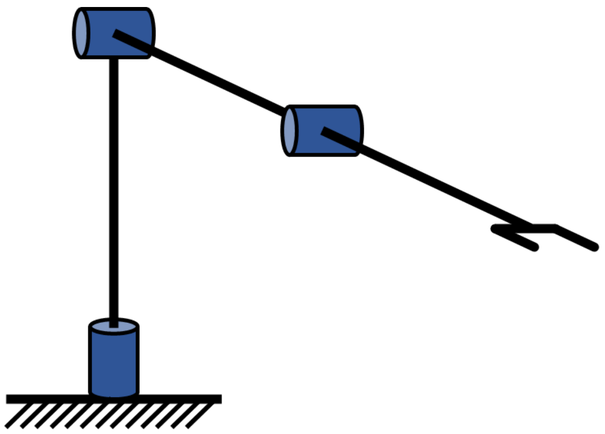
DE3 Applied Robotics **Coursework 1 - MODEL**

Module Leader: Dr Petar Kormushev Robot Kinematics, Dynamics, and Control

1 Introduction

Accurate robot **modelling** is arguably the most important part of robot design and control. A precise robot model should describe well both the kinematics and dynamics properties of the robot. As you have learned in Term 1, kinematics relates the position of the robot in **task space** and **joint space**. In this coursework, we begin by looking at the kinematics of a robot arm with three revolute joints:



Three files will be used in this coursework, `kinematics.py`, `robot_model_gazebo.xacro`, and `controller_settings.yaml`. All three files can be found inside the Virtual Machine in the coursework 1 folder: '/home/de3robotics/Desktop/ DE3Robotics/src/coursework_1'.

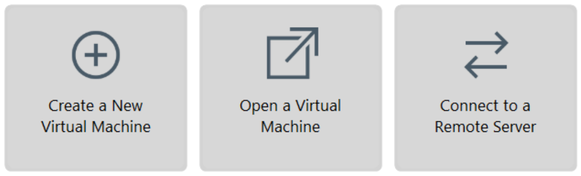
File	Description	Relative Path
`kinematics.py`	Computes the forward and inverse kinematics of the robot arm, sends desired positions to the simulator.	src/kinematics.py
`robot_model_gazebo.xacro`	Defines the model of the robot.	urdf/robot_model_gazebo.xacro
`controller_settings.yaml`	Defines the control parameters of the position controller of the robot.	config/controller_settings.yaml

2 Getting Started

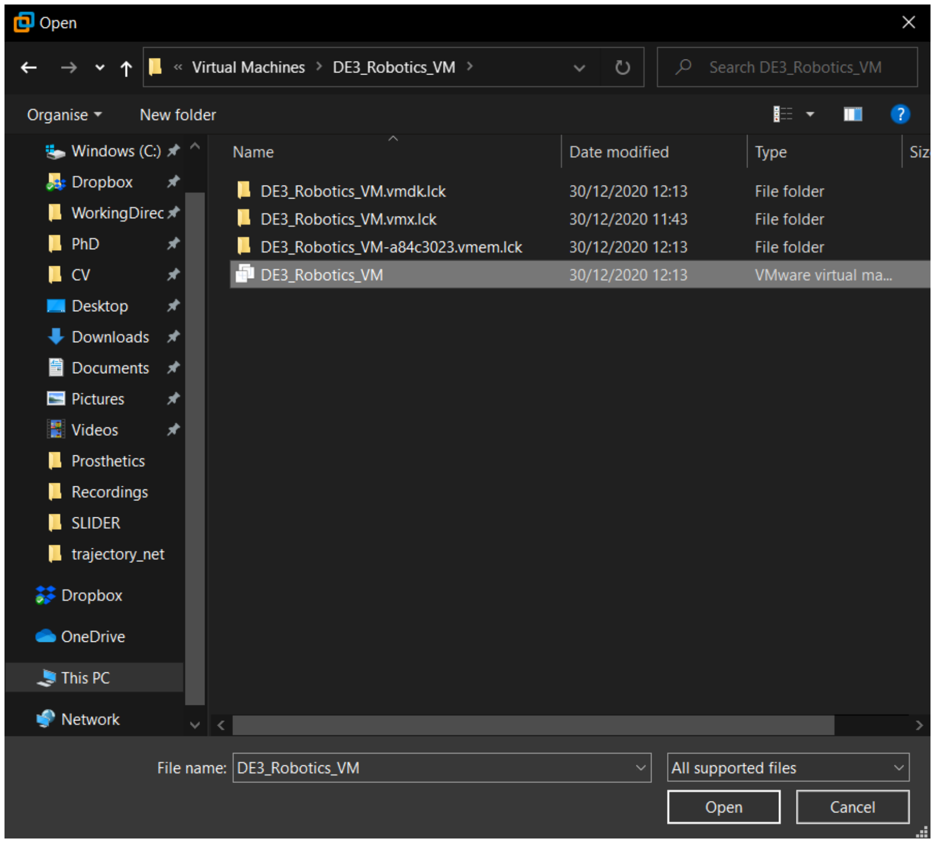
2.1 Running the Virtual Machine

Download the virtual machine `DE3_Applied_Robotics_VM.zip` file using the link in Blackboard. Warning, it is a very large file (> 4 GB compressed file, and around 12 GB once uncompressed), so make sure you have enough free space on your computer drive!

When it has downloaded, unzip and extract the virtual machine folder to the directory you would like to keep it in. Preferably, use an SSD drive if you have one in your computer. Then, open VMware Player (or VMware Fusion or VMware Workstation) and select 'Open a Virtual Machine':



Navigate to the virtual machine directory and select the virtual machine file (file type 'VMware virtual machine configuration'):

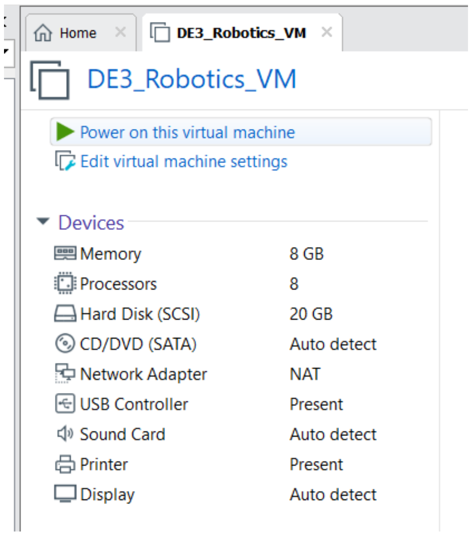


Name	Date modified	Type
DE3_Robotics_VM.vmdk.lck	30/12/2020 12:13	File folder
DE3_Robotics_VM.vmx.lck	30/12/2020 11:43	File folder
DE3_Robotics_VM-a84c3023.vmem.lck	30/12/2020 12:13	File folder
DE3_Robotics_VM	30/12/2020 12:13	VMware virtual ma...

File name: DE3_Robotics_VM

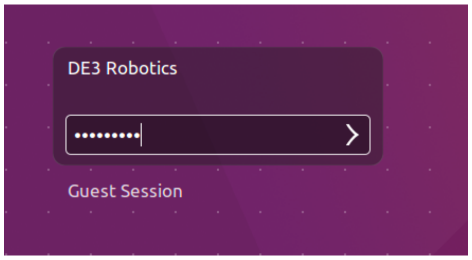
All supported files

The virtual machine should now be loaded inside VMware. Next, turn on the virtual machine by clicking 'Power on this virtual machine':



Memory	8 GB
Processors	8
Hard Disk (SCSI)	20 GB
CD/DVD (SATA)	Auto detect
Network Adapter	NAT
USB Controller	Present
Sound Card	Auto detect
Printer	Present
Display	Auto detect

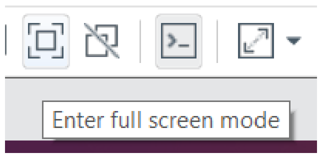
The virtual machine will power on, and you will reach the login screen.



Log in to user 'DE3 Robotics' using the password '`deniro123`'.

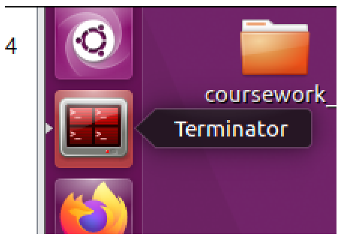
Hint: Adjusting the view of the virtual machine

The performance of the virtual machine will improve if you run it in full screen mode. You can do this by clicking the 'Enter full screen mode' button:



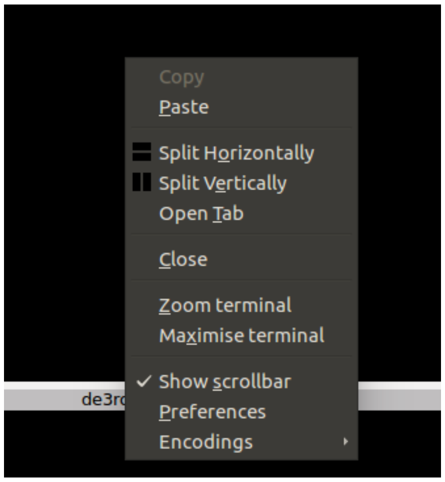
2.2 Loading Gazebo Simulator

Open Terminator from the sidebar:



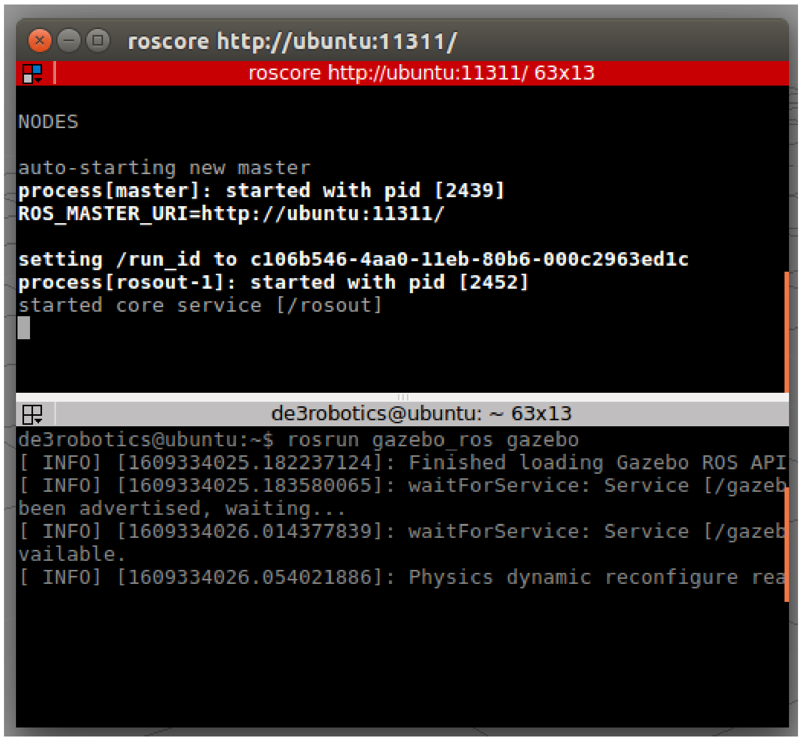
Hint: Splitting Terminator into multiple panes

We will run multiple instances of the terminal at several points through the coursework. It will be useful to display them all in one convenient way. To do this, you can right click on Terminator and click 'Split Horizontally' or 'Split Vertically':



Alternatively, you can use keyboard shortcuts '`Ctrl+Shift+o`' to split horizontally, and '`Ctrl+Shift+e`' to split vertically.

In one terminal, start roscore by typing '`roscore`' and pressing enter. In a second terminal, start Gazebo simulator by typing '`rosrun gazebo_ros gazebo`' and pressing enter:


roscore http://ubuntu:11311/
roscore http://ubuntu:11311/ 63x13
de3robotics@ubuntu: ~ 63x13

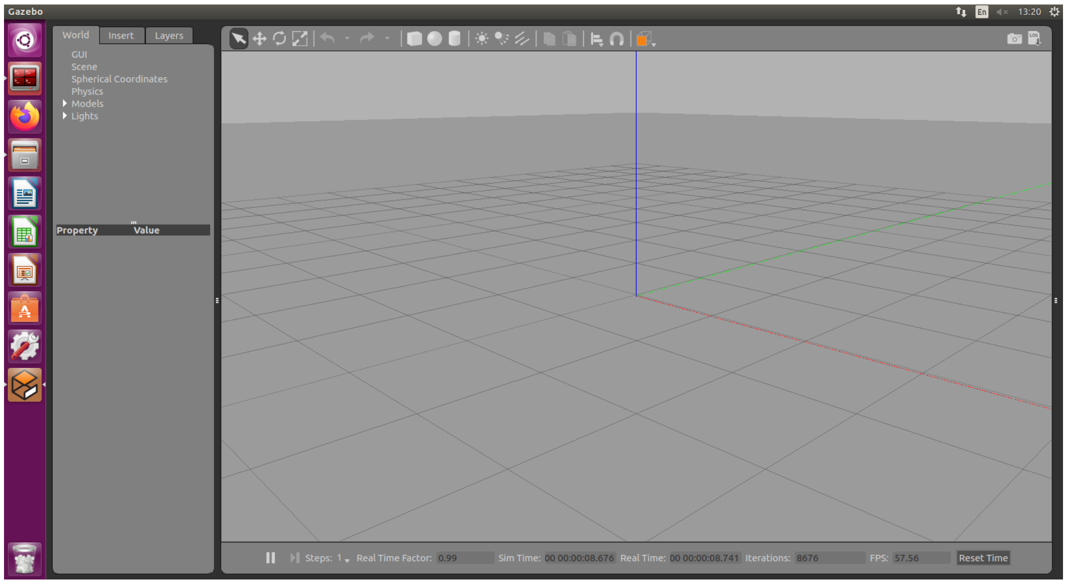
```
NODES

auto-starting new master
process[master]: started with pid [2439]
ROS_MASTER_URI=http://ubuntu:11311/

setting /run_id to c106b546-4aa0-11eb-80b6-000c2963ed1c
process[rosout-1]: started with pid [2452]
started core service [/rosout]
```

```
de3robotics@ubuntu:~$ rosrun gazebo_ros gazebo
[ INFO] [1609334025.182237124]: Finished loading Gazebo ROS API
[ INFO] [1609334025.183580065]: waitForService: Service [/gazeb
been advertised, waiting...
[ INFO] [1609334026.014377839]: waitForService: Service [/gazeb
vailable.
[ INFO] [1609334026.054021886]: Physics dynamic reconfigure rea
```

Gazebo should start, and an empty world should load:



To close Gazebo, press '`Ctrl+c`' in the terminal that is running Gazebo.

Hint: Making sure Gazebo is closed

Sometimes, problems can occur when multiple instances of Gazebo are running, or haven't shut down properly. To make sure that Gazebo is properly closed, run the following command from terminal:

```
killall gzserver gzclient
```

If Gazebo is completely closed, you should get the following response:

```
gzserver: no process found
gzclient: no process found
```

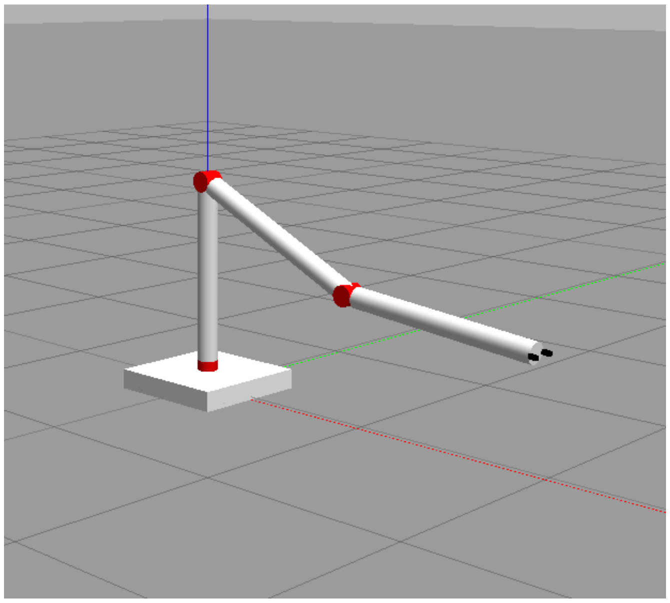
Now, we can add the robot arm to the simulation! In a new terminal window, run the following commands:

```
cd Desktop/DE3Robotics
source devel/setup.bash
```

Then run this final command to add the robot arm to the simulation:

```
roslaunch coursework_1 coursework_1.launch
```

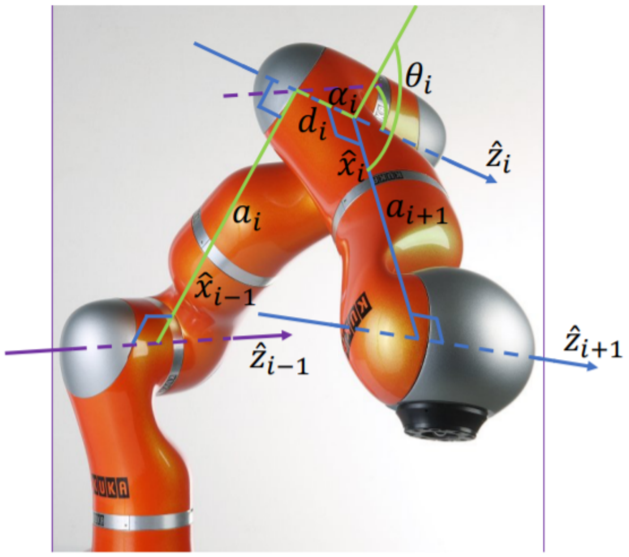
You should see the robot arm in Gazebo:



3 Forward Kinematics

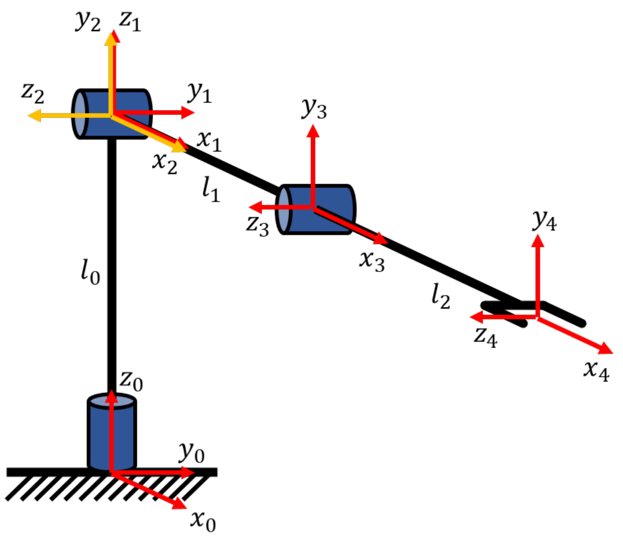
Forward kinematics converts coordinates from **joint space to task space**. We would like to calculate the forward kinematics of our robot arm, which has three revolute joints.

We will use the Devanit-Hartenberg convention to compute the forward kinematics of the robot arm. Four parameters (the D-H parameters) define each joint of a robot; d, θ, a, α.



Task A: Compute the initial D-H table of the robot arm

Based on the diagram below, calculate the D-H parameters of each link of the robot arm.



Inside the file `kinematics.py`, find the line where `self.DH_tab` is created. Complete the code to create the initial D-H table of the robot arm. The length of link `n` is stored in `self.links[n]`. Remember that we use radians, not degrees!

As shown in the lecture notes, the D-H parameters can be used to compute the transformation matrix from joint $i-1$ to joint i:

$$^{i-1}T_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_i \\ \sin\theta_i \cos\alpha_i & \cos\theta_i \cos\alpha_i & -\sin\alpha_i & -\sin\alpha_i d_i \\ \sin\theta_i \sin\alpha_i & \cos\theta_i \sin\alpha_i & \cos\alpha_i & \cos\alpha_i d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

Task B: Complete the code for the D-H matrix

Inside the file `kinematics.py`, find the function `DH_matrix`. Complete the code to compute the transformation matrix from frame $i-1$ to frame i, given the D-H parameters `d`, `theta`, `a`, `alpha`, in the function.

To calculate the pose of the end-effector of the robot in the base frame of the robot, ^0T_N, the transformation matrices of each joint can be multiplied together:

$$^0T_N =^0 T_1\,^1T_2 \dots ^{N-1}T_N \quad (2)$$

This can be done iteratively, in a loop. Each iteration, the following calculation is performed:

$$^0T_i =^0 T_{i-1}\,^{i-1}T_i \quad (3)$$

Task C: Complete the code to compute forwards kinematics

Inside the file `kinematics.py`, find the function `getFK`. We will use the numpy (a Python package for numerical operations) function `np.matmul` to perform matrix multiplication:

$$AB = \texttt{np.matmul(A, B)}$$

Complete the code inside the for loop to perform the calculation presented in equation (3).

To test your code, open a new terminal and run the following commands:

```
cd Desktop/DE3Robotics/src/coursework_1/src
python3 kinematics.py fk
```

The Python script will test your forward kinematics code at several joint positions. If everything runs correctly, you should see the following message:
"Forward Kinematics calculations correct, well done!".

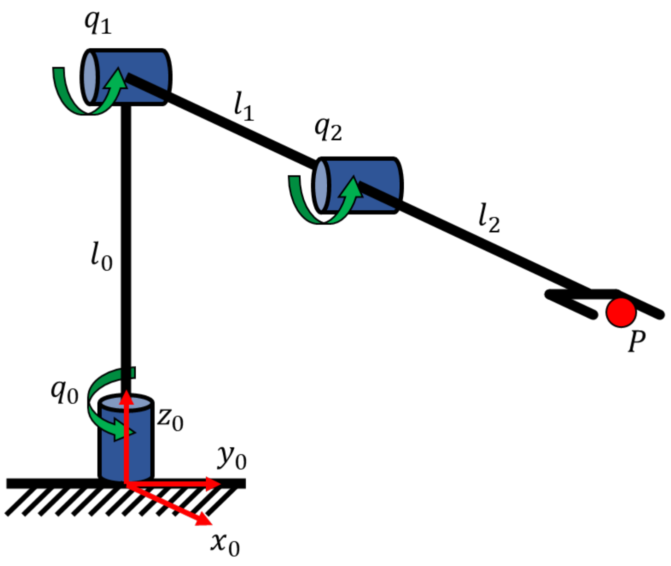
If the calculations are incorrect, you will see the following message:
"Forward Kinematics calculations incorrect, exiting."

We have now completed the forward kinematic model of the robot!

4 Inverse Kinematics

Inverse kinematics converts coordinates from **task space to joint space**. We would like to calculate the geometric inverse kinematics of our robot arm, which has three revolute joints.

The diagram below shows the home position of the robot arm:



As seen in its home configuration, link l_0 points along z_0, and links l_1 and l_2 point along y_0. This means that the joint angles q_0, q_1, and q_2 are measured relative to this home configuration.

From the forward kinematic model, we see that the coordinates of the end-effector in the world frame are:

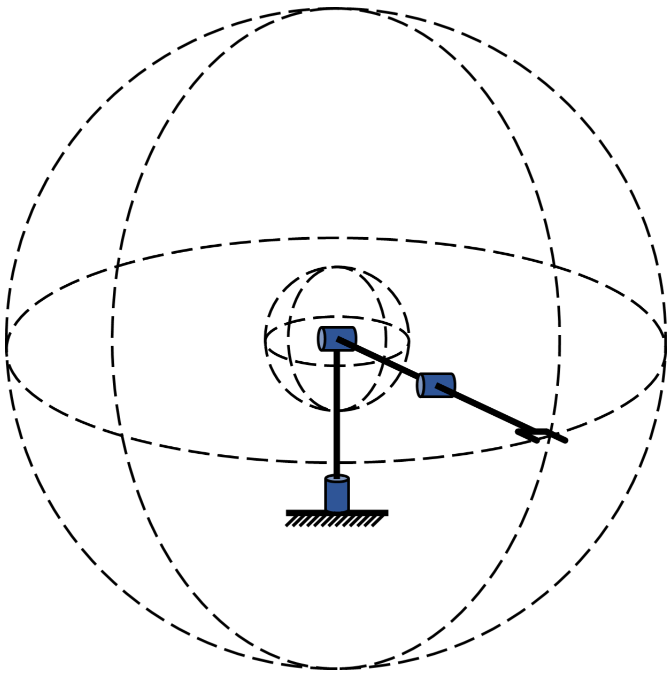
$$x_P = \big(l_1\cos(q_1) + l_2\cos(q_1+q_2)\big)\cos(q_0) \tag{4}$$

$$y_P = \big(l_1\cos(q_1) + l_2\cos(q_1+q_2)\big)\sin(q_0) \tag{5}$$

$$z_P = l_0 + \big(l_1 \sin(q_1) + l_2 \sin(q_1 + q_2)\big) \tag{6}$$

Before we try to compute the inverse kinematics of the robot arm, we should first be able to check if a given point is inside its reachable workspace.

The workspace of the robot arm is a spherical shell centred on the end of its first link:



A point (in the reference frame of the base), $P = (x_P,\, y_P,\, z_P)$, inside the larger sphere satisfies the following equation:

$$x_P^2 + y_P^2 + (z_P - l_0)^2 \leq (l_1 + l_2)^2 \tag{7}$$

Similarly, a point, $P = (x_P,\, y_P,\, z_P)$, outside the smaller sphere satisfies the following equation:

$$x_P^2 + y_P^2 + (z_P - l_0)^2 \geq (l_1 - l_2)^2 \tag{8}$$

We will refer to $val = x_P^2 + y_P^2 + (z_P - l_0)^2$, $rmax = (l_1 + l_2)$, and $rmin = (l_1 - l_2)$. A point inside the workspace of the robot arm satisfies **both** equations (8) and (7).

Task D: Checking if a point is in the workspace of the robot arm

Inside the file `kinematics.py`, find the function `checkInWS`. Complete the code to calculate `rmax`, `rmin`, and `val`.
Note: to perform an exponent in Python, use the numpy function `np.power`:

$$x^2 = \texttt{np.power(x, 2)}$$

To test your code, open a new terminal and run the following commands:

```
cd Desktop/DE3Robotics/src/coursework_1/src
python3 kinematics.py ws
```

The Python script will test your work space code at several task space positions. If everything runs correctly, you should see the following message:

"Workspace calculations correct, well done!".

If the calculations are incorrect, you will see the following message:
"Workspace calculations incorrect, exiting."

Now that we have the ability to check if a desired position is inside the workspace of the robot arm, we can calculate geometric inverse kinematics to find the joint angles required for the robot arm to reach the desired position.

Calculating q_0 is fairly straight forward:

$$\frac{y_P}{x_P} = \frac{\big(l_1\cos(q_1) + l_2\cos(q_1+q_2)\big)\sin(q_0)}{\big(l_1\cos(q_1) + l_2\cos(q_1+q_2)\big)\cos(q_0)}$$

$$\frac{y_P}{x_P} = \frac{\sin(q_0)}{\cos(q_0)}$$

$$\frac{y_P}{x_P} = \tan(q_0)$$

$$q_0 = \text{arctan2}(y_P,\, x_P) \tag{9}$$

Task E: Calculating inverse kinematics

Find expressions to calculate q_1 and q_2 of the robot arm.

Hint: Calculating inverse kinematics

The forward kinematic equations can be simplified:

$$\sqrt{x_P^2 + y_P^2} = l_1\cos(q_1) + l_2\cos(q_1+q_2)$$

$$z_P - l_0 = l_1\sin(q_1) + l_2\sin(q_1+q_2)$$

These equations look a lot like the equations for a planar two-link manipulator!

Task F: Calculating inverse kinematics in Python

You will notice that there are two solutions to the inverse kinematics. We will call these solutions $\boldsymbol{q_a} = [q_{a,0},\, q_{a,1},\, q_{a,2}]^T$ and $\boldsymbol{q_b} = [q_{b,0},\, q_{b,1},\, q_{b,2}]^T$.
Inside the file `kinematics.py`, find the function `getIK`. Complete the code to calculate the 3×1 arrays `q_a` and `q_b`.

To test your code, open a new terminal and run the following commands:

```
cd Desktop/DE3Robotics/src/coursework_1/src
python3 kinematics.py ik
```

The Python script will test your inverse kinematics code at several task space positions. If everything runs correctly, you should see the following message:
"Inverse Kinematics calculations correct, well done!".

If the calculations are incorrect, you will see the following message:

”Inverse Kinematics calculations incorrect, exiting.”

We have now computed the inverse kinematics of the robot arm!

5 Differential Kinematics

Now that we can relate the robot arm’s position in joint space and task space, we can look at its **velocity** in joint space and task space. This is known as differential kinematics. Starting by differentiating the forward kinematic equations given in equations (4-6):

$$\dot{x}_P = -\big(l_1\cos(q_1) + l_2\cos(q_1+q_2)\big)\sin(q_0)\dot{q}_0 - \big(\dot{q}_1 l_1 \sin(q_1) + (\dot{q}_1+\dot{q}_2)l_2\sin(q_1+q_2)\big)\cos(q_0) \tag{10}$$

$$\dot{y}_P = \big(l_1\cos(q_1) + l_2\cos(q_1+q_2)\big)\cos(q_0)\dot{q}_0 - \big(\dot{q}_1 l_1 \sin(q_1) + (\dot{q}_1+\dot{q}_2)l_2\sin(q_1+q_2)\big)\sin(q_0) \tag{11}$$

$$\dot{z}_P = \big(\dot{q}_1 l_1\cos(q_1) + (\dot{q}_1+\dot{q}_2)l_2\cos(q_1+q_2)\big) \tag{12}$$

In robotics, the Jacobian helps us calculate the task space velocity from joint space velocity:

$$\dot{\boldsymbol{x}} = \mathrm{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{13}$$

The Jacobian can be found by writing equations (10-12) in matrix form:

$$\begin{bmatrix}\dot{x}_P\\ \dot{y}_P\\ \dot{z}_P\end{bmatrix} = \begin{bmatrix}J_{1,1} & J_{1,2} & J_{1,3}\\ J_{2,1} & J_{2,2} & J_{2,3}\\ J_{3,1} & J_{3,2} & J_{3,3}\end{bmatrix}\begin{bmatrix}\dot{q}_0\\ \dot{q}_1\\ \dot{q}_2\end{bmatrix} \tag{14}$$

Task G: Computing the Jacobian [Only for groups]

Based on equations (10-12), calculate the Jacobian matrix of the robot arm.
Inside the file `kinematics.py`, find the line where `self.Jacobian` is created. Complete the code to calculate the Jacboian.

To test your code, open a new terminal and run the following commands:

```
cd Desktop/DE3Robotics/src/coursework_1/src
python3 kinematics.py dk
```

The Python script will test your differential kinematics code at several task space positions. If everything runs correctly, you should see the following message:
”Differential Kinematics calculations correct, well done!”.

If the calculations are incorrect, you will see the following message:
”Differential Kinematics calculations incorrect, exiting.”

We have now computed the differential kinematics of the robot arm!
The robot is now ready to move around in the simulation!

6 Robot Control

For the robot to follow a desired trajectory, it must be controlled. The robot arm uses a simple position controller to make sure it reaches desired joint positions. The position controller is similar to the controllers seen in lectures, except the dynamics of the robot are not considered.

The position controller is a proportional-integral-derivative controller. The torque, τ_i, on joint i is equal to

$$\tau_i = K_p(q_{i,\,d} - q_i) + K_d(\dot{q}_{i,\,d} - \dot{q}_i) + K_i\int_0^t (q_{i,\,d} - q_i)dt \tag{15}$$

Where $q_{i,d}$ and $\dot{q}_{i,d}$ are the desired position and velocity of joint i, respectively.
To move the robot arm with these gains, open a new terminal and run the following commands:

```
cd Desktop/DE3Robotics/src/coursework_1/src
python3 kinematics.py full
```

Task H: Tuning Controller Gains

Inside the file `controller_settings.yaml` you will find the gains for the proportional, integral, and derivative controller on each joint. Tune these gains until the robot arm is able to follow its trajectory without error - judge this visually, this is not an exact science!
Explain your method for tuning these gains in the report, and provide the final gains of your controller.

The robot arm should move around as expected, with (almost) zero error when it reaches each position!
If everything runs correctly, you should see a position error of `[0, 0, 0]` for steps 1, 2, and 4. Step 3 is outside the workspace of the robot arm, so a position error of `[-3, 1, -2]` is expected.
Now, we will look at what happens when the properties of the robot arm change.

Task I: Adapting the Robot Arm [Only for groups]

Inside the file `robot_model_gazebo.xacro` you will find the simulator's description of the robot arm. This contains each joint and link of the robot arm. At the top of the file, important parameters are defined. These include the lengths of each link, and the mass of the end effector.
Now imagine the robot arm picks up a heavy object, of mass 30 kg. Find the parameter that defines the end effector mass, and change its value to equal 30.
Now run your Python code again, and report how the behaviour of the robot arm has changed.

Task J: Adapting Controller Gains [Only for groups]

Repeat the gain tuning process you followed before for the modified robot parameters.
Explain your method for tuning these gains in the report, and provide the final gains of your controller.

Congratulations! You have now completed the first coursework!