t = [t t(end)+t_sup]
theta1v = gradient(theta1,0.02);
theta2v = gradient(theta2,0.02);
```

```
d3v = gradient(d3,0.02);
theta1ac = gradient(theta1v,0.02);
theta2ac = gradient(theta2v,0.02);
d3ac = gradient(d3v,0.02)

t = [t t+13 t+26 t+39]
theta1 = [theta1 theta1 theta1 theta1];
theta2 = [theta2 theta2 theta2 theta2];
d3 = [d3 d3 d3 d3];
theta1v = [theta1v theta1v theta1v theta1v];
theta2v = [theta2v theta2v theta2v theta2v];
d3v = [d3v d3v d3v d3v];
theta1ac = [theta1ac theta1ac theta1ac theta1ac];
theta2ac = [theta2ac theta2ac theta2ac theta2ac];
d3ac = [d3ac d3ac d3ac d3ac];

figure;
subplot(3,3,1);
plot(t,theta1)
title("Joint 1 Angle")
subplot(3,3,2)
plot(t,theta1v)
title("Joint 1 Angular Velocity")
subplot(3,3,3)
plot(t,theta1ac)
title("Joint 1 Angular Accelaration")
subplot(3,3,4)
plot(t,theta2)
title("Joint 2 Angle")
subplot(3,3,5)
plot(t,theta2v)
title("Joint 2 Angular Velocity")
subplot(3,3,6)
plot(t,theta2ac)
title("Joint 2 Angular Acceleration")
subplot(3,3,7)
plot(t,d3)
title("Joint 3 extension")
subplot(3,3,8)
plot(t,d3v)
title("Joint 3 Extension Velocity")
subplot(3,3,9)
plot(t,d3ac)
title("Joint 3 Extension Acceleration")
```

Equation of motions MATLAB Script

```
syms q1 q2 q3 q1dot q2dot q3dot q1ddot q2ddot q3ddot;
m_rec = 1e-2;
m_cyl = 5e-2;
c_rec = 002*[cos(q1)*sin(q2);sin(q1)*sin(q2);-cos(q2)];
c_cyl = (0.5*q3 + 0.04).*[cos(q1)*sin(q2);sin(q1)*sin(q2);-cos(q2)];
I_cyl = 1e-15*eye(3,3);
I_rec = 1e-15*eye(3,3);
Jv_rec = 0.02*[-sin(q1)*sin(q2) cos(q2)*cos(q1) 0;
               cos(q1)*sin(q2)  cos(q2)*sin(q1) 0;
               0                sin(q2)         0];
r = 0.04 + 0.5*q3;
Jv_cyl = [-r*sin(q1)*sin(q2) r*cos(q2)*cos(q1) cos(q1)*sin(q2);
              r*cos(q1)*sin(q2) r*cos(q2)*sin(q1) sin(q1)*sin(q2);
              0                 r*sin(q2)         -cos(q2)];
Jw_rec = [0 sin(q1) 0; 0 -cos(q1) 0; 1 0 0];
Jw_cyl = [0 sin(q1) 0; 0 -cos(q1) 0; 1 0 0];
R_rec = [cos(q2)*cos(q1) sin(q1) cos(q1)*sin(q2);
         sin(q1)*cos(q2) -cos(q1) sin(q1)*sin(q2);
         sin(q2)          0       -cos(q2)];
R_cyl = R_rec;
%%%%%%%% D-MATRIX COMPUTATION %%%%%%%%%%
D= m_rec*transpose(Jv_rec)*Jv_rec + ...
    transpose(Jw_rec)*R_rec*I_rec*transpose(R_rec)*Jw_rec + ...
    m_cyl*transpose(Jv_cyl)*Jv_cyl + transpose(Jw_cyl)*R_cyl*I_cyl*transpose(R_cyl)*Jw_cyl;
D = vpa(simplify(D),4);
```

```
disp('D =');
disp(D)
d11 = D(1,1);
d12 = D(1,2);
d13 = D(1,3);
d21 = D(2,1);
d22 = D(2,2);
d23 = D(2,3);
d31 = D(3,1);
d32 = D(3,2);
d33 = D(3,3);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%% CHRISTOFELL SYMBOL COMPUTATION %%%%
% i = 1, j = 1, k=1
c111 = 0.5*(diff(d11,'q1') + diff(d11,'q1') - diff(d11,'q1'));
% i = 2, j = 1, k=1
c211 = 0.5*(diff(d11,'q2')+diff(d12,'q1') - diff(d21,'q1'));
% i=  3, j=  1, k=1
c311 = 0.5*(diff(d11,'q3')+diff(d13,'q1')- diff(d31,'q1'));
% i= 1, j=2, k=1
c121 = c211;
% i= 2, j=2, k=1
c221 = 0.5*(diff(d12,'q2')+diff(d12,'q2') - diff(d22,'q1'));
% i= 3, j=2, k= 1
c321 = 0.5*(diff(d12,'q3') + diff(d13,'q2') - diff(d32,'q1'));
% i= 1, j=3, k=1
c131 = c311;
% i= 2, j=3, k=1
c231 = c321;
% i= 3, j=3, k=1
c331 = 0.5*(diff(d13,'q3')+diff(d13,'q3')-diff(d33,'q1'));
% i= 1, j=1, k=2
c112 = 0.5*(diff(d21,'q1')+diff(d21,'q1')-diff(d11,'q2'));
% i= 2, j=1, k=2
c212 = 0.5*(diff(d21,'q2')+diff(d22,'q1')-diff(d21,'q2'));
% i= 3, j=1, k=2
c312 = 0.5*(diff(d21,'q3')+diff(d23,'q1')-diff(d31,'q2'));
% i= 1, j=2, k=2
c122 = c212;
% i=2, j=2, k=2
c222 = 0.5*(diff(d22,'q2')+diff(d22,'q2')-diff(d22,'q2'));
% i=3, j=2, k=2
c322 = 0.5*(diff(d22,'q3')+diff(d23,'q2')-diff(d32,'q2'));
% i=1, j=3, k=2
c132 = c312;
% i=2, j=3, k=2
c232 = c322;
% i=3, j=3, k=2
c332 = 0.5*(diff(d23,'q3')+diff(d23,'q3')-diff(d33,'q2'));
% i=1, j=1, k=3
c113 = 0.5*(diff(d31,'q1')+diff(d31,'q1')-diff(d11,'q3'));
% i=2, j=1, k=3
c213 = 0.5*(diff(d31,'q2')+diff(d32,'q1')-diff(d21,'q3'));
% i=3, j=1, k=3
c313 = 0.5*(diff(d31,'q3')+diff(d33,'q1')-diff(d31,'q3'));
% i=1, j=2, k=3
c123 = c213;
% i=2, j=2, k=3
c223 = 0.5*(diff(d32,'q2')+diff(d32,'q2')-diff(d22,'q3'));
% i=3, j=2, k=3
c323 = 0.5*(diff(d32,'q3')+diff(d33,'q2')-diff(d32,'q3'));
% i=1, j=3, k=3
c133 = c313;
% i=2, j=3, k=3
c233 = c323;
% i=3, j=3, k=3
c333 = 0.5*(diff(d33,'q3')+diff(d33,'q3')-diff(d33,'q3'));

C = [c111*q1dot+c211*q2dot+c311*q3dot c121*q1dot+c221*q2dot+c321*q3dot c131*q1dot+c231*q2dot+
    c331*q3dot; ...
     c112*q1dot+c212*q2dot+c312*q3dot c122*q1dot+c222*q2dot+c322*q3dot c132*q1dot+c232*q2dot+
         c332*q3dot; ...
```

```
    c113*q1dot+c213*q2dot+c313*q3dot c123*q1dot+c223*q2dot+c323*q3dot c133*q1dot+c233*q2dot+
        c333*q3dot];
C= vpa(simplify(C),4);
disp('C =');
disp(C);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%% GRAVITY TERM COMPUTATION %%%%%%%%%%%%%%

g= [0;10;0];

P = m_rec*g'*c_rec + m_cyl*g'*c_cyl;
g1 = diff(P,'q1');
g2 = diff(P,'q2');
g3 = diff(P,'q3');
g = [g1;g2;g3];
g = vpa(simplify(g),4);
disp('g=');
disp(g);

Tau = D*[q1ddot;q2ddot;q3ddot] + C*[q1dot;q2dot;q3dot] + g;
Tau = vpa(simplify(Tau),4);
disp('Tau=');
disp(Tau);
```

References

[1] Jong-Kil Lee. "Instantaneous kinematics, motion planning and mechanical efficiency of a quadrupedal walking machine". PhD thesis. University of Illinois at Chicago, 1990.

[2] J Kenneth Salisbury. "Active stiffness control of a manipulator in cartesian coordinates". In: *1980 19th IEEE conference on decision and control including the symposium on adaptive processes*. IEEE. 1980, pp. 95–100.

[3] Mark W Spong, Seth Hutchinson, Mathukumalli Vidyasagar, et al. *Robot modeling and control*. Vol. 3. Wiley New York, 2006.