ME650
Robot Manipulators

Spring 2022

Obstacle Avoidance for an 8-DOF 8R Manipulator in 2D Space via Task Priority Redundancy Resolution and Artificial Potential Field Methods

---

28th April, 2022

"I pledge that I have abided by the graduate student code of Academic Integrity."

The paper has been prepared by:

Aldrin D. Padua

ABSTRACT

This paper implemented two obstacle avoidance resolution methods namely, Task Priority Redundancy Resolution (TPRR) and Artificial Potential Field (APF), to an arbitrary 8-DOF 8R manipulator in 2D space. Both were successful at guiding the motion of the subject manipulator, ultimately leading the end-effector to the desired position in space. However, few key differences were observed during the implementation which mainly revolved around the relationship of parameter-tuning and desirable results. The TPRR method proved to be easier to tune, thus, more advantageous than AP. It is measured by the difficulty in finding the tuning parameter values that would converge the algorithm with a desirable accuracy without making the convergence time suffer while still avoiding obstacles. This trade-off was easily minimized using TPRR while it was much harder to control in the APF method.

INTRODUCTION

One of the most popular obstacle avoidance methods in robot manipulators and probably one of the few methods that are always taught in class is the Artificial Potential Field (APF). The idea is to move a manipulator's end-effector through an artificial potential field until it reaches a desired destination. In this field of artificial potentials, the robot's arms are subjected into two kind of forces namely, (1) force of attraction (F_{att}), and (2) force of repulsion (F_{rep}). F_{att} is a force coming from the destination, while F_{rep} is a force emitted by obstacles which ensured a collision-free motion. While APF is very popular, there other methods which prove to be just as effective but relatively more efficient.

This paper explored one of the mentioned relatively better methods namely, the Task Priority Redundancy Resolution (TPRR). TPRR enjoys more ease in tuning parameters than APF which means that a desired time of convergence is more attainable without sacrificing too much of desired accuracy, or vice versa.

This paper implemented both obstacle avoidance methods in an arbitrary 8-DOF 8R manipulator in 2D space as shown in Figure 1.

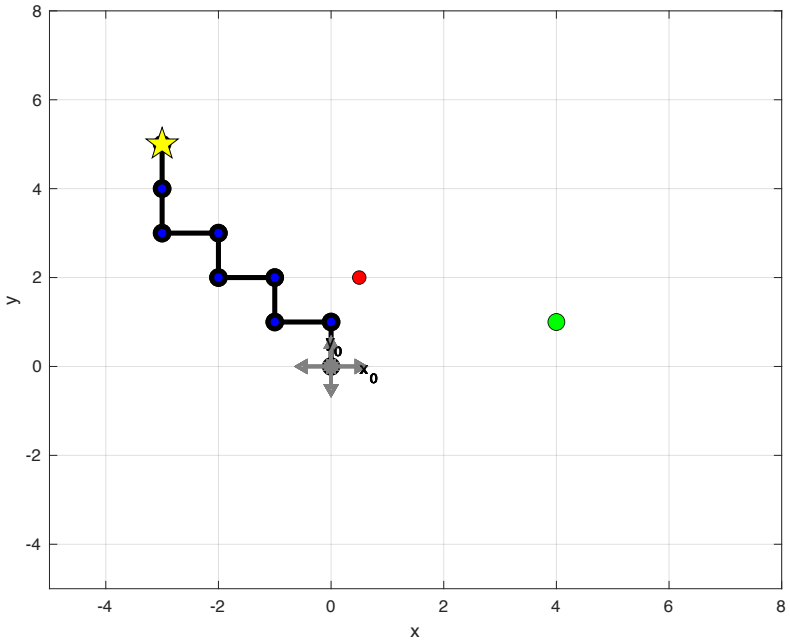

Figure 1. The subject of the paper is an arbitrary 8-DOF 8R manipulator whose link lengths are all equal. The yellow pentagram represents the end-effector, the blue circles represent the joints, and the red and green circles represent the obstacle and the goal, respectively.

In the implementation, the following assumptions were made:

1. For both methods, the end-effector's goal is solely defined by the position and not the orientation. Thus, the orientation of the end-effector with respect to the world frame does not matter.
2. For APF, the attractive potential field only uses the parabolic well function at all times. (This is for the sake of simplicity. Also, doing so does not prove to cause any issues in the motion of the subject manipulator. Hence, the need for the transition to conic well potential is deemed unnecessary.)
3. For APF, the goal is solely defined for the end-effector. All other joints are free to converge at any point satisfying the obstacle avoidance objective. Hence, the resultant torque vector is purely a battle between the forces acting upon the end effector and the closest point to the obstacle.
4. The lengths of all manipulator links are equal.
5. The obstacle is just a point in space.

The objective of this paper is to compare the two methods and highlight the strong points and areas of improvements of each method.

THEORY AND EXPERIMENTAL PROCEDURE

I. 8-DOF 8R Manipulator

The D-H table for the subject manipulator was derived as follows:

Table 1. This is the DH-table of the 8-DOF 8R subject manipulator of this paper.

Link No.	θ_i	d_i	a_i	α_i
1	q_1	0	l_1	0
2	q_2	0	l_2	0
3	q_3	0	l_3	0
4	q_4	0	l_4	0
5	q_5	0	l_5	0
6	q_6	0	l_6	0
7	q_7	0	l_7	0
8	q_8	0	l_8	0

where

$\theta_i =$
$the\ angle\ between\ x_{i-1}\ and\ x_i\ \ axes, or\ simply\ the\ angle\ of\ rotation\ of\ joint\ j_i$,
$d_i =\ the\ displacement\ between\ x_{i-1}\ and\ x_i\ \ axes\ measured\ along\ the\ z_{i-1}\ axis,$
$a_i =\ distance\ between\ z_i\ and\ z_{i-1}\ \ axes\ measured\ along\ the\ x_i\ axis$,
$\alpha_i =\ the\ angle\ by\ which\ z_{i-1}$ axis need to rotate to mimic the position of z_i axis,
$l_i = link\ length$, and
$i = joint\ number$.

Using DH convention, the system of equations for the end effector's position was determined and the Jacobian was derived as follows:

$$J = \begin{bmatrix} -(l_8 s_{12345678} + l_7 s_{1234567} + l_6 s_{123456} + l_5 s_{12345} + l_4 s_{1234} + l_3 s_{123} + l_2 s_{12} + l_1 s_1) \dots \\ -(l_8 s_{12345678} + l_7 s_{1234567} + l_6 s_{123456} + l_5 s_{12345} + l_4 s_{1234} + l_3 s_{123} + l_2 s_{12}) \dots \\ -(l_8 s_{12345678} + l_7 s_{1234567} + l_6 s_{123456} + l_5 s_{12345} + l_4 s_{1234} + l_3 s_{123}) \dots \\ -(l_8 s_{12345678} + l_7 s_{1234567} + l_6 s_{123456} + l_5 s_{12345} + l_4 s_{1234}) \dots \\ -(l_8 s_{12345678} + l_7 s_{1234567} + l_6 s_{123456} + l_5 s_{12345}) \dots \\ -(l_8 s_{12345678} + l_7 s_{1234567} + l_6 s_{123456}) \dots \\ -(l_8 s_{12345678} + l_7 s_{1234567}) \dots \\ -(l_8 s_{12345678}); \\ l_8 c_{12345678} + l_7 c_{1234567} + l_6 c_{123456} + l_5 c_{12345} + l_4 c_{1234} + l_3 c_{123} + l_2 c_{12} + l_1 c_1 \dots \\ l_8 c_{12345678} + l_7 c_{1234567} + l_6 c_{123456} + l_5 c_{12345} + l_4 c_{1234} + l_3 c_{123} + l_2 c_{12} \dots \\ l_8 c_{12345678} + l_7 c_{1234567} + l_6 c_{123456} + l_5 c_{12345} + l_4 c_{1234} + l_3 c_{123} \dots \\ l_8 c_{12345678} + l_7 c_{1234567} + l_6 c_{123456} + l_5 c_{12345} + l_4 c_{1234} \dots \\ l_8 c_{12345678} + l_7 c_{1234567} + l_6 c_{123456} + l_5 c_{12345} \dots \\ l_8 c_{12345678} + l_7 c_{1234567} + l_6 c_{123456} \dots \\ l_8 c_{12345678} + l_7 c_{1234567} \dots \\ l_8 c_{12345678} \dots \end{bmatrix}$$

where s and c correspond to sine and cosine functions, and their subscripts denote the summation of the angles of rotation in each applicable joint.

All Jacobian for each point along the robot's length were also derived and are detailed in the attached source codes entitled "sense.m."

II. Task Priority Redundancy Resolution

The TPRR method relies on the general solution for non-homogenous linear system of equations given by the following equation:

$$\dot{q} = J^{+}\dot{x} + (I - J^{+}J)\eta \qquad (Eq. 1).$$

TPRR works by exploiting the remaining redundancy of the solution as given by the homogenous part (second term) in Eq. 1. Mathematically speaking, the solution to the obstacle avoidance, resides in the null space of ${J_1}^{+}J_1$.

To derive the TPRR equation, first, $\dot{q}$ is solved for the primary task which is to drive the end-effector to the desired position. Then, $\dot{q}$ is solved again for the secondary task which is to avoid the obstacle. In this part, $\dot{x}$ is no longer that of the end-effector's, but rather, that of the arm's closest point to the obstacle (x_2). The same goes for the Jacobian.

Using above logic, η is derived and the resulting equation is as follows (*Maciejewski & Klein, 1985*):

$$\dot{q} = {J_1}^{+}\dot{x_1} + [J_2(I - {J_1}^{+}J_1)]^{+}\left(\dot{x_2} - J_2{J_1}^{+}\dot{x_1}\right) \qquad (Eq. 2)$$

where $J_1$is the Jacobian of the primary task, J_2 is the Jacobian of the secondary task, $\dot{x_1}$is the end-effector velocity, and $\dot{x_2}$ is the velocity of x_2.

Maciejewski & Klein further modified Eq. 2 to add a control to the degree of influence that an obstacle has on the motion of the manipulator. The control parameters were inserted as follows:

$$\dot{q} = {J_1}^{+}\dot{x_1} + \alpha_1[J_2(I - {J_1}^{+}J_1)]^{+}\left(\alpha_2\dot{x_2} - J_2{J_1}^{+}\dot{x_1}\right) \qquad (Eq.3).$$

The alphas along with the error tolerance, ε, served as the tuning parameters for the algorithm implemented in this paper to minimize the trade-off between accuracy, obstacle avoidance, and convergence time. The first tuning parameter, α_1, denoted the gain of the whole homogenous part of the solution, and α_2 denoted the gain of $x_2$velocity.

The motion was propagated in conjunction with the Resolved Rates Algorithm (RRA) which served as the step trigger to move the end-effector from initial point to the destination while avoiding the obstacle.

III. Artificial Potential Field

APF method revolves around the idea of constructing a virtual potential field by treating the robot as point particle and making it travel from the initial point to the goal while avoiding collision. The resultant motion is similar to a ball travelling down a downhill slope, maneuvering its way around obstacles to ultimately reach its destination. The artificial potential field has two components. The first component ($U_{att\,|\,q\_goal}$) attracts the robot to the goal and the second one repels the robot from the boundary of the obstacle ($U_{rep\,|\,q\_rep}$).

Taking the negative gradient of each potential field components yields the forces of attraction and repulsion needed to drive the robot motion. This is called the gradient descent technique. Mathematically, it means that the partial derivative of the potential field function is taken with respect to each concerned variable. Taking the negative side of the partial derivative gives the minimum function point, hence, the technique is also called the steepest descent. Intuitively, it is represents the down-hill slope that will get the point particle-robot to its next destination at every step in the quickest way possible.

The force of attraction is derived as follows:

$$F_{att,i}(q) = \begin{cases} -\zeta_i\left(o_i(q) - o_i(q_{goal})\right) & : \quad \left\|o_i(q) - o_i(q_{goal})\right\| \leq d \qquad (Eq.4) \\ -d\zeta_i\dfrac{o_i(q) - o_i(q_{goal})}{\left\|o_i(q) - o_i(q_{goal})\right\|} & : \quad \left\|o_i(q) - o_i(q_{goal})\right\| > d \qquad (Eq.5) \end{cases}$$

where ζ_i is the gain of the attractive force, o_i refers to the position coordinate of a point along the robot's length, q denotes the current configuration, q is the goal configuration, and d is a value set

to transition between the two well functions. Eq. 4 is also called the gradient of the parabolic potential while Eq. 5 is the gradient of the conic well potential.

For simplicity, this paper considered d to be always infinite, hence, Eq. 4 was solely used to compute the force of attraction at every step. It was experimentally determined that doing so does not cause any issue with the resultant motion within the constraints set in the context of this paper.

The force of repulsion is derived as follows:

$$F_{rep,i}(q) = \begin{cases} -\eta_i \left(\frac{1}{\rho(o_i(q))} - \frac{1}{\rho_o}\right) \frac{1}{\rho^2(o_i(q))} \nabla\rho(o_i(q)) & (Eq.6) \\ 0 & (Eq.7) \end{cases}$$

where η_i is the gain of the repulsive force, $\rho(o_i(q))$ is the distance from the closest point in the manipulator to the obstacle, ρ_o is the radius of the region of influence (ROI) of the obstacle, and $\nabla\rho\big(o_i(q)\big)$ is the direction of the repulsive force. Eq. 6 results when any point along the robot's arm is within the ROI of the obstacle. Eq. 7 means that the robot does not feel any repulsive force which results from it being outside the boundaries of the ROI.

Normally, APF requires each desired point in the manipulator to have its own respective goal. This leads to the total torque computation as follows:

$$\tau = \sum_i {J^T}_{o_i(q)} F_{att,i}(q) + \sum_i {J^T}_{o_i(q)} F_{rep,i}(q) \qquad \text{(Eq. 8)}$$

where J is the Jacobian of each point.

However, in the context of this paper and in accordance with the assumptions set in Part I, all other points along the robot's length except the end-effector's were made free to converge at any location as long as their combination avoided the obstacle. So, the torque equation used at every step of the algorithm is as follows:

$$\tau = {J_1}^T F_{att,i}(q) + {J_2}^T F_{rep,i}(q) \qquad \text{(Eq. 9)}$$

where J_1 is always the Jacobian related the end-effector, and J_2 is the Jacobian related to the closest point to the obstacle. The first term in Eq. 9 is always present while the seconds term drops or appears depending on whether a point falls within the ROI of the obstacle. The torque formulas used in Eq. 8 and 9 were derived using the principle of virtual work (*Spong et al, 2006*).

In the algorithm, the joint velocity vector at every step was computed in proportion to the normalized torque vector as given in the following equation:

$$\dot{q} = \alpha \frac{\tau}{\|\tau\|} \qquad \text{(Eq. 10).}$$

The three tuning parameters for the APF algorithm were then determined as ζ_i (Eq.4), η_i (Eq. 6), and α (Eq. 10) along with the error tolerance, ε.

RESULTS AND DISCUSSION

Each method was tested with the following constant parameter values:

- Initial configuration: $q_0 = [90° \ 90° \ -90° \ 90° \ -90° \ 90° \ -90° \ 0]'$
- Desired end-effector position: $x_{goal} = [3 \ 1]'$
- Obstacle position: $x_{obs} = [0.9 \ 1.8]'$
- Radius of the ROI: $radius_{ROI} = 0.3 \ unit$

I. Task Priority Redundancy Resolution Algorithm

Table 2. This is the summary for all trials done under the TPRR algorithm. Results show that despite differences in the outcome, the algorithm proved to be highly convergent, thus, easier to tune.

Trial No.	TUNING PARAMETERS			RESULTS		
	Error Tolerance (ε)	α_1	α_2	Closest Distance to Obstacle ($dist2obs$)	Converged?	Convergence Time
1	10^{-2}	$\frac{1}{\|\|x_2 - x_{obs}\|\|}$	1	0.45	TRUE	145 s
2	10^{-1}			0.31		143 s
3	10^{-2}		0.5	0.34		145 s
4	10^{-2}		10	0.90		162 s
5	10^{-2}	0.5	1	0.25		145 s
6	10^{-2}	10		0.56		147 s

Results in Table 1 shows how easy it is to tune the parameters in the TPRR algorithm. This is proven by the fact that the changes in the tuning parameters did not affect the convergence outcome of the every motion.

Trials 1 and 2 show that changing only the error tolerance of the algorithm impacts the distance maintained by the closest point x_2 to the obstacle ($dist2obs$), while the convergence time is minimally affected. The value of $dist2obs$ denotes how well the algorithm kept the distance of x_2 away from the boundaries of the obstacle's region of influence (ROI) as denoted by the parameter $radius_{ROI}$. This means that decreasing the desired accuracy puts $dist2obs$ near the boundary of the obstacle's ROI. In the context of the results in Table 1, the effect is not significantly bad as it is still outside the boundaries. However, it poses a limit and a warning to not go any further beyond the imposed change in tolerance as it might worsen the results. Intuitively, it will mean that a part of the robot's arm will come nearer the obstacle, thus, increasing a chance of collision in cases of unexpected external stimuli.

Trials 3 and 4 show how changing α_2 significantly affects dist2obs as well as the convergence time. The tuning parameter $\alpha_2$represents the gain or magnitude of influence of the secondary task's

velocity (velocity that drives the x_2 away from the obstacle's ROI) over that of the primary task's (velocity that drives the end-effector to the goal). Comparing Trial 3 to Trial 1, it is proven that decreasing this parameter would result in a $dist2obs$ just a few units outside the ROI. This means that further value decrements would only push x_2 nearer and nearer to the obstacle. On the other hand, increasing this parameter as shown in Trial 4 is proven to cause $dist2obs$ to be much greater than $radius_{ROI}$ which means that x_2 gets further and further away from the obstacle. However, it comes at the expense of increasing the convergence time.

Trials 5 and 6 show how solely changing $\alpha_1$affects $dist2obs$. The tuning parameter $\alpha_1$represents the magnitude of influence of the whole secondary task over the primary task. This means that if the value is less than 1, then the impact of the primary task is much greater than that of the secondary task. In application, this means that the end-effector's velocity is overpowering the velocity of the x_2, thus, the latter gets pulled much closer to the obstacle, significantly falling within the boundaries of the ROI as shown in Trial 5. On the other hand, increasing $\alpha_1$to a value much higher than α_2 means the secondary task is overpowering the primary leading to $dist2obs$ much larger than $radius_{ROI}$, thus x_2 gets further away form the obstacle. Time is minimally affected.

For best results and to ensure observance of the obstacle's ROI, it is best to use parameters following the logic and proportion behind Trial 1. The dynamicity of the formulation for $\alpha_1$and its proportion to α_1 is a good measure to minimize risk of collision.

II. Artificial Potential Field Algorithm

Table 3. This is the summary for all trials done under the APF algorithm. Results the difficulty in tuning the parameters as evidenced by the convergence rate.

Trial No.	**TUNING PARAMETERS**				**RESULTS**		
	Error Tolerance (ε)	ζ	η	α	Closest Distance to Obstacle ($\|\|x_2 - x_{obs}\|\|$)	Converged?	Convergence Time
1	10^{-2}	1	1	0.005	0.31	TRUE	$258\ s$
2	10^{-1}				0.30	TRUE	$224\ s$
3	10^{-2}			0.01	0.30	FALSE	inf
4	10^{-1}				0.30	TRUE	$188\ s$
5	10^{-2}		0.5	0.005	0.30	FALSE	inf
6			10		0.30	FALSE	inf
7		0.5	1		0.31	TRUE	$273\ s$
8		10			0.30	FALSE	inf

While convergence is not (or rarely) a problem with TPRR algorithm, it is a much greater concern in APF as shown in Table 2. Tuning the parameters to ensure convergence while at the same time minimizing the trade-off between accuracy (ε) and time is non-trivial with this algorithm.

Trials 1 and 2 show that the trade-off between accuracy and time is very significant. When a much higher accuracy is desired as in Trial 1, the convergence time suffers. Comparing this to Part I Trial 1, the time difference is large. To combat this, one may choose to sacrifice a little more of accuracy. Trial 2 shows that decreasing the accuracy by a magnitude of 10 cuts the convergence time by around 13%; still inferior to TPRR.

One might think of further enhancing the convergence time by increasing α, the proportionality variable to the normalized torque vector of the resultant joint velocity, as in Trials 3 and 4. Results show that given a desired and relatively low accuracy, solely increasing α, would definitely yield a chance to cut the time much more as in Trial 4; again, still inferior to TPRR. However, Trial 3 proves that using this tuning method would not guarantee convergence if accuracy is increased. The reason for this non-convergence is mainly overshooting, a problem that has been observed experimentally. When accuracy is increased, the algorithm tends to overshoot the goal. In application, it means that the end-effector bounces off two different values trying to reach a point that is in between. This can most definitely be addressed by adding a control to the velocity when the end-effector is significantly near the goal. This additional control, however, is not implemented in this paper.

Trials 5 and 6 show the effects of purely tuning out η, the impact or magnitude of influence of the artificial repulsive force. Both trials show that when η is increased or decreased to a value more than and less than 1, respectively, the algorithm does not converge. Again, this is an issue with overshooting.

Lastly, Trials 7 and 8 show the effects of purely tuning out ζ, the magnitude of influence of the artificial attractive force. Trial 7 shows the decreasing the value below 1 only worsens the convergence time. Intuitively, this is because the force of attraction has lesser influence, thus, the pull it provides to drive the end-effector to the goal gets weaker, and the velocity gets slower. One might attempt to increase ζ to make the attractive force more powerful so that the end-effector will be dragged faster to its goal. However, this is not the way to go since increasing the velocity of the end-effector by increasing ζ only leads to another situation of overshooting as shown in Trial 8.

While APF algorithm is significantly harder to tune, it offers the stability for $dist2obs$. As seen from table 2, in all trials, $dist2obs$ value is kept at an almost constant value. This is a good characteristic since it minimizes risk of collision in application. This comes natural with APF as opposed to TPRR.

III. Near Singularity Behavior of TPRR Algorithm

The weak points of APF method was highlighted in the previous section. While all the discussion made TPRR seem like a invulnerable method, it has its fair share of weakness. Going back to the theory, it is important to note that the TPRR algorithm relies on the pseudo inverse formulation. While singularity itself does not pose a threat because the pseudoinverse just assigns zero component to the missing DOF, a near singularity point does (Maciejewski & Klein, 1985). When a joint is near singularity, the pseudoinverse would assign an extremely big component

corresponding to that DOF. This ultimately leads to a drastic change in computed joint velocities and a very erratic behavior of the robot as shown in the following figures.

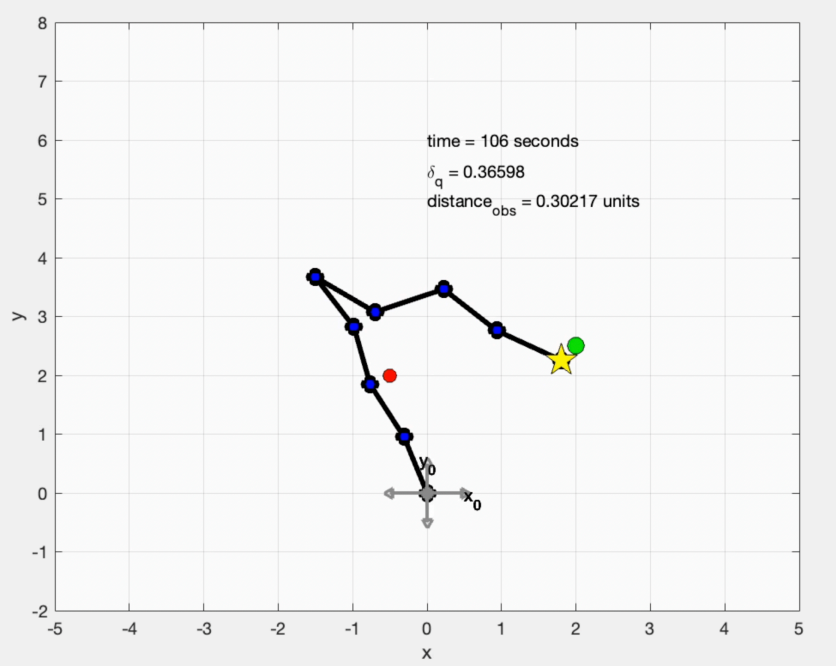

Figure 2. The figure shows joint 2 and joint 3 in a near singularity configuration (almost fully stretched out pose) at t=106 seconds.

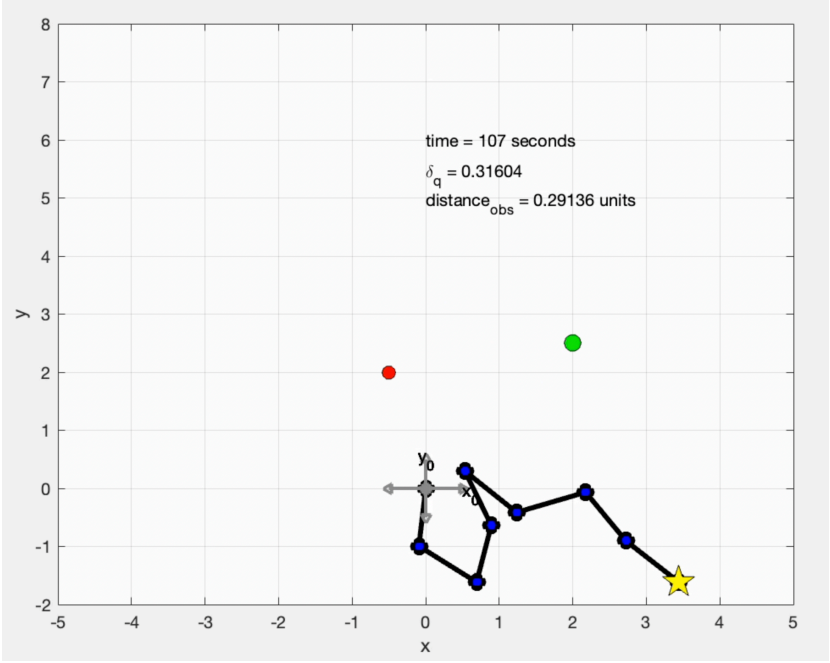

Figure 3. The figure shows the pose immediately following that of Figure 2 at t=107 seconds. This represents the erratic behavior resulting from the drastic change in computed joint velocities.

This behavior can of course be addressed by limiting the values of the joint velocities by effectively evaluating the rank of the Jacobian and setting the thresholds. This control technique, however, is not part and is not implemented in this paper.

CONCLUSION

The TPRR algorithm has three tuning parameters namely, (1) the error tolerance and the convergence criterion ε, (2) the magnitude of influence of the secondary task over the primary task α_1, and (3) the magnitude of influence of the secondary task velocity α_2. Tuning the three mentioned parameters to get the best result is practically easy in TPRR since every combination has an extremely high rate of convergence. Despite this very advantageous characteristic, the robot must not be operated near singularities to avoid non-convergence issues that results from the nature of the pseudoinverse formulation. Unless an control or limiting feature is added to the algorithm for regions near singularities, just avoiding it would be the only solution.

The APF algorithm has four main tuning parameters namely, (1) the error tolerance and the convergence criterion ε, (2) the magnitude of influence of the attractive force ζ, (3) the magnitude of influence of the repulsive force η, and (4) the proportionality variable to the resultant torque α. In contrast to TPRR, the APF algorithm is relatively more difficult to tune. One variation of one parameter does not guarantee convergence due to issues of overshooting. Also, when the desired accuracy is increased and the algorithm is convergent, it is most certain that the convergence time will greatly suffer. However, the APF algorithm has been proven to be perform well in keeping the distance of the closest point in the arm to the obstacle at an almost constant magnitude. This ensures obstacle-avoidance at all times.

REFERENCES

1. Maciejewski, A. A., & Klein, C. A. (1985). Obstacle Avoidance for Kinematically Redundant Manipulators in Dynamically Varying Environments. *The International Journal of Robotics Research*, 4(3), 109–117. https://doi.org/10.1177/027836498500400308
2. Spong, M. W., Hutchinson S., & Vidyasagar M. (2006). *Robot Modeling and Control*. John Wiley & Sons, Inc.

APPENDIX

I. Task Priority Redundancy Resolution – Source Code

A. Main Script – *driver.m*

```
% this drirver script implements the Task Priority Resolution algorithm
% to path planning of 8-DOF Robotic Arm
clc;
close all;
clear;

%% define constants and variables

% initial and final conditions
qs = (pi/180)*[90 90 -90 90 -90 90 -90 0]';
[p0,frames] = dirkin(qs);
pgoal = [2 2.5]';
delta_q = inf;

% constant
obs = [-0.5 2]';

% tuning parameters
eps = 10^-2; % convergence criterion
radius = 0.3; % radius of influence of obstacle
dt = 1; % time step

% current configuration and position
qcur = qs;
pcur = p0;

% for movieVector
ind = 1;
time = 0;
dist2obstacle = inf;

%% for animation
fig1 = figure(1);


while delta_q > eps
    clf
    % perform direct kinematics
    [pcur, frames] = dirkin(qcur);

    n = length(frames);

    % form the top priority task Jacobian (main Jacobian)
    J1 = [cross([0;0;1],frames(1:3,4,n))];
    for i = 1:n-1
        J1(1:3,i+1) = [cross(frames(1:3,3,i),(frames(1:3,4,n)-
frames(1:3,4,i)))];
```

```
    end

    % extract only the translational part of the main Jacobian
    J1 = J1(1:2,:);

    % extract the secondary jacobian (2nd task) & the closest point
    [J2,closest2obs] = sense(qcur,obs,frames,radius);

    % draw the robot and plot the expected trajectory line
    drawrobot(qcur)
    hold on
    plot(obs(1),obs(2),'-ok','MarkerSize',8,'MarkerFaceColor','r');
    hold on
    plot(pgoal(1),pgoal(2),'-ok','MarkerSize',10,'MarkerFaceColor','g');
    hold on
    text(0,6,strcat('time =',{' '},num2str(time),' seconds'),"FontSize",10);
    hold on
    text(0,5.4,strcat('\delta_q =',{' '},num2str(delta_q)),"FontSize",10);
    hold on
    text(0,4.9,strcat('distance_o_b_s =',{' '},num2str(dist2obstacle),...
        ' units'),"FontSize",10);
    hold on

    % store frames into vector for rendering
    movieVector(ind) = getframe(fig1, [30 10 500 400]);

    % compute dq based on solutions in Prob3-Q3
    [dq,delta_q] = RRA_TaskPrio(pcur,pgoal,obs,J1,J2,closest2obs);

    % update the current configuration
    qcur = qcur + dq*dt;

    % update global values
    ind = ind + 1;
    time = time + dt;
    dist2obstacle = norm(closest2obs-obs);

end

% pause the final video frame for t/framerate seconds
t = 30;
for i = 1:t
    clf
    drawrobot(qcur)
    hold on
    plot(obs(1),obs(2),'-ok','MarkerSize',8,'MarkerFaceColor','r');
    hold on
    plot(pgoal(1),pgoal(2),'-ok','MarkerSize',10,'MarkerFaceColor','g');
    hold on
    text(0,6,strcat('time =',{' '},num2str(time),' seconds'),"FontSize",10);
    hold on
    text(0,5.4,strcat('\delta_q =',{' '},num2str(delta_q)),"FontSize",10);
    hold on
    text(0,4.9,strcat('distance_o_b_s =',{' '},num2str(dist2obstacle),...
        ' units'),"FontSize",10);
```

```
    hold on
end

hold off
% create video file
myWriter = VideoWriter('tprr-test-7-demo', 'MPEG-4');
myWriter.FrameRate = 10;
open(myWriter);
writeVideo(myWriter,movieVector);
close(myWriter);
```

B. Function – *dirkin.m*

```
function [x, frames] = dirkin(q)
%DIRKIN Direct Kineematics of 4DOF planar robot
%   Input:
%       q - destination robot configuration (in degrees)
%   Output:
%       x - gripper position vector

% constants
a = 1;
a1 = a;
a2 = a;
a3 = a;
a4 = a;
a5 = a;
a6 = a;
a7 = a;
a8 = a;

% DH Table in Matrix form (theta, d, a, alpha)
DH_Mat = [q(1) 0 a1 0;
    q(2) 0 a2 0;
    q(3) 0 a3 0;
    q(4) 0 a4 0;
    q(5) 0 a5 0;
    q(6) 0 a6 0;
    q(7) 0 a7 0;
    q(8) 0 a8 0];


% extract row size of DH_Mat
m = size(DH_Mat,1);

% initialie frames
frames = [];

temp = [];

% lambda function to convert compute homogeneous matrix
HMat = @(theta, d, a, alpha)...
    [cos(theta) -sin(theta)*cos(alpha) sin(theta)*sin(alpha) a*cos(theta);...
    sin(theta)  cos(theta)*cos(alpha) -cos(theta)*sin(alpha) a*sin(theta);...
```

```
    0 sin(alpha) cos(alpha) d;...
    0 0 0 1];

% convert each row of DH_Mat to corrresponding homogeneous matrix
for i = 1:m
    Row = DH_Mat(i,:);
    cur = HMat(Row(1),Row(2),Row(3),Row(4));
    if isempty(frames)
        frames(:,:,i) = cur;
    else
        frames(:,:,i) = temp*cur;
    end
    temp = frames(:,:,i);
end

x = frames(1:2,4,m);

end
```

C. Function – *drawrobot.m*

```
function drawrobot(q)
%DRAWROBOT Draw the 4DOF Planar Arm
%   Input:
%       q - configuration (thetas in degrees)
%   Output:
%       plot of the robotic arm pose

% run dirkin to generate frames
[~,frames] = dirkin(q);

% joint positions vector
pos = [0;0];
for i=1:length(frames)
    pos = [pos frames(1:2,4,i)];
end

% plot
for i = 1:length(pos)-1
    % plot frame 0 (world axis)
    plot([0 0.5],[0 0],'->','color','#808080','LineWidth', 2,'MarkerSize',5)
    plot([0 -0.5],[0 0],'-<','color','#808080','LineWidth',
2,'MarkerSize',5);
    plot([0 0],[0 0.5],'-^','color','#808080','LineWidth', 2,'MarkerSize',5);
    plot([0 0],[0 -0.5],'-v','color','#808080','LineWidth',
2,'MarkerSize',5);

    % plot the arm
    plot([pos(1,i) pos(1,i+1)],[pos(2,i) pos(2,i+1)], '-ok', 'LineWidth', ...
        3,'MarkerSize', 8, 'MarkerFaceColor', 'blue')
    if(i == length(pos)-1)
        plot(pos(1,i+1),pos(2,i+1), 'pk','MarkerSize', 20, ...
            'MarkerFaceColor', 'y')
    end
```

```
    text(0.5,-0.1,'x_0')
    text(-0.1,0.5,'y_0')
    hold on
end
hold off
grid on
xlabel('x')
ylabel('y')
xlim([-5 5])
ylim([-2 8])

end
```

D. Function – *sense.m*

```
function [J2,c] = sense(q,obs,frames,radius)
%SENSE Range Sensor
%   Input
%       q - current configuration
%       obs - obstacle cooridnates
%       frames - frames of the current q (output form dirkin)
%       radius - radius of the region of influence of the obstacle
%   Output
%       J2 - secondary Jacobian
%       c - coordinates of the closest point in the arm to the obstacle

% define resolution of arm points
dx = 0.01;

% link lengths
% constants
a = 1;
a1 = a;
a2 = a;
a3 = a;
a4 = a;
a5 = a;
a6 = a;
a7 = a;
a8 = a;

% startpoints of each link
l1_strt_pt = [0 0];
l2_strt_pt = [frames(1:2,4,1)];
l3_strt_pt = [frames(1:2,4,2)];
l4_strt_pt = [frames(1:2,4,3)];
l5_strt_pt = [frames(1:2,4,4)];
l6_strt_pt = [frames(1:2,4,5)];
l7_strt_pt = [frames(1:2,4,6)];
l8_strt_pt = [frames(1:2,4,7)];
l_strt_pts = {l1_strt_pt l2_strt_pt l3_strt_pt l4_strt_pt l5_strt_pt ...
              l6_strt_pt l7_strt_pt l8_strt_pt};
```

```
% Jacobian formula for points along each link
J_l1 = @(len_o) [-len_o*sin(q(1)) 0 0 0 0 0 0 0;...
                 len_o*cos(q(1)) 0 0 0 0 0 0 0];
J_l2 = @(len_o) [-(len_o*sin(q(1)+q(2))+a1*sin(q(1))) ...
                 -len_o*sin(q(1)+q(2)) 0 0 0 0 0 0;...
                  (len_o*cos(q(1)+q(2))+a1*cos(q(1)))  ...
                  len_o*cos(q(1)+q(2)) 0 0 0 0 0 0];
J_l3 = @(len_o) [-
(len_o*sin(q(1)+q(2)+q(3))+a2*sin(q(1)+q(2))+a1*sin(q(1))) ...
                 -(len_o*sin(q(1)+q(2)+q(3))+a2*sin(q(1)+q(2))) ...
                 -len_o*sin(q(1)+q(2)+q(3)) ...
                 0 0 0 0 0;...

(len_o*cos(q(1)+q(2)+q(3))+a2*cos(q(1)+q(2))+a1*cos(q(1)))  ...
                 (len_o*cos(q(1)+q(2)+q(3))+a2*cos(q(1)+q(2)))  ...
                 len_o*cos(q(1)+q(2)+q(3)) ...
                 0 0 0 0 0];
J_l4 = @(len_o) [-
(len_o*sin(q(1)+q(2)+q(3)+q(4))+a3*sin(q(1)+q(2)+q(3))+a2*sin(q(1)+q(2))+a1*s
in(q(1))) ...
                 -
(len_o*sin(q(1)+q(2)+q(3)+q(4))+a3*sin(q(1)+q(2)+q(3))+a2*sin(q(1)+q(2))) ...
                 -(len_o*sin(q(1)+q(2)+q(3)+q(4))+a3*sin(q(1)+q(2)+q(3))) ...
                 -(len_o*sin(q(1)+q(2)+q(3)+q(4))) ...
                  0 0 0 0;...

(len_o*cos(q(1)+q(2)+q(3)+q(4))+a3*cos(q(1)+q(2)+q(3))+a2*cos(q(1)+q(2))+a1*c
os(q(1)))  ...

(len_o*cos(q(1)+q(2)+q(3)+q(4))+a3*cos(q(1)+q(2)+q(3))+a2*cos(q(1)+q(2)))  ..
.
                  (len_o*cos(q(1)+q(2)+q(3)+q(4))+a3*cos(q(1)+q(2)+q(3))) ...
                  (len_o*cos(q(1)+q(2)+q(3)+q(4))) ...
                  0 0 0 0];
J_l5 = @(len_o) [-
(len_o*sin(q(1)+q(2)+q(3)+q(4)+q(5))+a4*sin(q(1)+q(2)+q(3)+q(4))+a3*sin(q(1)+
q(2)+q(3))+a2*sin(q(1)+q(2))+a1*sin(q(1))) ...
                 -
(len_o*sin(q(1)+q(2)+q(3)+q(4)+q(5))+a4*sin(q(1)+q(2)+q(3)+q(4))+a3*sin(q(1)+
q(2)+q(3))+a2*sin(q(1)+q(2))) ...
                 -
(len_o*sin(q(1)+q(2)+q(3)+q(4)+q(5))+a4*sin(q(1)+q(2)+q(3)+q(4))+a3*sin(q(1)+
q(2)+q(3))) ...
                 -
(len_o*sin(q(1)+q(2)+q(3)+q(4)+q(5))+a4*sin(q(1)+q(2)+q(3)+q(4))) ...
                 -(len_o*sin(q(1)+q(2)+q(3)+q(4)+q(5))) ...
                  0 0 0;...

(len_o*cos(q(1)+q(2)+q(3)+q(4)+q(5))+a4*cos(q(1)+q(2)+q(3)+q(4))+a3*cos(q(1)+
q(2)+q(3))+a2*cos(q(1)+q(2))+a1*cos(q(1))) ...

(len_o*cos(q(1)+q(2)+q(3)+q(4)+q(5))+a4*cos(q(1)+q(2)+q(3)+q(4))+a3*cos(q(1)+
q(2)+q(3))+a2*cos(q(1)+q(2))) ...
```

```
(len_o*cos(q(1)+q(2)+q(3)+q(4)+q(5))+a4*cos(q(1)+q(2)+q(3)+q(4))+a3*cos(q(1)+
q(2)+q(3))) ...

(len_o*cos(q(1)+q(2)+q(3)+q(4)+q(5))+a4*cos(q(1)+q(2)+q(3)+q(4))) ...
                 (len_o*cos(q(1)+q(2)+q(3)+q(4)+q(5))) ...
                 0 0 0];

J_l6 = @(len_o) [-
(len_o*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6))+a5*sin(q(1)+q(2)+q(3)+q(4)+q(5))+a4
*sin(q(1)+q(2)+q(3)+q(4))+a3*sin(q(1)+q(2)+q(3))+a2*sin(q(1)+q(2))+a1*sin(q(1
))) ...
                -
(len_o*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6))+a5*sin(q(1)+q(2)+q(3)+q(4)+q(5))+a4
*sin(q(1)+q(2)+q(3)+q(4))+a3*sin(q(1)+q(2)+q(3))+a2*sin(q(1)+q(2))) ...
                -
(len_o*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6))+a5*sin(q(1)+q(2)+q(3)+q(4)+q(5))+a4
*sin(q(1)+q(2)+q(3)+q(4))+a3*sin(q(1)+q(2)+q(3))) ...
                -
(len_o*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6))+a5*sin(q(1)+q(2)+q(3)+q(4)+q(5))+a4
*sin(q(1)+q(2)+q(3)+q(4))) ...
                -
(len_o*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6))+a5*sin(q(1)+q(2)+q(3)+q(4)+q(5))) .
..
                -(len_o*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6))) ...
                 0 0;...

(len_o*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6))+a5*cos(q(1)+q(2)+q(3)+q(4)+q(5))+a4
*cos(q(1)+q(2)+q(3)+q(4))+a3*cos(q(1)+q(2)+q(3))+a2*cos(q(1)+q(2))+a1*cos(q(1
))) ...

(len_o*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6))+a5*cos(q(1)+q(2)+q(3)+q(4)+q(5))+a4
*cos(q(1)+q(2)+q(3)+q(4))+a3*cos(q(1)+q(2)+q(3))+a2*cos(q(1)+q(2))) ...

(len_o*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6))+a5*cos(q(1)+q(2)+q(3)+q(4)+q(5))+a4
*cos(q(1)+q(2)+q(3)+q(4))+a3*cos(q(1)+q(2)+q(3))) ...

(len_o*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6))+a5*cos(q(1)+q(2)+q(3)+q(4)+q(5))+a4
*cos(q(1)+q(2)+q(3)+q(4))) ...

(len_o*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6))+a5*cos(q(1)+q(2)+q(3)+q(4)+q(5))) .
..
                 (len_o*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6))) ...
                 0 0];

J_l7 = @(len_o) [-
(len_o*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7))+a6*sin(q(1)+q(2)+q(3)+q(4)+q(5
)+q(6))+a5*sin(q(1)+q(2)+q(3)+q(4)+q(5))+a4*sin(q(1)+q(2)+q(3)+q(4))+a3*sin(q
(1)+q(2)+q(3))+a2*sin(q(1)+q(2))+a1*sin(q(1))) ...
                -
(len_o*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7))+a6*sin(q(1)+q(2)+q(3)+q(4)+q(5
)+q(6))+a5*sin(q(1)+q(2)+q(3)+q(4)+q(5))+a4*sin(q(1)+q(2)+q(3)+q(4))+a3*sin(q
(1)+q(2)+q(3))+a2*sin(q(1)+q(2))) ...
                -
(len_o*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7))+a6*sin(q(1)+q(2)+q(3)+q(4)+q(5
```

```
)+q(6))+a5*sin(q(1)+q(2)+q(3)+q(4)+q(5))+a4*sin(q(1)+q(2)+q(3)+q(4))+a3*sin(q
(1)+q(2)+q(3))) ...
                -
(len_o*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7))+a6*sin(q(1)+q(2)+q(3)+q(4)+q(5
)+q(6))+a5*sin(q(1)+q(2)+q(3)+q(4)+q(5))+a4*sin(q(1)+q(2)+q(3)+q(4))) ...
                -
(len_o*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7))+a6*sin(q(1)+q(2)+q(3)+q(4)+q(5
)+q(6))+a5*sin(q(1)+q(2)+q(3)+q(4)+q(5))) ...
                -
(len_o*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7))+a6*sin(q(1)+q(2)+q(3)+q(4)+q(5
)+q(6))) ...
                -(len_o*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7))) ...
                0;...

(len_o*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7))+a6*cos(q(1)+q(2)+q(3)+q(4)+q(5
)+q(6))+a5*cos(q(1)+q(2)+q(3)+q(4)+q(5))+a4*cos(q(1)+q(2)+q(3)+q(4))+a3*cos(q
(1)+q(2)+q(3))+a2*cos(q(1)+q(2))+a1*cos(q(1))) ...

(len_o*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7))+a6*cos(q(1)+q(2)+q(3)+q(4)+q(5
)+q(6))+a5*cos(q(1)+q(2)+q(3)+q(4)+q(5))+a4*cos(q(1)+q(2)+q(3)+q(4))+a3*cos(q
(1)+q(2)+q(3))+a2*cos(q(1)+q(2))) ...

(len_o*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7))+a6*cos(q(1)+q(2)+q(3)+q(4)+q(5
)+q(6))+a5*cos(q(1)+q(2)+q(3)+q(4)+q(5))+a4*cos(q(1)+q(2)+q(3)+q(4))+a3*cos(q
(1)+q(2)+q(3))) ...

(len_o*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7))+a6*cos(q(1)+q(2)+q(3)+q(4)+q(5
)+q(6))+a5*cos(q(1)+q(2)+q(3)+q(4)+q(5))+a4*cos(q(1)+q(2)+q(3)+q(4))) ...

(len_o*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7))+a6*cos(q(1)+q(2)+q(3)+q(4)+q(5
)+q(6))+a5*cos(q(1)+q(2)+q(3)+q(4)+q(5))) ...

(len_o*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7))+a6*cos(q(1)+q(2)+q(3)+q(4)+q(5
)+q(6))) ...
                (len_o*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7))) ...
                0];

J_l8 = @(len_o) [-
(len_o*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7)+q(8))+a7*sin(q(1)+q(2)+q(3)+q(4
)+q(5)+q(6)+q(7))+a6*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6))+a5*sin(q(1)+q(2)+q(3)
+q(4)+q(5))+a4*sin(q(1)+q(2)+q(3)+q(4))+a3*sin(q(1)+q(2)+q(3))+a2*sin(q(1)+q(
2))+a1*sin(q(1))) ...
                -
(len_o*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7)+q(8))+a7*sin(q(1)+q(2)+q(3)+q(4
)+q(5)+q(6)+q(7))+a6*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6))+a5*sin(q(1)+q(2)+q(3)
+q(4)+q(5))+a4*sin(q(1)+q(2)+q(3)+q(4))+a3*sin(q(1)+q(2)+q(3))+a2*sin(q(1)+q(
2))) ...
                -
(len_o*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7)+q(8))+a7*sin(q(1)+q(2)+q(3)+q(4
)+q(5)+q(6)+q(7))+a6*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6))+a5*sin(q(1)+q(2)+q(3)
+q(4)+q(5))+a4*sin(q(1)+q(2)+q(3)+q(4))+a3*sin(q(1)+q(2)+q(3))) ...
                -
(len_o*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7)+q(8))+a7*sin(q(1)+q(2)+q(3)+q(4
)+q(5)+q(6)+q(7))+a6*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6))+a5*sin(q(1)+q(2)+q(3)
+q(4)+q(5))+a4*sin(q(1)+q(2)+q(3)+q(4))) ...
```

```
                _
(len_o*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7)+q(8))+a7*sin(q(1)+q(2)+q(3)+q(4
)+q(5)+q(6)+q(7))+a6*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6))+a5*sin(q(1)+q(2)+q(3)
+q(4)+q(5))) ...
                _
(len_o*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7)+q(8))+a7*sin(q(1)+q(2)+q(3)+q(4
)+q(5)+q(6)+q(7))+a6*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6))) ...
                _
(len_o*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7)+q(8))+a7*sin(q(1)+q(2)+q(3)+q(4
)+q(5)+q(6)+q(7))) ...
                -(len_o*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7)+q(8)));...

(len_o*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7)+q(8))+a7*cos(q(1)+q(2)+q(3)+q(4
)+q(5)+q(6)+q(7))+a6*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6))+a5*cos(q(1)+q(2)+q(3)
+q(4)+q(5))+a4*cos(q(1)+q(2)+q(3)+q(4))+a3*cos(q(1)+q(2)+q(3))+a2*cos(q(1)+q(
2))+a1*cos(q(1))) ...

(len_o*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7)+q(8))+a7*cos(q(1)+q(2)+q(3)+q(4
)+q(5)+q(6)+q(7))+a6*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6))+a5*cos(q(1)+q(2)+q(3)
+q(4)+q(5))+a4*cos(q(1)+q(2)+q(3)+q(4))+a3*cos(q(1)+q(2)+q(3))+a2*cos(q(1)+q(
2))) ...

(len_o*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7)+q(8))+a7*cos(q(1)+q(2)+q(3)+q(4
)+q(5)+q(6)+q(7))+a6*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6))+a5*cos(q(1)+q(2)+q(3)
+q(4)+q(5))+a4*cos(q(1)+q(2)+q(3)+q(4))+a3*cos(q(1)+q(2)+q(3))) ...

(len_o*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7)+q(8))+a7*cos(q(1)+q(2)+q(3)+q(4
)+q(5)+q(6)+q(7))+a6*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6))+a5*cos(q(1)+q(2)+q(3)
+q(4)+q(5))+a4*cos(q(1)+q(2)+q(3)+q(4))) ...

(len_o*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7)+q(8))+a7*cos(q(1)+q(2)+q(3)+q(4
)+q(5)+q(6)+q(7))+a6*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6))+a5*cos(q(1)+q(2)+q(3)
+q(4)+q(5))) ...

(len_o*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7)+q(8))+a7*cos(q(1)+q(2)+q(3)+q(4
)+q(5)+q(6)+q(7))+a6*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6))) ...

(len_o*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7)+q(8))+a7*cos(q(1)+q(2)+q(3)+q(4
)+q(5)+q(6)+q(7))) ...
                (len_o*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7)+q(8)))];

J_l = {J_l1 J_l2 J_l3 J_l4 J_l5 J_l6 J_l7 J_l8};

arm_points = [];

% all points along a1, dx apart
for i=0:dx:a1
    arm_points(end+1,:) = [i*cos(q(1)) i*sin(q(1)) 1];
end

% all points along a2, dx apart
for i=dx:dx:a2
    arm_points(end+1,:) = [i*cos(q(1)+q(2))+a1*cos(q(1)) ...
        i*sin(q(1)+q(2))+a1*sin(q(1)) 2];
end
```

```
% all points along a3, dx apart
for i=dx:dx:a3
    arm_points(end+1,:) =
[i*cos(q(1)+q(2)+q(3))+a2*cos(q(1)+q(2))+a1*cos(q(1)) ...
        i*sin(q(1)+q(2)+q(3))+a2*sin(q(1)+q(2))+a1*sin(q(1)) 3];
end

% all points along a4, dx apart
for i=dx:dx:a4
    arm_points(end+1,:) =
[i*cos(q(1)+q(2)+q(3)+q(4))+a3*cos(q(1)+q(2)+q(3))+a2*cos(q(1)+q(2))+a1*cos(q
(1)) ...

i*sin(q(1)+q(2)+q(3)+q(4))+a3*sin(q(1)+q(2)+q(3))+a2*sin(q(1)+q(2))+a1*sin(q(
1)) 4];
end

% all points along a5, dx apart
for i=dx:dx:a5
    arm_points(end+1,:) =
[i*cos(q(1)+q(2)+q(3)+q(4)+q(5))+a4*cos(q(1)+q(2)+q(3)+q(4))+a3*cos(q(1)+q(2)
+q(3))+a2*cos(q(1)+q(2))+a1*cos(q(1)) ...

i*sin(q(1)+q(2)+q(3)+q(4)+q(5))+a4*sin(q(1)+q(2)+q(3)+q(4))+a3*sin(q(1)+q(2)+
q(3))+a2*sin(q(1)+q(2))+a1*sin(q(1)) 5];
end

% all points along a6, dx apart
for i=dx:dx:a6
    arm_points(end+1,:) =
[i*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6))+a5*cos(q(1)+q(2)+q(3)+q(4)+q(5))+a4*cos
(q(1)+q(2)+q(3)+q(4))+a3*cos(q(1)+q(2)+q(3))+a2*cos(q(1)+q(2))+a1*cos(q(1)) .
..

i*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6))+a5*sin(q(1)+q(2)+q(3)+q(4)+q(5))+a4*sin(
q(1)+q(2)+q(3)+q(4))+a3*sin(q(1)+q(2)+q(3))+a2*sin(q(1)+q(2))+a1*sin(q(1))
6];
end

% all points along a7, dx apart
for i=dx:dx:a7
    arm_points(end+1,:) =
[i*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7))+a6*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(
6))+a5*cos(q(1)+q(2)+q(3)+q(4)+q(5))+a4*cos(q(1)+q(2)+q(3)+q(4))+a3*cos(q(1)+
q(2)+q(3))+a2*cos(q(1)+q(2))+a1*cos(q(1)) ...

i*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7))+a6*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6
))+a5*sin(q(1)+q(2)+q(3)+q(4)+q(5))+a4*sin(q(1)+q(2)+q(3)+q(4))+a3*sin(q(1)+q
(2)+q(3))+a2*sin(q(1)+q(2))+a1*sin(q(1)) 7];
end

% all points along a8, dx apart
for i=dx:dx:a8
```

```
    arm_points(end+1,:) =
[i*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7)+q(8))+a7*cos(q(1)+q(2)+q(3)+q(4)+q(
5)+q(6)+q(7))+a6*cos(q(1)+q(2)+q(3)+q(4)+q(5)+q(6))+a5*cos(q(1)+q(2)+q(3)+q(4
)+q(5))+a4*cos(q(1)+q(2)+q(3)+q(4))+a3*cos(q(1)+q(2)+q(3))+a2*cos(q(1)+q(2))+
a1*cos(q(1)) ...

i*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6)+q(7)+q(8))+a7*sin(q(1)+q(2)+q(3)+q(4)+q(5
)+q(6)+q(7))+a6*sin(q(1)+q(2)+q(3)+q(4)+q(5)+q(6))+a5*sin(q(1)+q(2)+q(3)+q(4)
+q(5))+a4*sin(q(1)+q(2)+q(3)+q(4))+a3*sin(q(1)+q(2)+q(3))+a2*sin(q(1)+q(2))+a
1*sin(q(1)) 8];
end


% test the point closest to the obstacle
closest2obs = [];

for i = 1:length(arm_points)
    if isempty(closest2obs) || ...
            (norm(closest2obs(1:2)-obs) > norm(arm_points(i,1:2)'-obs))
        closest2obs = arm_points(i,:)';
    end
end

% get link number for closest point to obs
l_no_1 = closest2obs(3);

if norm(closest2obs(1:2)-obs) <= radius
    J2 = J_l{l_no_1}(norm(closest2obs(1:2)-l_strt_pts{l_no_1}));
else
    J2 = zeros(2,length(q));
end

% coordinate of the closest point to the obstacle
c = closest2obs(1:2);
```

E. Function – *RRA_TaskPrio.m*

```
function [q_dot,delta_p] = RRA_TaskPrio(pc,pd,obs,J1,J2,closest2obs)
%RRA_TASKPRIO RESOLVED RATES ALGORIHTM FOR TASK PRIORITY RESOLUTION
%   Input:
%       pc - current position coordinates
%       pd - destination position coordinates
%       obs - obstacle coordinates
%       J1 - Jacobian of the first task
%       J2 - Jacobian of the secondary task
%       closest2obs - position coordinates of the closest point in the arm
%               wrt to the obstacle
%       radius - radius of the region of influence of the obstacle
%   Output:
%       qd_dot - rate of change of configuration
%       delta_p - currrent position error norm

% tuning parameters
alpha1 = 10;%1/norm(closest2obs-obs);
```

```
alpha2 = 1;

% For the task priority resolution formula
I_size = size(J1,2);
J1_tilda = J2*(eye(I_size,I_size)-pinv(J1)*J1);

% resolved rates algorithm parameters (translational)
eps_p = 0.001;
vmin = 0.001;
vmax = 0.05;
lambda_p = 50;

% error parameters
delta_p = norm(pd-pc);

% normalized position error vector
nhat = (pd - pc)/norm(pd-pc);

% conditions for vmag (linear velocity magnitude)
if  delta_p/eps_p >  lambda_p
    vmag = vmax;
else
    vmag = vmin + ((vmax-vmin)*(delta_p-eps_p))/(eps_p*(lambda_p-1));
end
x_dot = vmag*nhat;

% direction of closest point to obstacle
nhat2 = -(obs'-closest2obs)/norm(obs' - closest2obs);

% velocity of closest point away from the obstacle
x2_dot = vmag*nhat2;

q_dot = pinv(J1)*x_dot+alpha1*pinv(J1_tilda)*(alpha2*x2_dot-
J2*pinv(J1)*x_dot);

end
```

II. Artificial Potential Field Method – Source Code

A. Main Script – *driver.m*

```
% this drirver script implements the Artificial Potential Field algorithm
% to path planning of 8-DOF Robotic Arm
clc;
close all;
clear;

%% define constants and variables

% initial and final conditions
qs = (pi/180)*[90 90 -90 90 -90 90 -90 0]';
[p0,frames] = dirkin(qs);
```

```
pgoal = [2 2.5]';
delta_q = inf;

% constant
obs = [0.5 2]';

% tuning parameters
eps = 10^-2; % convergence criterion
radius = 0.3; % radius of influence of obstacle
dt = 1; % time step

% current configuration and position
qcur = qs;
pcur = p0;

% for movieVector
ind = 1;
time = 0;
dist2obstacle = inf;

%% for animation
fig1 = figure(1);

while delta_q > eps
    clf
    % perform direct kinematics
    [pcur, frames] = dirkin(qcur);

    n = length(frames);

    % form the top priority task Jacobian (main Jacobian)
    J1 = [cross([0;0;1],frames(1:3,4,n))];
    for i = 1:n-1
        J1(1:3,i+1) = [cross(frames(1:3,3,i),(frames(1:3,4,n)-
frames(1:3,4,i)))];
    end

    % extract only the translational part of the main Jacobian
    J1 = J1(1:2,:);

    [J2,closest2obs] = sense(qcur,obs,frames,radius);

    % draw the robot and plot the expected trajectory line
    drawrobot(qcur)
    hold on
    plot(obs(1),obs(2),'-ok','MarkerSize',8,'MarkerFaceColor','r');
    hold on
    plot(pgoal(1),pgoal(2),'-ok','MarkerSize',10,'MarkerFaceColor','g');
    hold on
    text(0,6,strcat('time =',{' '},num2str(time),' seconds'),"FontSize",10);
    hold on
    text(0,5.4,strcat('\delta_q =',{' '},num2str(delta_q)),"FontSize",10);
    hold on
    text(0,4.9,strcat('distance_o_b_s =',{' '},num2str(dist2obstacle),...
        ' units'),"FontSize",10);
```

```
    hold on

    % store frames into vector for rendering
    movieVector(ind) = getframe(fig1, [30 10 500 400]);

    % compute dq based on solutions in Prob3-Q3
    [dq,delta_q] = APF(pcur,pgoal,obs,J1,J2,closest2obs,radius);

    % update the current configuration
    qcur = qcur + dq*dt;

    % update global values
    ind = ind + 1;
    time = time + dt;
    dist2obstacle = norm(closest2obs-obs);

end

% pause the final video frame for t/framerate seconds
t = 30;
for i = 1:t
    clf
    drawrobot(qcur)
    hold on
    plot(obs(1),obs(2),'-ok','MarkerSize',8,'MarkerFaceColor','r');
    hold on
    plot(pgoal(1),pgoal(2),'-ok','MarkerSize',10,'MarkerFaceColor','g');
    hold on
    text(0,6,strcat('time =',{' '},num2str(time),' seconds'),"FontSize",10);
    hold on
    text(0,5.4,strcat('\delta_q =',{' '},num2str(delta_q)),"FontSize",10);
    hold on
    text(0,4.9,strcat('distance_o_b_s =',{' '},num2str(dist2obstacle),...
        ' units'),"FontSize",10);
    hold on
    movieVector(ind) = getframe(fig1, [30 10 500 400]);
    ind=ind+1;
end

hold off
% create video file
myWriter = VideoWriter(['apf-test9-vs-tprr-test7-demo'], 'MPEG-4');
myWriter.FrameRate = 10;
open(myWriter);
writeVideo(myWriter,movieVector);
close(myWriter);
```

B. Function – *dirkin.m* (Same as Part I B)

C. Function – *drawrobot.m* (Same as Part I C)

D. Function – *sense.m* (Same as Part I D)

E. Function – *APF.m*

```
function [q_dot,delta_p] = APF(pc,pd,obs,J1,J2,closest2obs,rho_o)
% APF ARTIFICIAL POTENTIAL FIELD ALGORITHM
%   Input:
%       pc - current position coordinates
%       pd - destination position coordinates
%       obs - obstacle coordinates
%       J1 - Jacobian of the first task
%       J2 - Jacobian of the secondary task
%       closest2obs - position coordinates of the closest point in the arm
%               wrt to the obstacle
%       rho_o - radius of the region of influence of the obstacle
%   Output:
%       qd_dot - rate of change of configuration
%       delta_p - currrent position error norm

% tuning parameters
zeta = 1; % gain of the attractive force
eta = 1; % gain of the repulsive force
alpha = 0.005;% gain (<1)

% error parameters
delta_p = norm(pd-pc);

% Force of attraction formula (negative gradient of U_att)
% d is always assumed to be infinite, hence,
F_att = -zeta*(pc-pd);

rho_p = norm(closest2obs-obs);
rho_p_grad = (closest2obs-obs)/norm(closest2obs-obs);

if rho_p <= rho_o
    F_rep = (eta*((1/rho_p)-(1/rho_o))/(rho_p^2))*rho_p_grad;
else
    F_rep = [0 0]';
end

tau = J1'*F_att+J2'*F_rep;
q_dot = alpha*tau/norm(tau);

end
```