ME 381

Experiment 7: Kinematics of a five-bar parallel planar manipulator

Aim:

Perform forward and inverse kinematic analyses of a five-bar parallel manipulator. Introduction to trajectory planning of the robot end-effector.

Preparation

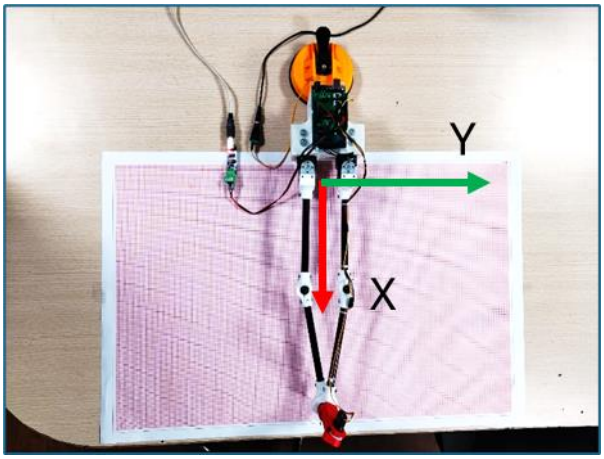
Software setup:

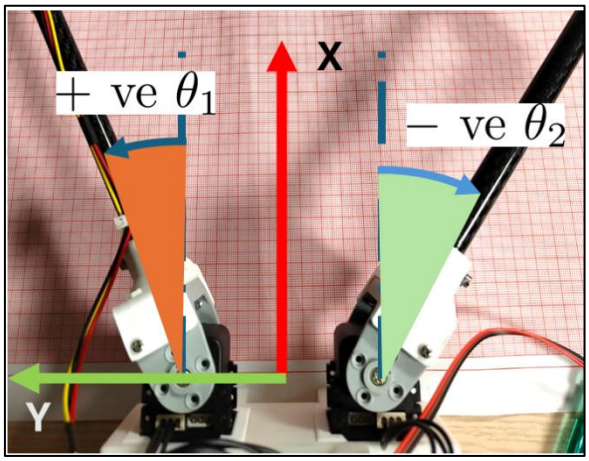
Following Python libraries are necessary for this experiment

1. All the libraries installed for experiment 6: PySerialTransfer, keyboard, numpy, and mujoco.
2. **matplotlib, os**

Hardware setup:

1. Lay flat and stick the graph paper to the table using cello tape.
2. Mark the origin using pen by aligning the indexing plate to the grid and fix the robot.
3. Plug in the power supply and set it between 11-12 V.
4. Connect the computer to the robot through the USB-RS-485 converter using the USB cable provided. *Note: Do not connect to the Arduino MEGA port.*
5. Set the USB latency to 4 ms as described in Appendix A.





Tasks

1. Forward kinematics (20 min)
2. Visualising robot motion (20 min)
3. Trajectory planning (10 min discussion)
4. Trajectory tracking (20 min)

1. Forward kinematics

The forward kinematic problem of the five-bar parallel robot has multiple solutions, i.e., for the given values of the motor angles θ_1 and θ_2, the end effector could be at multiple locations.

Since the mathematical treatment of parallel manipulators is beyond the scope of ME 381, we would be using a physics engine—MuJoCo to calculate the position of the end-effector and compare it with the observations from the physical robot.

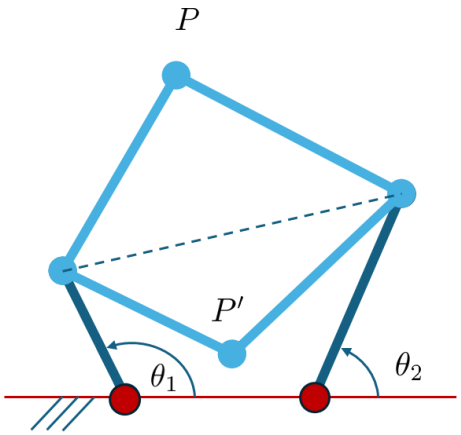


Figure 1:A schematic depicting two solutions for the end effector position.

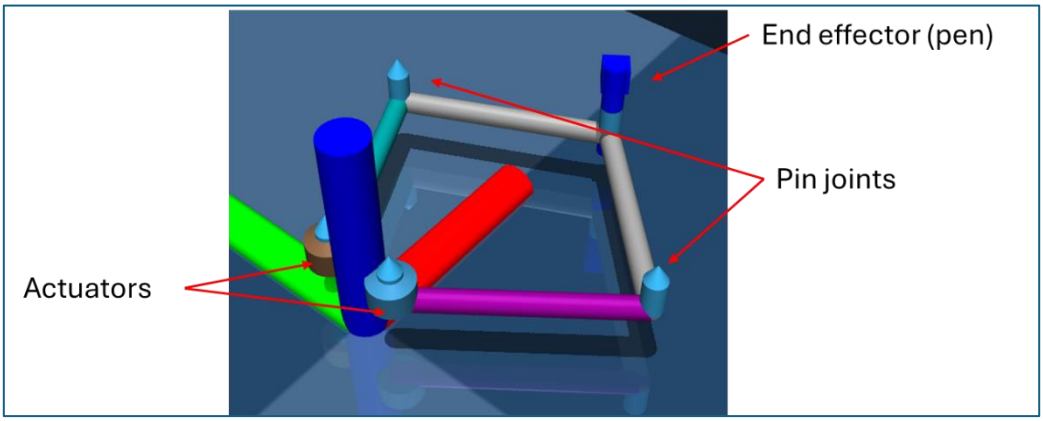
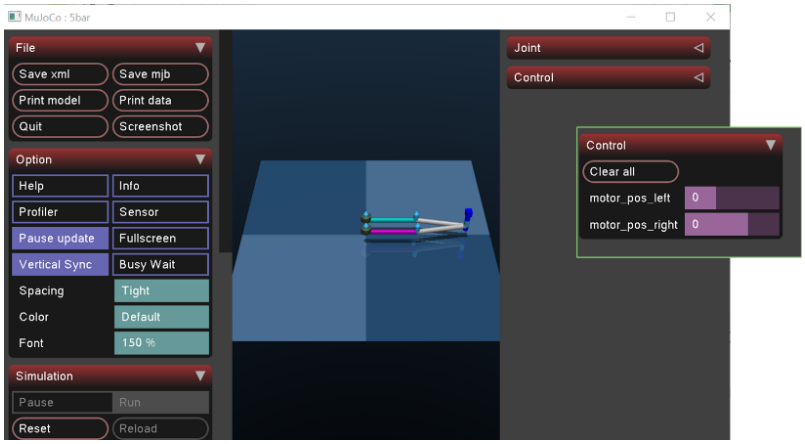


Figure 2: A snapshot of the MuJoCo scene

Steps and report

1. Run the '**forwardkin_5bar.py**' file. A GUI will be launched as shown below.
2. Open the **Control** dropdown menu on the right.



3. Carefully vary the motor angles (displayed in radians) and observe the robot's motion.

4. Set the joint angles to different values
    a. Use the **Spacebar** to do `penDown()/penUp()` and make a mark on the paper.
    b. Measure the coordinates of the marked point.
    c. The end effector position is printed in the PyCharm terminal. Note the values only when the robot reaches equilibrium.
5. **Each student should mention all the values in their report.**

Sr. no.	Left motor (radian)	Right motor (radian)	Calculated coordinates (X,Y) (MuJoCo) (units: mm)	Measured coordinates (X,Y) (units: mm)
1	0	0		
2				
3				
4				
5				
6				

PTO

2. Visualising robot motion

In this section, the robot joint angles are copied into the simulation. One can visualise trajectories drawn by the physical robot in the simulation. This method can be used for teaching continuous paths to the robot.

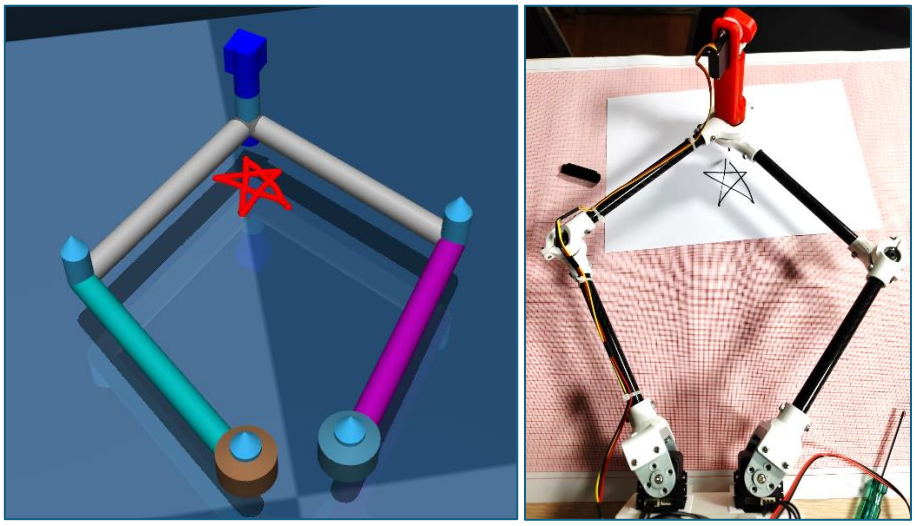
Anatomy of the code

1. Initialise the five-bar robot object.
2. Turn off the motor torques (so that it can be moved freely)
3. Create an instance of MuJoCo simulation (`mujoco.viewer.launch_passive()`)
4. Simulation loop: advance the control simulation for 1/60 sec.
    a. The control inputs, i.e., the desired robot position defined by θ_1 and θ_2 are obtained from the physical robot
    b. The robot motion is integrated (`mj_step()`)
    c. Keyboard inputs are registered
    d. Trajectory way-point plotting and rendering (`viewer.sync()`)

Steps and report

1. Run the '**shadow_5bar.py**' file
2. Use the **Spacebar** to perform `penUp()` and `penDown()`
3. Draw a figure of your choice. Take the actual photo of the drawn shape and a snapshot of the GUI. Every student should report one figure of their choice.

Example:



PTO

3. Trajectory planning of the robot end-effector

Trajectory planning in Cartesian space has two aspects:

- **Path planning**: of the end-effector, e.g., line segment, circular arc, etc.
- **Temporal planning**: one must also provide details on the expected duration of the motion, velocity and acceleration profile to be followed by the end-effector.

Both the objectives are achieved through parametric motion planning:

1. The spatial motion is planned as $\big(X(u), Y(u)\big),\ u \in [u_{LB}, u_{UB}]$. For example, to track a line segment, one could define $u_{LB} = 0,$ at the starting point, and $u_{UB} = 1$ at the endpoint.
2. Temporal motion is planned by defining $u(t),\ t \in [0, T]$. Trapezoidal, parabolic, and cubic variation with time are commonly used time profiles.

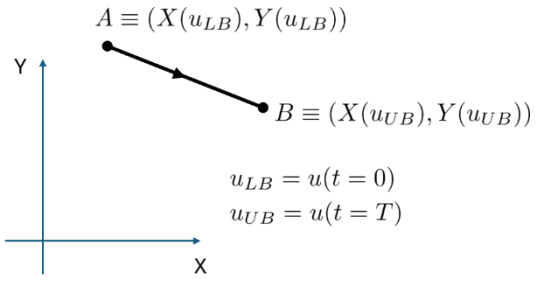


Figure 3 Parametric path planning

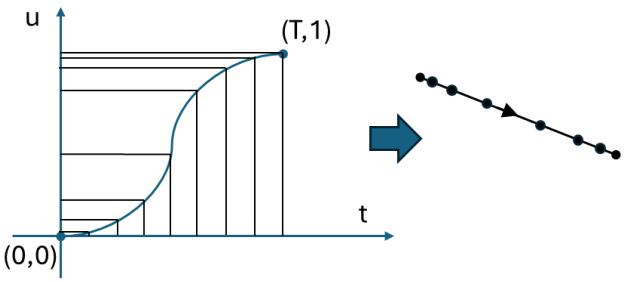
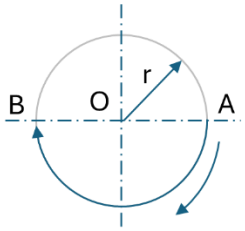


Figure 4 Cubic variation of the parameter 'u' with time. The initial and final $\dot{u}$ are set to zero here.

The cubic variation with time allows for gradual acceleration and deceleration of the actuators.

Homework: Plot $X(t)$ and $Y(t)$ for the following problem

1. **Spatial motion**: write a Python function to output a circular trajectory to be followed by the end-effector (it does not matter which robot is being used). The details of the trajectory are shown in the image below.
    a. Function inputs: centre, radius, parameter θ.
    b. Function outputs: end-effector coordinates $X(\theta)$ and $Y(\theta)$.
    c. The robot should track the trajectory in a clockwise manner as θ increases.
2. **Temporal motion**: use the '**cubic_time_traj()**' function in '**IK_functions.py**' to obtain $\theta(t)$ and hence, $X(t)$ and $Y(t)$ such that
    a. The end effector traces a circular arc starting at A and ending at B, with the centre $O = (200,200)$ and radius $r = 50$ mm within $T = 5$ seconds.
    b. The initial and final velocity of the end effector should be zero.



Once, $X(t)$ and $Y(t)$ are obtained, one can use the robot-specific inverse kinematics to get the joint angles as a function of time.

A few trajectory-tracking examples are implemented in the next section.

4. Trajectory tracking: straight-line segment

The following examples should be implemented based on the method developed in the previous section.

Steps and report:

1. Trajectory tracking examples:
    - Enter the endpoints of the line segments and motion duration according to the following table in the file: '**trajectory_line_segment_5bar.py**' and run the same.
    - Run '**trajectory_lemniscate_5bar.py**'
    - Run '**trajectory_ellipse_5bar.py**'
2. Run the program
    - Understand the steps (and the resulting plots) in trajectory planning.
    - Check the **th1_dot** and **th2_dot** plots so that the values **never exceed 3** rad/s
        - Increase the trajectory duration 'T' if needed
3. **Report**:
    - Photos of the trajectory
    - Planned vs actual variation of joint angles and joint velocities

Table 1 Details of the straight-line segment

Sr no.	P1	P2	Duration (s)
1	[0.3, 0.18]	[0.3,-0.15]	3
2	[0.3, 0.18]	[0.3,-0.15]	1

REPORT

File name: Exp6_<Name>_<RollNo>_<A6> (batch and group no.)

<Your details>

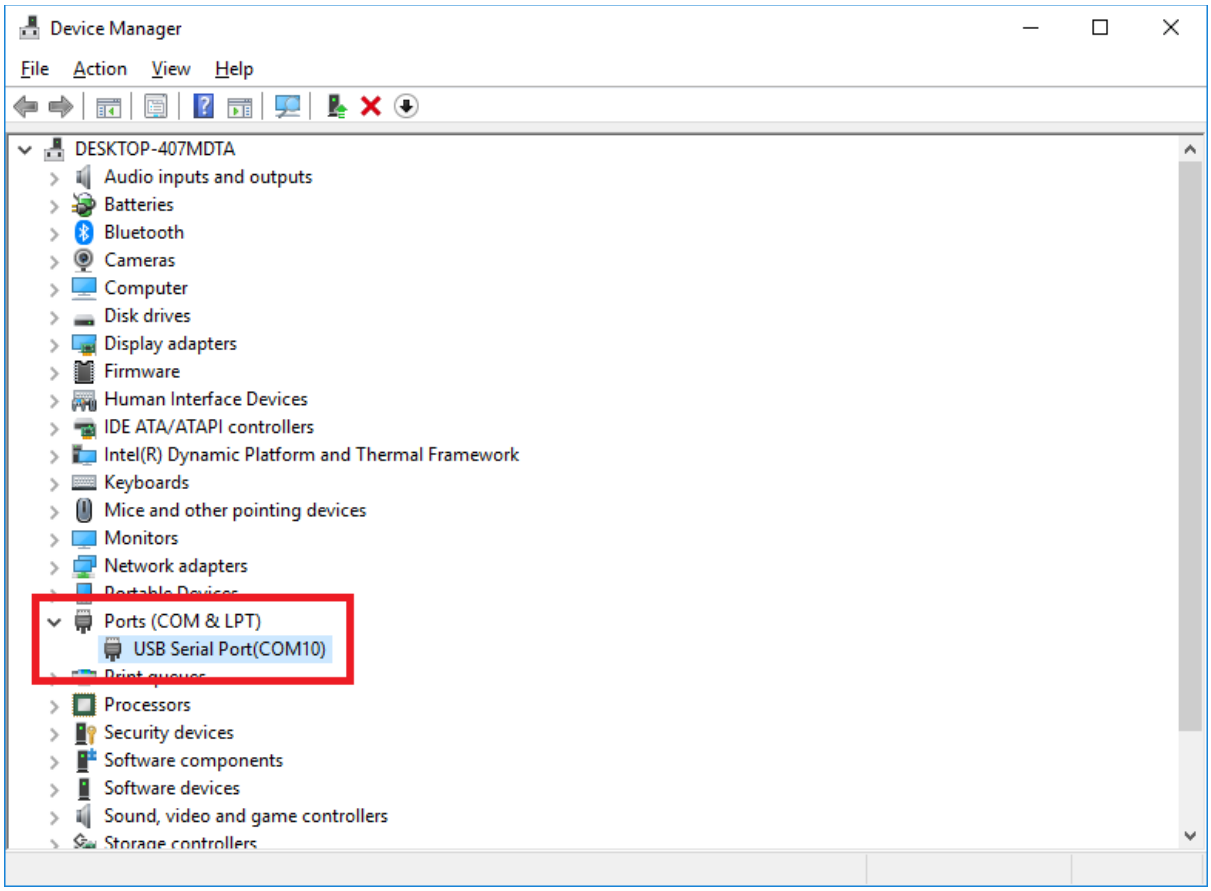
Sr no.	**Experiment section**	**Report**
1	Forward kinematics	Table
2	Visualising the robot motion	Photos and screen captures
3	Trajectory planning	Homework: plots X(t), Y(t)
4	Trajectory tracking: straight-line segment	Plots: planned vs actual variation of joint angles and joint velocities

Appendix A

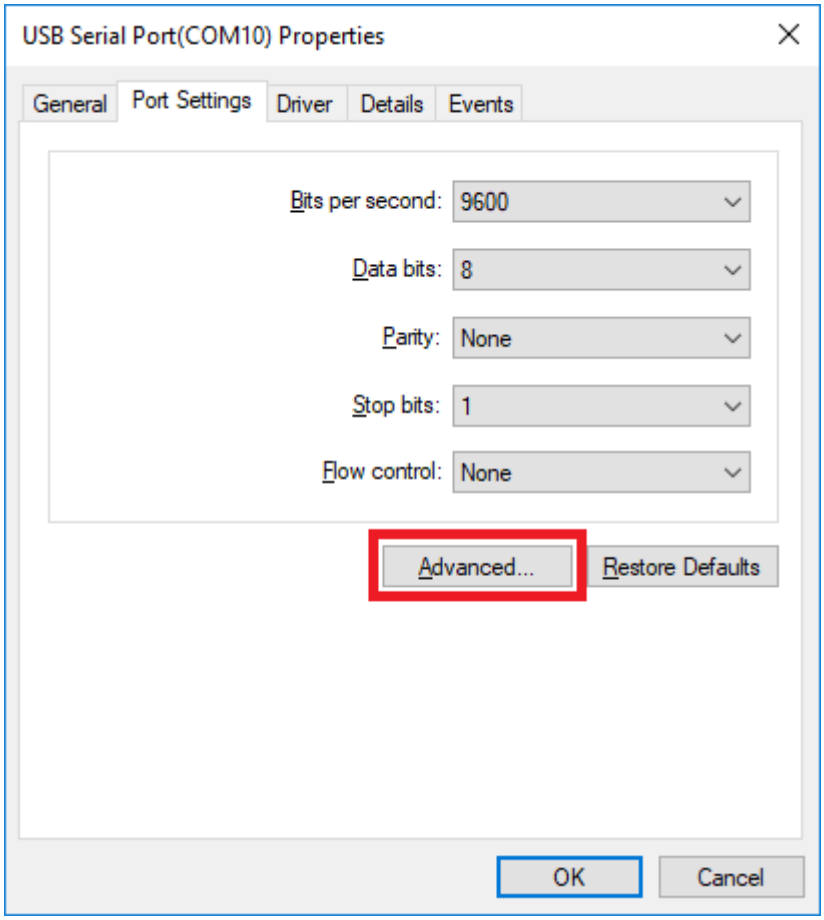
USB Latency Setting

Windows

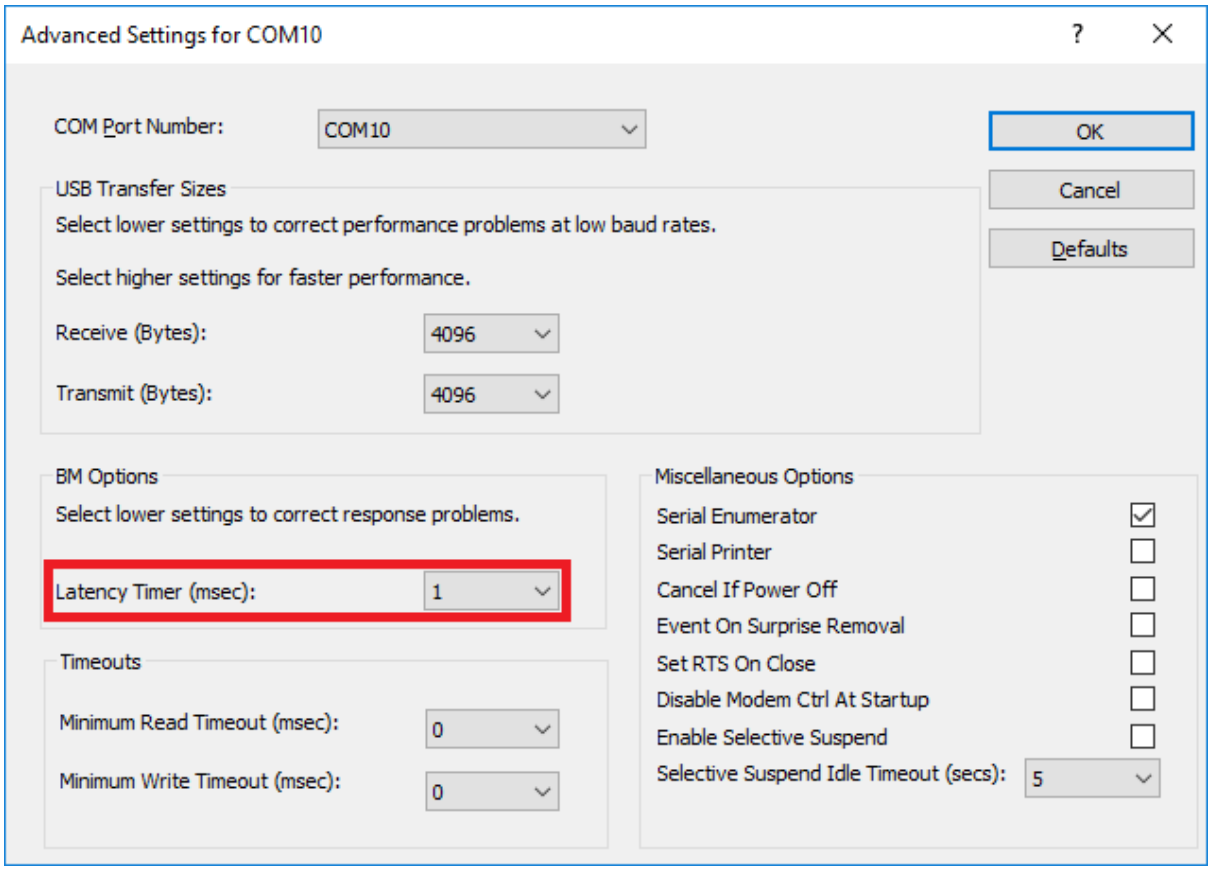
1. Open Device Manager. Go to Ports item and right click on the relative serial port to select Properties.



2. In the Properties window, go to Port Settings tab and click Advanced button.



3. Set the Latency Timer (msec) to 1-4 ms and click OK to confirm the change.



6. 8. 1. 2. Linux

1. Execute below commands to configure the `latency_timer` to `1ms`.

```
# cat /sys/bus/usb-serial/devices/ttyUSB0/latency_timer
16
# echo 1 > /sys/bus/usb-serial/devices/ttyUSB0/latency_timer
# cat /sys/bus/usb-serial/devices/ttyUSB0/latency_timer
1
```