MCEN90028 Robotics Systems
Assignment 4 (34 marks)

Due date: Friday ~~29 May 2020 5pm (Week 11)~~
5 June 2020 5pm (Week 12)

1 General Instructions

1. This is an individual assignment.
2. You are expected to construct you own MATLAB codes in this assignment and not call Peter Corke's toolbox anywhere in your codes.
3. Please read the submission instructions carefully before submitting your assignment.

2 Assignment Description and Tasks

Task 1. Construction of a Robot Dynamic Simulator (8 marks)

In the first task, construct a MATLAB code that represents a 2 link RR (revolute-revolute) robot as shown in Figure 1.

Kinematically,

- $q_1 = 0$ when link 1 is aligned with the inertial X axis.
- $q_2 = 0$ when the robot is straight.
- the axes of rotation for both joints (i.e. $\hat{z}_1$ and $\hat{z}_2$ are defined to point out of the page.
- q_1 and q_2 are not bounded by any joint limits in Tasks 1 and 2. Joint limits are introduced in Task 3 onwards.

The robot has link lengths of L_1, L_2 as indicated, with masses m_1, m_2 as shown and angular moments of inertia:

$$I_1 = \begin{bmatrix} I_{xx1} & 0 & 0 \\ 0 & I_{yy1} & 0 \\ 0 & 0 & I_{zz1} \end{bmatrix}$$

$$I_2 = \begin{bmatrix} I_{xx2} & 0 & 0 \\ 0 & I_{yy2} & 0 \\ 0 & 0 & I_{zz2} \end{bmatrix}$$

about their centres of mass.

The values for the dynamics parameters are:

- $m_1 = 2kg$, $m_2 = 1kg$.
- $I_{zz1} = 0.5kgm^2$, $I_{zz2} = 0.3kgm^2$. $I_{xx1} = I_{xx2} = I_{yy1} = I_{yy2} = 0$.
- $L_1 = 1m$, $L_2 = 0.6m$, $r_{C1} = 0.5m$, $r_{C2} = 0.3m$
- gravity acts in the negative Y direction.

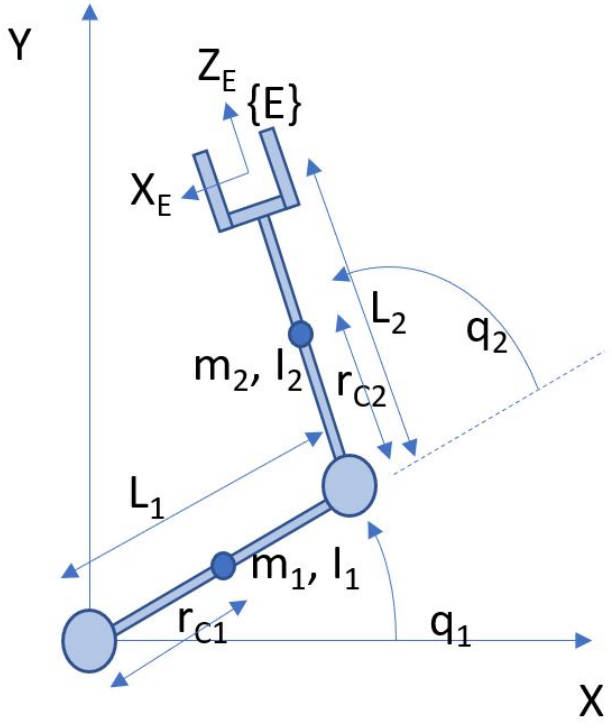

Figure 1: Planar Robot to be simulated. Note that gravity is acting in the negative Y direction.

Download the MATLAB script files provided for the Assignment:

- *Example_RP_robot.m*
- *runrobot.m*

Run the main script file *Example_RP_robot.m*. It will call *runrobot.m*. The *runrobot.m* file contains the robot dynamics (the equation of motion). It takes the input of the current robot state $y = [q, \dot{q}]^T$ and outputs $\dot{y} = [\dot{q}, \ddot{q}]^T$. This gradient is used by the numerical integrator *ode45* in the main file, that takes the joint torque and the robot state, performs the numerical integration to produce the motion of the robot $[q, \dot{q}]^T$ at the next time instance.

The *Example_RP_robot.m* will produce a plot of the movement of the robot vs time. Note that the duration of runtime t_f has been set to $5s$ in the example and the sampling period dt has been set to $0.1s$.

For Task 1, perform the necessary calculations to obtain the equation of motion for the robot described in Figure 1, using the provided sample code *Example_RP_robot.m* and *runrobot.m* as a starting point.

For the report, for Task 1:

1. produce the derivation of the equation of motion of the robot in this assignment (from DH frame assignments, DH parameter, Jacobian matrices to the centres of mass, and finally the matrices A, b, G). (This essentially reviews the kinematics content for the entire subject to date.)

2. produce the plots q_1 vs t and q_2 vs t for $t_f = 5s$, from the initial displacement of $q_1 = q_2 = 0$ at rest, with sampling period $dt = 0.01s$ and joint torque $\tau_1 = \tau_2 = 0$ throughout. Set friction coefficient to $friction\_coeff = 0.1$. Please take note that the friction coefficient in the example has been set to zero. (Go ahead and play around with different settings to familiarise yourself with the code and the robotic system, but please submit only the plot requested to assist with marking).

3. produce the plots q_1 vs t and q_2 vs t for $t_f = 5s$, exactly the same as the previous point, but instead of setting $\tau_1 = \tau_2 = 0$, get the robot to compensate for the gravity forces acting on the robot. Explain how this is achieved in your report. Your result should show that the robot is balanced perfectly against gravity and therefore is not moving. Go ahead and play with the resulting code, for example, intentionally mis-estimate the mass, for example, underestimate the masses and watch as the robot floats like a balloon. Discuss the performance of the proposed method based on your observation.

For your own enjoyment, produce an XY robot animation plot, i.e. the plot of link 1 and link 2 (and the end-effector) of the robot movement in space (like in Figure 1) using MATLAB plot function, such that way you can watch the movement of the robot that you produce visually. This animation can be produced by combining the plot function and the *pause* function. Dictate the values of the axes so MATLAB does not adaptively re-adjust the axes.
For example:

```
for t=0:dt:tf
    plot(.......)
    axis([-5 5 -5 5]) %this sets the axes to avoid re-sizing of ...
       the view of the plot
    pause(.1) %this sets the display speed to 0.1 to correspond ...
       to real time dt
end
```

Listing 1: Example of avoiding re-sizing in MATLAB plots

For the Matlab files to be submitted for Task 1:

- Name your main file as *Task1_main.m*.
- Name your robot dynamics file as *Task1_dynamics.m*.
- Label the figures produced in the code according to the subtasks.
- Use Code Sections in MATLAB to structure the subtasks.

Task 2. PID Controller on robot in joint space (6 marks)

After Task 1, the robot, represented by the *ode45* function, takes the input of joint torque $[\tau_1, \tau_2]^T$. The robot is therefore in "torque-controlled mode". In Task 2, we wish to send joint velocity commands to the robot. This means, in the main file, we would send a joint velocity command at each sampling instance to a PID loop, which implements a closed loop control at sampling period $dt_{PIDloop} = 0.001s$ to regulate the torque sent to the robot joints so that the robot joint velocity would track the joint velocity command given. Note that $dt_{PIDloop}$ is different from dt, which is the main robot control loop. Tune your PID controller to your best efforts – feel free to go back to earlier subjects (such as ELEN90055 Control Systems) if you forgot how to do so. A keyword reminder of *Zielger-Nichols* may help job your memory, though there are many ways. Your tuning does not need to be perfect or optimised, as long as it performs reasonably well. A rough guide would be a settling time for $5\% - 10\%$ tolerance band (aim for 5%) of $\leq 0.01s$ for a $\delta\dot{q} = 1^\circ/s$ commanded step velocity for individual joints.

The outcome of Task 2 is equivalent in practice to having servo motors for your robot, which accept velocity command.

For the report, for Task 2:

1. produce the plots q_1 vs t, q_2 vs t, $\dot{q}_1$ vs t, and $\dot{q}_2$ vs t; for $t = 0$ to $t_f = 5s$, from the initial displacement of $q_1 = 30^\circ$ and $q_2 = 75^\circ$ at rest at $t = 0$, to arrive at the final joint displacement of $q_1 = -15^\circ$ and $q_2 = 45^\circ$, with $\dot{q}_1(t_f) = \dot{q}_2(t_f) = 0$ at $t_f = 5s$. Reference trajectory is to be cubic polynomial of time in joint space.

Sampling period $dt = 0.01s$ (note again, dt is the sampling period of the main control loop, not the PID loop). Set friction coefficient to $friction\_coeff = 0.1$.

2. produce the plots of the corresponding joint torque τ_1 vs t and τ_2 vs t (corresponding to the resulting robot motion in the first point of Task 2).

3. produce the XY plot of the robot (like in Figure 1) which consist of link 1 and link 2 (and the end-effector, e.g. marked with a circle of triangle), every 0.5s. (meaning at $t = [0, 0.5, 1, ..., 5]s$) for the resulting robot motion in Task 2.

For the Matlab files to be submitted for Task 2:

- Name your main file as *Task2_main.m*.
- Name your robot dynamics file as *Task2_dynamics.m*.
- Label the figures produced in the code according to the subtasks.
- Use Code Sections in MATLAB to structure the subtasks.

Hint
Added on 20 May 2020.
During the workshop today, it appeared that a lot of students were not able to get past Task 2, which was to construct a PID closed loop controller to allow the robot to receive a joint velocity command $\dot{q}_C$ instead of the torque command. This is equivalent (in real life) to having a velocity servo drive for your motors, which accepts velocity command from your main robot loop.

This velocity control loop is to run 10 times faster than the main robot control loop. This essentially allows the joint velocity controller 10 sampling instances to produce the velocity step response on the robot joints for every joint velocity command issued by the main loop.

How is it done?
This is a closed-loop system with $\dot{q}_C$ as the reference input, a PID as the controller (that you are meant to design and tune), the robot as the plant (the robot as a plant takes in joint torque τ as the input and robot joint motion q and $\dot{q}_C$ as outputs) and the output $\dot{q}$ is used as a feedback to this velocity control loop. Therefore:

$$\begin{aligned} e_{vc} = &\quad \dot{q}_C - \dot{q}; \\ \tau = &\quad K_p e + K_d \dot{e} + K_I \int_0^t e d\tau; \end{aligned} \tag{1}$$

where vc denotes "velocity control", and K_p, K_d, K_I are diagonal matrices containing the gains assigned to each joint, of size $n \times n$, where n is the number of joints in the robot.

Hope that helps. Have fun!

Task 3. Joint space control of the velocity commanded robot (8 marks)

Continuing from where we left off in Task 2, this time, we are going to get the end-effector of the robot to move in "task space", while still utilising joint space control. That is, you are to generate a reference trajectory (using cubic polynomial) to describe the reference trajectory in (x_{ref}, y_{ref}) for the end-effector, from an initial coordinate (x_i, y_i) to the final coordinate (x_f, y_f) in a straight line. The reference trajectory is converted (using inverse kinematics) within at each robot sampling instance dt into a reference joint space trajectory, that is then used to command the velocity commanded robot (i.e. as the input to the velocity PID control of the robot joints).

NOTE: In Task 3, construct a joint limit of $-60^\circ \leq q_1 \leq 60^\circ$ and $-90^\circ \leq q_2 \leq 90^\circ$ in your code.

Two reference trajectories for the end-effector are to be generated and tracked by the robot:

- **Trajectory A**: initial end-effector position: $(x_i, y_i) = (0.71, 1.08)m$ at $(\dot{x}_i, \dot{y}_i) = (0, 0)m/s$, final end-effector position: $(x_f, y_f) = (1.485, 0.041)m$ and velocity at the final position $(\dot{x}_f, \dot{y}_f) = (0, 0)m/s$. Trajectory duration: $t_f = 5s$.
- **Trajectory B**: initial end-effector position: $(x_i, y_i) = (0.71, 1.08)m$ at rest and comes to a complete stop at final end-effector position: $(x_f, y_f) = (1.36, -0.59)m$. Trajectory duration: $t_f = 5s$.

For the report, for Task 3:

1. produce the plots of the end-effector position x vs t and y vs t for **Trajectory A**. Superimpose the reference trajectory x_{ref} and y_{ref} on the plot.
2. produce the XY plot of the robot (like in Figure 1) which consist of link 1 and link 2 (and the end-effector, e.g. marked with a circle or triangle), every 0.5s, (meaning at $t = [0, 0.5, 1, ..., 5]s$) for the resulting robot motion in **Trajectory A**. Comment on the difference between the resulting motion and that produced in Task 2.
3. produce the XY plot of the robot (like in Figure 1) which consist of link 1 and link 2 (and the end-effector, e.g. marked with a circle or triangle), every 0.5s, (meaning at $t = [0, 0.5, 1, ..., 5]s$) for the resulting robot motion in **Trajectory B**.

   Note:

   - Trajectory B is designed so that it passes outside the reachable workspace dictated by the joint limits of the robot.

- Superimpose the rough plot of the reachable workspace to aid your thinking.
- Discuss your results and observations.

For the Matlab files to be submitted for Task 3:

- Name your main file as *Task3_main.m*.
- Name your robot dynamics file as *Task3_dynamics.m*.
- Label the figures produced in the code according to the subtasks.
- Use Code Sections in MATLAB to structure the subtasks.

Task 4. Task space control of the velocity commanded robot (6 marks)

In Task 4, we would like to implement Task Space control, which means: our error term in the main loop ought to be in the task space. Using end-effector **Trajectory A** in Task 3, construct the error signal $e_x = x_{ref} - x_r$ and $e_y = y_{ref} - y_r$ at every sampling instace dt, where x_{ref} and y_{ref} are the reference position of the end-effector and x_r and y_r are the actual end-effector of the robot, obtained by performing forward kinematics of the output of the robot simulator $(q, \dot{q})$. Generate task space velocity command $(\dot{x}_C, \dot{y}_C)$ through a P-controller of the error (or PI controller – state what you chose to implement in your report). Use the inverse of the Jacobian matrix to obtain the joint velocity command to be sent to the velocity commanded robot.

In Task 4, the joint limit of $-60^\circ \leq q_1 \leq 60^\circ$ and $-90^\circ \leq q_1 \leq 90^\circ$ introduced in Task 3 is retained.

For the report, for Task 4:

1. produce the plots of the end-effector position x vs t and y vs t for **Trajectory A**. Superimpose the reference trajectory x_{ref} and y_{ref} on the plot.
2. produce the XY plot of the robot (like in Figure 1) which consist of link 1 and link 2 (and the end-effector, e.g. marked with a circle of triangle), every 0.5s, (meaning at $t = [0, 0.5, 1, ..., 5]s$) for the resulting robot motion in **Trajectory A**. Comment on any observation you had between the resulting motion and that produced in Task 3.

For the Matlab files to be submitted for Task 4:

- Name your main file as *Task4_main.m*.
- Name your robot dynamics file as *Task4_dynamics.m*.

- Label the figures produced in the code according to the subtasks.
- Use Code Sections in MATLAB to structure the subtasks.

Task 5. Task space control of the velocity commanded robot with reactive obstacle avoidance (6 marks)

Task 5 continues from Task 4, where the robot is to execute Trajectory A with Task Space control of the velocity commanded robot. However, a circular obstacle appears in the path. The robot does not know where it is beforehand (it is not a known obstacle from the start as the one you encountered in Assignment 3). However, you have been given a wonderful sensor, mounted at the end-effector. This sensor provides one set of measurement at each time instance. It can be called at each time instance by:

```
[d, theta] = sensor_t5(obs_init, xr, yr)
```

Inputs:

obs_init	A variable to handle the generation of a new obstacle. Set it to 1 if a new obstacle needs to be generated, 0 if not initialising the obstacle. An example code is provided below shown in Listing 2.
xr, *yr*	The robots end-effector coordinate in the **inertial** frame at the time the function is called.

Outputs:

d	The distance (in metre) of the end-effector from the nearest detected point (of the obstacle) (therefore it is a scalar value, not a vector).
theta	The angle (in radians) to the same point from the end-effector, measure from the **inertial** positive X axis.

```
newObs=1;   % create a new obstacle in the environment.
for ...
    ...
    [d, th]=sensor_t5(newObs,x,y);
    newobs=0;   % avoid changing the obstacle during the motion.
    ...
end
```

Listing 2: Example of initialising the obstacle

The function *sensor_t5.p* is provided for this assignment. The sensor output variables d and θ are shown in Figure 2. It is assumed that there is only **ONE** (circular) obstacle in the scenario.
The sensor has a maximum range of $0.35m$, beyond which it will not detect the obstacle and will output the value $d = 1000$ and $theta = 1000$. This means that you will not see the obstacle until it is within $0.35m$ of the end-effector. The strategy you

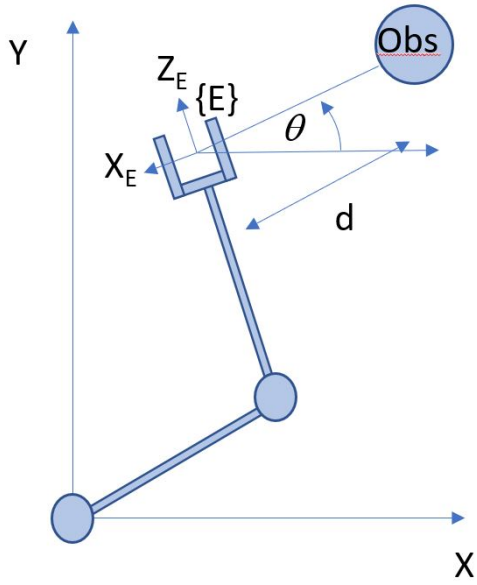

Figure 2: The sensor outputs: d and θ shown in this diagram. The obstacle (Obs) is assumed to be always a circle. Its location is recorded inside the function of the sensor.

need to come up with is therefore described as being "reactive", or in the literature, it is often called "online obstacle avoidance".

You therefore initialise Task 5 driving the robot end-effector to track **Trajectory A**. In the effort to avoid the obstacle, you are to get your robot to track the desired trajectory as much as possible and only deviate from it where it is necessary to avoid the obstacle. In the main loop, you will call the *sensor_t5(obs_init, xr, yr)* function at each sampling instance and decide what to do. This has not been explicitly covered in our lectures, but it is part of the open-ended brain teaser to the class. There is plenty in the literature in the 80s - 90s on these type of problems if you wish to read up. Focus on the reactive strategies.

In Task 5, the joint limit of $-60^\circ \leq q_1 \leq 60^\circ$ and $-90^\circ \leq q_1 \leq 90^\circ$ introduced in Task 3 is retained.

For the report, for Task 5:

- In less than 500 words, explain the outline of your strategy in guiding the robot end-effector through Trajectory A while avoiding the obstacle. Please do not use ready made functions, such as those in the Matlab Robotics Toolbox. Include references if you use material from the literature.

  IMPORTANT

If you utilised any methods taken from the literature (from a paper, textbook, etc), provide a reference to the source. Add a section at the end of the report titled "references". Learn how to cite literature source appropriately in your own time at: https://students.unimelb.edu.au/academic-skills/explore-our-resources/referencing

At this stage of your master degree, ignorance in how to appropriately acknowledge your source is not a valid excuse if you were found to plagiarise external material. If you have not already done so, please educate yourself as to the boundary of what constitute acceptable and non-acceptable practice. Those boundaries include the cases of how to appropriately report material that you may share with your peers in class. Know the difference between group discussion and helping each other to learn material and copying someone else's work.

- As part of your explanation above, produce the necessary plots to demonstrate your result implementing a Task Space control to the velocity commanded robot to traverse **Trajectory A**, with the adjustments made to avoid the obstacle. Include in the results relevant variables over t such as: the reference trajectory, the actual trajectory traversed by the robot to avoid the obstacle, sensor readings during the operation, and other variables relevant to demonstrating the effectiveness or in explaining the inner working of the strategy of your choice. Refer to these plot when explaining your answer and explain the plots.

3 Submission

In your report, detail the steps and understanding in producing your answers. Do **not** include the MATLAB codes in the body of the report. Include only the pseudo-code if necessary and explain the logic.

You need to submit **one report per person** plus the necessary MATLAB codes.

The report should be in an appropriate engineering report format and submitted as a PDF. The report should be **no more than 30 pages** (everything included) with **12pt font size**.

Compress your report and all your relevant MATLAB files as a .zip file with folder name **"Assignment4_[#]"**, where you replace [#] with your student ID (eg. Assignment4_123456) and submit to LMS.

Note: only the last submission will be assessed.

Submission checklist:

- ☐ Report is no more than 30 pages
- ☐ Report has 12pt font size
- ☐ Report saved as PDF
- ☐ Titles of files are in the right format
- ☐ Compressed all files as a single .zip file

4 Academic Integrity

We take academic integrity seriously. Please note that while students may discuss and help each other in the thinking process, you should work on your assignments separately.

Details about academic integrity can be found on MCEN90028 Canvas page (under Subject Overview) or at `http://academicintegrity.unimelb.edu.au/`. Please check with the tutors or the lecturer if you are in doubt. Ignorance is not a valid reason for academic misconducts.