Robotic Systems Engineering CW2

Raymond Danks

November 2019

1 Q1

An example of this 2D 4DOF Manipulator can be seen below:

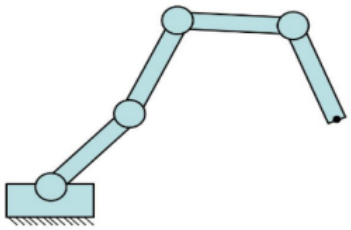

Figure 1: 4DOF 2D Manipulator

It should be noted that this question asks about the **Pose** of the end effector, meaning both the orientation and the position, not just its position. Also note that a solution to an inverse kinematics problem is merely a position/overall pose of the robotic manipulator.
Since the pose of the end effector must remain constant, the final link must be fixed in place, as otherwise the end effector's position and/or orientation would change. For this final link to remain fixed in position, the final joint must be fixed in position too.
In order for the end effector to maintain the correct position, each of the middle two joints can only be in a position where they are mirrored between the linear line connecting the current joint to the next joint. This is shown in a more intuitive manner in the diagram below:

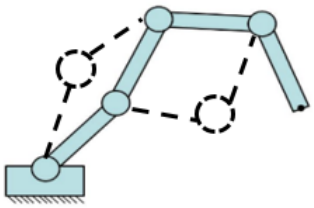

Figure 2: Possible positions of 4DOF manipulator

The dotted lines are possible link positions and the dotted circles are possible joint positions. These (along with the given position of course) are the only positions which will correspond to the same end effector pose - notice how each joint is flipped over the line of symmetry previously outlined in the previous paragraph. Any combination of these positions can be used (eg any of the dotted positions or the positions shown by the actual image) to reach the same end effetcor position. If a joint was put into a position which was not along this line of symmetry (some textbooks call this 'inverted'), then the position of the end effector would move.

Taking into account all of the possibilities outlined above, there are **four** possible solutions to the majority of poses of the end effector of a 2D , 4DOF arm. However, there is one exception case. This is the case where all of the links of the manipulator align (ie the joint angles are all the same, with respect to the world frame) and hence the manipulator is at "full extension". In this case, the distance from the base of the robot to the end effector is at its maximum (which is defined as the edge of the robot workspace). In this case, there is only one solution to a given end effector pose, as any other solution would not reach the maximum distance between the base of the robot and the end effector.

2 Q2

Two of the main criteria come from the physical constraints of the robot - this is a good way of discarding solutions.

Firstly, all solutions outside of the robot workspace should be discarded. The robot workspace is defined as the maximum radius from the base of the robot where the robot could reach. For example, if a robot has four links, each of length 1m, then an inverse kinematics solution which is 5m away from the base of the robot can never be reached and therefore these solutions (and similar ones) should be discarded.

Secondly, solutions which are not reachable due to the joint limits of the robot should be discarded. Revolute joints usually cannot fully rotate 360°, due to the mechanical constraints of the joints themselves and hence if an inverse kinematic solution would require these joint limits to be exceeded, the solution should be discarded.
Now that the main two methods for discarding solutions have been specified, some methods for picking form the multiple remaining solutions may be specified:

Any solution which would cause a singularity should be avoided. This will cause trouble for the robot in later operation (its DOF will be reduced), even if the robot can physically achieve the inverse kinematics solution. A singularity may occur when two joints are aligned and hence reducing the degrees of freedom of the robot. Inverse kinematic solutions which would cause a singularity should be avoided, even if they may be physically achievable.

The most optimal solution is likely the solution which requires the robot's joints to move the least from its starting position. This solution will be the most efficient. Out of the available inverse kinematic solutions, the one closest to the robot's starting position should be selected as it will be the quickest and most efficient (disregarding obstacle avoidance inputs etc).

3 Q3

This comes down to the quadrants of the common three trigonometric functions: sin, cosine and tan. These can be seen below (numbered 1-4):

since $\tan(x) = \frac{sin(x)}{cos(x)}$, if tan(x) gives a negative result, this may be because either sin(x) is in its negative quadrants (3 and 4), whilst cos(x) is in its positive quadrants(1 and 4) or vice versa. Similarly when tan(x) is positive, this could be because sin(x) is in its positive quadrants (1 and 2) and cos(x)

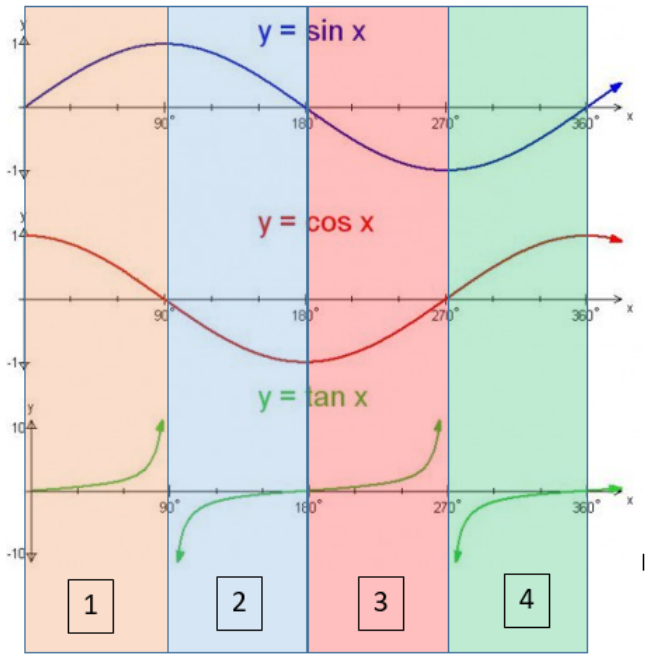

Figure 3: Trigonometric Quadrants (adapted from attached source[1])

is in its positive quadrants (1 and 4) or they are both in their negative quadrants.
This is the issue with atan($\frac{a}{b}$). Information about x and y are lost (their signs, whether they are positive or negative) because of the fraction. In this case, the combination of $a = atan(\frac{a}{b}) = atan(\frac{-a}{-b})$. atan($\frac{a}{b}$) actually simplifies down the whole system to the first and fourth quadrant - if the outcome is positive then the first quadrant is assumed, and if the outcome is negative then the fourth quadrant is assumed. This ignores the possible solutions whihc lay in the second and third quadrant.
atan2(y,x) solves this problem (the method will not be delved into as it is outside the scope of the question). Hence, in the cases when sin(x) and/or cos(x) lay inside the second and third quadrant, the solution to atan2(y,x) will be different to the solution to atan($\frac{x}{y}$ (ie when either, or both, of the respective angles used to calculate sin(x) and cos(x) are between 90 degrees and 270 degrees).

4 Q4

4.1 Q4b

Using a MATLAB script (which the author created for this task - please see the attached .m file), the symbolic forward kinematics equation describing the transformation from the base of the robot to the end effector can be seen as follows (note that all non zero lengths have been replaced with their repective variables).

$$ {}^{0}T_{5} = \begin{bmatrix} s(t1)*s(t5)+c(t234)*c(t1)*c(t5) & c(t234)*c(t1)*s(t5)-c(t5)*s(t1) & s(t234)*c(t1) & -c(t1)*(a1-a3*s(t23)-a2*s(t2)+a4*c(t234))-d5*s(t234)*c(t1) \\ c(t234)*c(t5)*s(t1)-c(t1)*s(t5) & c(t1)*c(t5)+c(t234)*s(t1)*s(t5) & s(t234)*s(t1) & -s(t1)*(a1-a3*s(t23)-a2*s(t2)+a4*c(t234))-d5*s(t234)*s(t1) \\ -s(t234)*c(t5) & -s(t234)*s(t5) & c(t234) & d1+a3*c(t23)+a2*c(t2)-d5*c(t234)+a4*s(t234) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1} $$

Where s = sin, c = cos, t1 = $theta_1$, t123 = $theta_1 + theta_2 + theta_3$ for example.
In order to derive a general form inverse kinematics solution, this matrix should be compared to the general form transformation matrix, which would represent a desired pose as a transformation from the base to the end effector:

$$ {}^{0}T_{5} = \begin{bmatrix} r11 & r12 & r13 & Pex \\ r21 & r22 & r23 & Pey \\ r31 & r32 & r33 & Pez \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2} $$

Now the elements between these two matrices can be compared in order to find equations for thetas 1-5 (hence the joint angles required). All of these equations can be defined as follows, for later use:

$$ r11 = s(t1)s(t5) + c(t234)c(t1)c(t5) \tag{3} $$

$$ r12 = c(t234)c(t1)s(t5) - c(t5)s(t1) \tag{4} $$

$$ r13 = s(t234)c(t1) \tag{5} $$

$$ r21 = c(t234)c(t5)s(t1) - c(t1)s(t5) \tag{6} $$

$$ r22 = c(t1)c(t5) + c(t234)s(t1)s(t5) \tag{7} $$

$$ r23 = s(t234)s(t1) \tag{8} $$

$$ r31 = -s(t234)c(t5) \tag{9} $$

$$r32 = -s(t234)s(t5) \tag{10}$$

$$r33 = c(t234) \tag{11}$$

$$Pex = -c(t1)(a1 - a3s(t23) - a2s(t2) + a4c(t234)) - d5s(t234)c(t1) \tag{12}$$

$$Pey = -s(t1)(a1 - a3s(t23) - a2s(t2) + a4c(t234)) - d5s(t234)s(t1) \tag{13}$$

$$Pez = d1 + a3c(t23) + a2c(t2) - d5c(t234) + a4s(t234) \tag{14}$$

Therefore, if the following division can be conducted in order to define tan(t1):

$$\frac{r23}{r13} = \frac{s(t234)s(t1)}{s(t234)c(t1)} = tan(t1) \tag{15}$$

Therefore, t1 can be defined as follows:

$$t1 = atan2(r13, r23) \tag{16}$$

Similarly, tan(t5) can be defined by the following equation:

$$\frac{r32}{r31} = \frac{-s(t234)s(t5)}{-s(t234)c(t5)} = tan(t5) \tag{17}$$

Therefore, t5 can be defined as follows:

$$t5 = atan2(r31, r32) \tag{18}$$

Now, it is time to involve the position elements, starting with Pez - the equation for Pez can be seen at the start of this question, substituting in the equations for r33 and r23 leads to the following:

$$Pez = d1 + a3c(t23) + a2c(t2) - d5r33 + \frac{a4r23}{s(t1)} \tag{19}$$

Therefore, making the unknowns the subjects of the equation:

$$Pez - d1 + d5r33 - \frac{a4r23}{s(t1)} = a3c(t23) + a2c(t2) = A \tag{20}$$

Notice how we have labelled this equation as A, so that it can be recalled later. Now involving Pey - looking at the given equation for Pey, using r33 and r32 the following substitutions can be made:

$$Pey = -s(t1)(a1 - a3s(t23) - a2s(t2) + a4r33 - \frac{d5r32}{s(t5)}) \tag{21}$$

Therefore, making the unknowns the subjects of the equation:

$$\frac{Pey}{s(t1)} + a1 + a4r33 - \frac{d5r32}{s(t5)} = a3s(t23) + a2s(t2) = B \tag{22}$$

Notice how this equation is labelled as B so it can be recalled later.

Now, if the equation $A^2 + B^2$ is calculated, this leads to the following:

$$A^2 + B^2 = (a3c(t23) + a2c(t2))^2 + (a3s(t23) + a2s(t2))^2 \tag{23}$$

Therefore,

$$A^2+B^2 = a3^2c^2(t23)+a3^2s^2(t23)+a2^2c^2(t2)+a2^2s^2(t2)+2a3c(t23)a2c(t2)+2a3s(t23)a2s(t2) \tag{24}$$

Substituting the equations $sin^2(t2) + cos^2(t2) = 1$ (and the same equation for t23) and cos(t23-t2) = cos(t23)cos(t2)+sin(t23)sin(t2), along with some factorising leads to the following equation:

$$A^2 + B^2 = a3^2 + a2^2 + 2a3a2cos(t3) \tag{25}$$

Therefore,

$$t3 = acos(\frac{A^2 + B^2 - a3^2 - a2^2}{2a3a2}) \tag{26}$$

Where A and B are previously defined, known equations.

Now, going back to equation A:

$$A = Pez - d1 + d5r33 - \frac{a4r23}{s(t1)} = a3c(t23) + a2c(t2) \tag{27}$$

substituting in the equation: $c(t23) = c(t2 + t3) = c(t2)c(t3) - s(t2)s(t3)$ leads to the following equation for A:

$$A = a3(c(t2)c(t3) - s(t2)s(t3)) + a2c(t2) \tag{28}$$

Therefore, rearranging for the unknowns leads to:

$$c(t2)(a3c(t3) + a2) - s(t2)(a3s(t3)) \tag{29}$$

Now if it can be defined that:

$$w1 = a3c(t3) + a2 \tag{30}$$

$$w2 = a3s(t3) \tag{31}$$

Notice that both w1 and w2 are fully KNOWN since an equation for theta3 has been previously derived (above). Therefore, this means that equation A can be rewritten as:

$$A = c(t2)w1 - s(t2)w2 \tag{32}$$

Now, if the following triangle is defined:

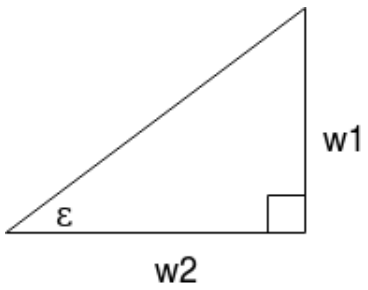

Figure 4: Right Angled Triangle

Then the angle specified, ϵ can be defined as:

$$tan(\epsilon) = \frac{w2}{w1} \tag{33}$$

Therefore,

$$\epsilon = atan2(w1, w2) \tag{34}$$

Now ϵ is a fully known value, since it is formed of known variables, w1 and w2. IF we now define the equation:

$$\frac{B}{\sqrt{w1^2 + w2^2}} = \frac{c(t2)w1 + s(t2)w2}{\sqrt{w1^2 + w2^2}} = c(t2)\frac{w1}{\sqrt{w1^2 + w2^2}} - s(t2)\frac{w2}{\sqrt{w1^2 + w2^2}} \tag{35}$$

Notice how the denominator of this equation is the hypotenuse of the given triangle. Now by using **trigonometry on the defined triangle** ($\cos(\epsilon) = \frac{w1}{hypotenuse}$ and $sin(\epsilon) = \frac{w2}{hypotenuse}$), the following equation can be derived (by recognising the hypotenuse too, as previously defined/mentioned):

$$\frac{B}{\sqrt{w1^2 + w2^2}} = c(t2)c(\epsilon) - s(t2)s(\epsilon) \tag{36}$$

Now, using the general identity previously noted for $cos(Part1 + Part2)$, this equation can be rewritten as follows:

$$\frac{B}{\sqrt{w1^2 + w2^2}} = c(t2 + \epsilon) \tag{37}$$

Therefore, an equation for theta2 can be derived as:

$$t2 = acos(\frac{B}{\sqrt{w1^2 + w2^2}}) - \epsilon \tag{38}$$

Where B, w1, w2 and ϵ are all previously defined, fully known values.

Now, looking at the equation for r33 once more:

$$r33 = c(t234) = cos(t2 + t3 + t4) \tag{39}$$

Since t2 and t3 are no longer unknowns in this equation (as we have derived expressions for them), an equation for the only unknown, t4, can be derived as follows:

$$t4 = acos(r33) - t2 - t3 \tag{40}$$

Now closed form expressions for all 5 of the theta angles have been found and hence the question is complete.

5 Q5

5.1 Q5a

A Straight Line Path trajectory model has been implemented in the code for Q5a. Since the data from the bag is joint angles, it makes sense to do one's analysis within the joint space, rather than the cartesian space. In the joint space, the straight line path trajectory model can be defined as:

$$q(t) = q_1 + t(q_2 - q_1) \tag{41}$$

$$\frac{dq}{dt} = q_2 - q_1 \tag{42}$$

Where q is the vector of joint angles (in the Youbot's case, this has 5 angles within it). This analysis is valid for when the robot is moving from joint position q_1 to q_2. Note that in the above equations, t is a normalised time between 0 and 1.
For a more robust and usable definition of the straight line path trajectory model in the youbot, do see the code - however, this question is about why the straight line path trajectory model has been chosen.

The first reason that the straight line trajectory model was chosen was due to its simplicity. Computationally, the straight line model is by far one of the least demanding trajectory models. Moreover, it is far more easily adjusted than other models as the only factor to change is the amount of "steps" between the given checkpoints, instead of changing a repertoire of other variables. This heavily increases the generability of the code. THis is especially useful when the integrity of the code, and its ability to produce the same results when run on different PCs, is important. In this case, this is extremely important (since the coursework must run on the assessor's pc) and hence an easily adjustable and robust (rather than tentative) trajectory model has been chosen - there is no need to overcomplicate a working model when the only criteria is to reach the given checkpoints at the specified times.

The second reason is due to the configuration of the Youbot itself. The youbot is a grasping arm and hence its workspace is a hemisphere around its base (with radius equal to the total arm length). This means that the workspace of the youbot is convex. This means that the straight line path between any two points (which are within the robot's workspace) will always

be within the robot's workspace. This gives the straight line path an advantage over other, more complex trajectory models as if a pair of points were right at the edge of the workspace, the path may instruct the robot to leave the workspace, causing both physical and computational errors.

Moreover, the straight line path method is also a very efficient method when it comes to distance travelled. Put succinctly, the shortest distance between two points is the line which is in between the two points, hence to minimise the distance travelled by the robot (and to increase efficiency), the straight line path trajectory model has been used.

It should be noted and addressed that the straight line path trajectory model can cause singularities within the robot. However, equivocating this coursework to an industrial task, it can be seen that in Q4e, a method for calculating whether a given joint trajectory will cause a singularity is defined. Hence, this should not be a worry in this use case - in an industrial application of this code, this singularity checker should be implemented into the straight line trajectory model and if one of the points from the straight line model should is predicted to create a singularity, a perturbation of the path could be in built to the trajectory plan. This previous work means that the advantages of the straight line trajectory model far outweighs the disadvantages, which is why it has been chosen for this use case.

5.2 Q5b

A different algorithm than the algorithms described in the lecture has been used to find the shortest path in this case. Many of the 'given' algorithms (such as Djikstra's) find the most efficient way to get from point A to point B through the given nodes, regardless of how many nodes it uses. There is a few problems with this - firstly, it assumes that it is known in which position the robot should end up; this takes away 1 degree of freedom which could be optimised (by assigning an end location). Moreover, this is not a true exploration of the travelling salesman problem. The generic travelling salesman problem is where the agent (in this case our robot) must visit each node, with the shortest possible distance and then return to its start location. The problem which was modelled during this task was an adjusted version of the travelling salesman problem, where the agent does not need to return to its start position at the end of the journey.

There are many general methods and algorithms to solve the determine the most efficient path within a travelling salesman problem, **however**, it should be asked whether this use case needs a general solution. The bag only contains 5 checkpoints which the robot arm must reach. This means that there are 120 possible combinations of checkpoints which the arm could take (5! = 120). This is a completely reasonable number to compute. General solutions are usually used in situations where either the actual code must be used for another use case or the amount of nodes (note that in this use case, nodes = checkpoints) is high enough that the total number of combinations is high enough that it would be too computationally intensive (for example if there were 8 joints, 8! = 40,320 combinations). However, neither of these conditions is the case for this use case. Hence, the brute force approach to the modified travelling salesman problem has been used. The diagram below shows the tree diagram which describes an example coombination within the algorithm:

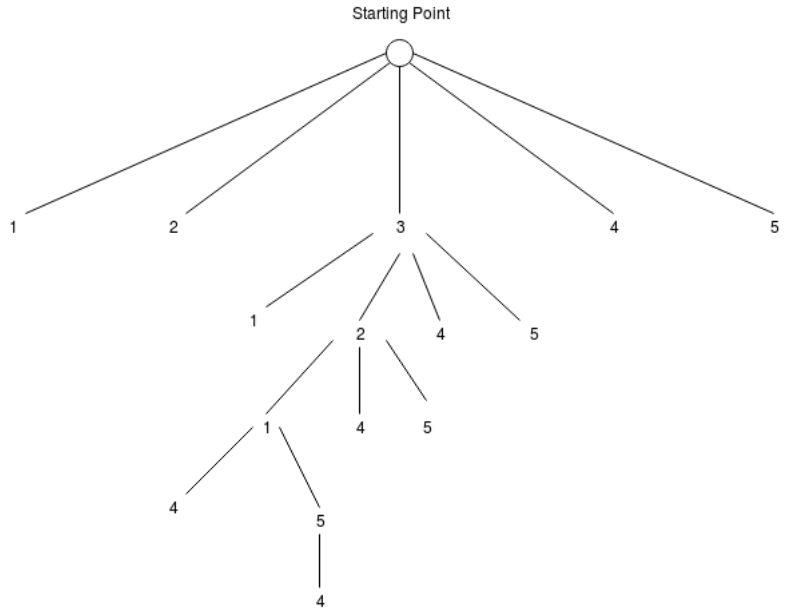

Figure 5: Example Combination: 3-2-1-5-4

By brute force, it is meant that every single combination of the 5 checkpoints (always starting from the starting point, which is NOT a checkpoint) has been calculated. For each of the movements between checkpoints, a euclidean distance in the cartesian space (x,y,z) has been calculated and for each combination, these distances are added up in order to give an overall path length derived from that specific combination.

It should be noted that this algorithm could be improved with the inclusion of the rotation at each checkpoint (and calculating this as an extra distance, increaing the accuracy of the algorithm), although it has not been in this case.

Once an overall path length for every combination has been derived, the minimum path length is selected, along with the combination/order of checkpoints which results in this minimum path. This is then published onto the robot.

The advantage of this system is that it should always find the shortest path/combination available to it out of the 5 checkpoints, since it tests all of them. The only improvement that could be made to the accuracy/refinement of the algorithm could be in the inclusion of the rotation as a separate distance, to optimise the overall distance travelled by the end effector. Whilst this method could not be extrapolated to robots with a higher number of joints, this was not the task set and thus formulating a system which does this is unnecessary - a brute force method which calculates all possibilities should give a solution which is optimal over other algorithms such as Djikstra's; therefore, when the computational load is not too high (in this case it is not, with 120 different combinations), this is the optimal way to find the shortest path.

Note that the straight line path trajectory model has been used in this algorithm too. This is because the shortest distance between two points is a straight line between the two, and since the goal in this question is to reduce the total distance travelled, the straight line path trajectory model is perfect for this use case.

One of the only downsides to this travelling salesman model is that nodes/checkpoints cannot be revisited and thus if the optimal solution requires visiting of nodes,

it will not be reached in this manner. However, intuitively, this is very unlikely (virtually impossible, due to the straight line path trajectory model) to be the case where visiting a checkpoint more than once will lead to a shorter overall path and hence this model stands robust and efficient.

NOTE TO ASSESSOR: Within my code, I did not know how to find the original pose of the robot. I saw on the Moodle QA thread that the idea is that we should find the shortest path from this beginning pose, however I found no mention of this elsewhere and investigating the different ros topics proved fruitless. Moreover, the robot does not seem to move on my pc (even though I am believe I am publishing the joint angles to it correctly) and hence cannot test the robot (but I can print the trajectory). Therefore, I have ASSUMED that the joint angles all start at zero (my_pt_initial = 0,0,0,0,0). IF this is not the case, please feel free to change this section (it is marked on the code) to test the actual distance (it is printed in the terminal), since the marks in this question come from our searching algorithm. Moreover, with this assumption that the initial condition is that all joint angles are at zero, the shortest path distance is 1.35826, with the following order of checkpoints: 3 -¿ 1 -¿ 5 -¿ 4 -¿ 2 (see console for these same stats if you change the initial joint configuration).

5.3 Q5b

In order to avoid obstacles during path planning, and assimilation with the physical world should be used. An artificial field should be attached to all of the obstacles. THis field should be designed ton act similarly to that of a magnetic field (or a reverse gravitational field). Essentially, the closer which the robot gets to the manipulator, the higher the 'force' on the manipulator and the less likely that it will be to get close to the obstacle. The equation for a reverse (repulsive) gravitational field would be as follows:

$$F = \frac{GM_1M_2}{r^2} \tag{43}$$

Where G is a constant and in robotics, $M_1 and M_2$ can be replaced with other pertinent variables. THe important part of this equation is the denominator; r^2. IN this case, r is the distance between the robot's arm and the centre of the artificial field. Hence, if the arm was in the same space as the centre of the artificial field, the force would go to infinite - meaning that the arm

cannot touch the centre of the field.
This is how the robot should avoid obstacles - if should be allowed to have its pose manipulated by the force if need be. Notice that due to the inverse square law defined by the equation above, a "pure" artificial field will always impose some kind of force onto the robot (no matter how large r is) hence, in real applications, each of these fields' effects should be restricted to a certain distance from their centre.
The goal position for the robot should be denoted as an attractive force (the same equation as above with a minus in front), so that the robot can purely move around the system using the forces acting upon it as guidelines towards the goal.
Whilst the end goal itself would represent the "global minima" in a gradient-descent style optimisation, it is possible for the robot's position to get stuck in a local minima, where all possible directions of movement have a higher force than is currently imparted upon the robot and it gets stuck. In this case, the robot should perform a random walk (randomly moving its end effector, still taking into account extreme forces so as to not hit obstacles) in order to exit the local minima and try again to reach the global minima/goal position
Getting back to the actual object avoidance, it is important to note that there are two main trails of thought when it coems to assigning the centre of the artificial repulsive fields onto the obstacles. One method is to place the centre of the field in the centre of the obstacle. This would mean that the field would effectivelty model the whole obstacle as a sphere (since that would be its force sphere of influence). This means that this method works well for obstacles which are generally spherical, or can be sufficiently modelled that way. Alternatively, the centre of the artificial field could be attached to a corner/vertex of an obstacle (or anywhere along its surface). THe advantage of this is that during a random walk, the end effector should never get close to the centre of the field (the surface of the obstacle) since the force will rise quickly towards infinity - this effect is moreso than it is when the centre of the field is placed within the obstacle. However, placing multiple artificial fields at different corners/vertices/edges/surfaces of the obstacle can lead to extended interaction of these fields (and indeed with the attractive field from the final point position) and hence the number of local minima could increase.

Generally, the second method (centre of field on corners) requires extensive knowledge of the shape of the obstacles, whilst the first method (placing

the centre of the field in the centre of the obstacle) allows for a much more general use, without extensive knowledge of the object (plus, all obstacles are then modelled as spheres, which is advantageous for general solutions) and hence this is the method which should be sued for this task.

References

[1] Mr. Homer Owlcation. Trigonometry—graphing the sine, cosine and tangent functions, March 2019.