Robotics: Planning and Navigation

Assignment -2b

Aravind Narayanan - 2019102014

Link for video: [Assignment-2b]

Constant time-scaling

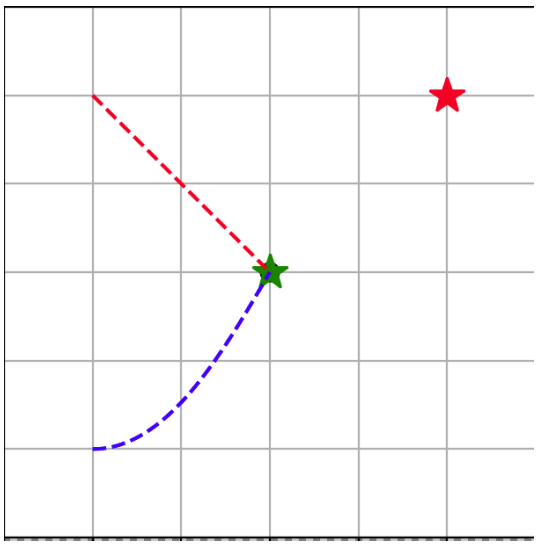
$$\dot{x}(\tau) = k\dot{x}(t)$$

Rule-based constant:

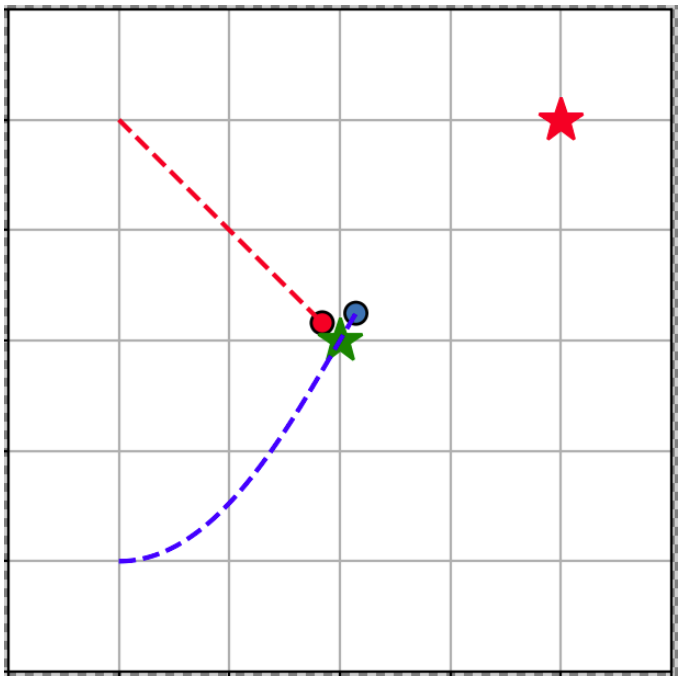
Approach: Fix a sensor radius of the robot, within which collisions with an obstacle may be detected. Scale the trajectory of the robot to avoid collisions with the obstacle once it reaches within the sensor limit by a predefined constant.

Chosen k value: 2.5218996686355557 (Chosen based the velocity limits)

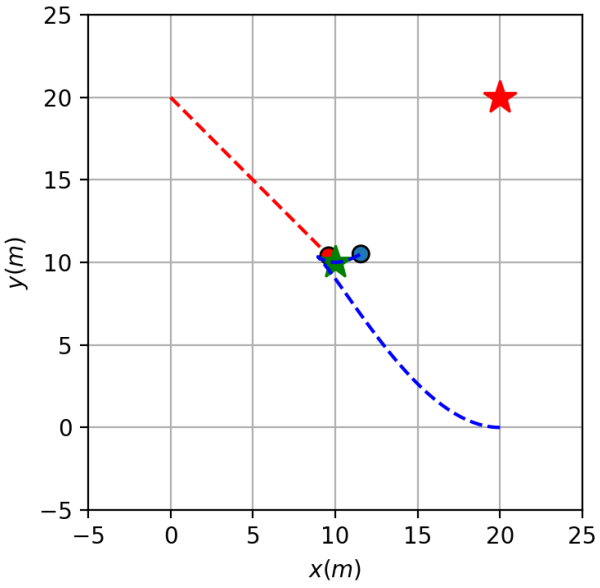
Initial Setup to show collision takes place normally:



After applying time-scaling:



Time scaling of a different start location:



RED Color: Dynamic Object

Blue Color: Robot

Red Star: Goal Location for Robot

Question: What problems do you notice with this simple approach?

In this case we only have a hard constraint on the scaling value being used which is dependent on the velocity limits etc. So this leads to a jerkier motion though it is able to avoid the obstacle. But rather in the collision cone approach we get a range of values based on the more robust velocity obstacle approach i.e. collision cones.

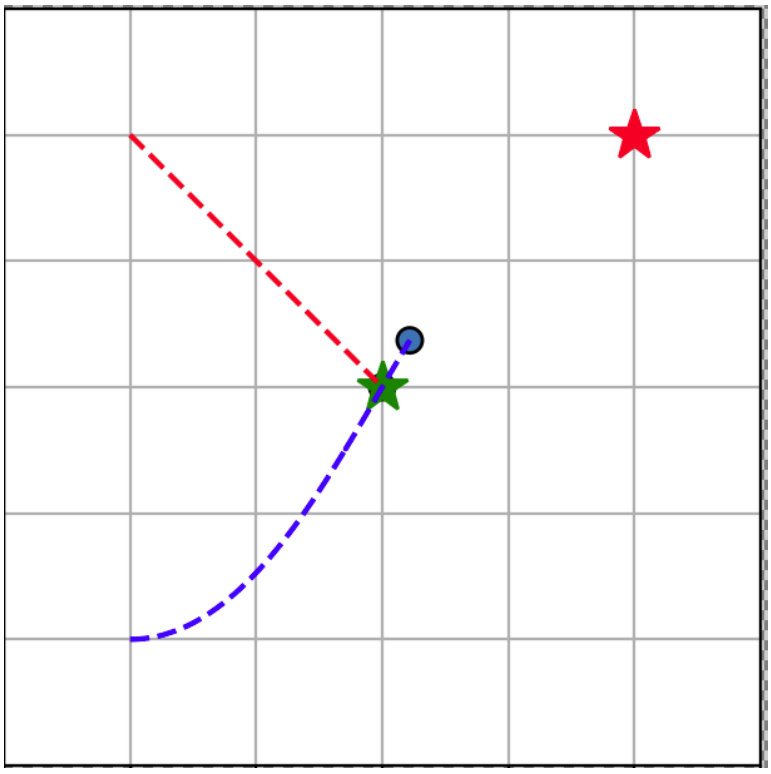
Collision-cone based:

Approach: Find a range of values for the appropriate scaling constant k based on the Velocity Obstacle approach discussed in class. As a hint, the constant should be the solution of a system of inequalities coming from the velocity limits and the collision cone. You may find the following useful: https://docs.scipy.org/doc/scipy/reference/generated/scipy. optimize.linprog.html. Choose any of the suitable constants and scale the robot trajectory to avoid collisions.

Polynomial to get range of s:

$$-((x_1 - x_2)(svx_1 - vx_2) + (y_1 - y_2)(svy_1 - vy_2))^2$$
$$+ \left((svx_1 - vx_2)^2 + (svy_1 - vy_2)^2\right)\left((x_1 - x_2)^2 + (y_1 - y_2)^2 - 1.0\right)$$

Result:

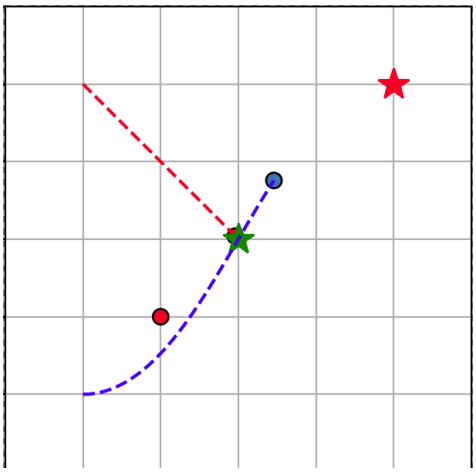


RED Color: Dynamic Object

Blue Color: Robot

Red Star: Goal Location for Robot

<u>Static Obstacle included:</u>



Linear time-scaling

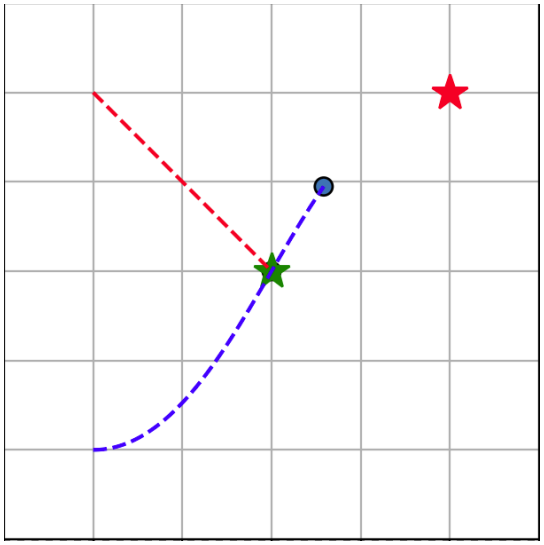
Linear time-scaling is written as:

$$\dot{x}(\tau) = s(t)\dot{x}(t)$$

Where scale s(t) linearly depends on :

$$s(t) = a + bt$$

Result:



Questions:

1. How can we ensure smoother robot trajectory using time-scaling to avoid discontinuities?

   Answer:

   In the generation of trajectory as we define the velocity limits etc, we also are able to get a trajectory optimisation where we minimize the cost function that encompasses two parts: 1) Cost function to reach the goal 2) **Cost function to ensure**

smoothness. This second cost function is dependent on velocities to generate a smoother trajectory without a lot of jerks and can be defined as follows:

$$C_{smooth} = \sum_i (v_{i+1} - v_i)^2 + \sum_i (w_{i+1} - w_i)^2$$

This way despite having to scale the trajectory for time scaling to be able to avoid obstacles, we are able to minimize it by optimizing on this cost function.

2. You have been taught Model Predictive Control for robot trajectory planning in class. Suppose you start with an initial guess for the robot trajectory. How would the trajectories given by MPC vary from that given by simple time-scaling with respect to the initial guess, given both are able to avoid collisions successfully?

   Answer:

   In MPC, we track the error continuously and model the trajectory and bring changes to it based on the waypoints being generated around the obstacle and then return back to its trajectory. But in the case of time scaling, we see that the trajectory path formed initially doesn't change but rather the avoidance is done based only on the scaling of the trajectory.

3. How would you extend time-scaling to a system of two robots trying to avoid collisions with each other?

   Answer:

   The rule base constant method has been employed here wherein the checking for collision is just one robot treating the other robot as a dynamic obstacle and vice versa by seeing if the center of object is 5m away from the robot.

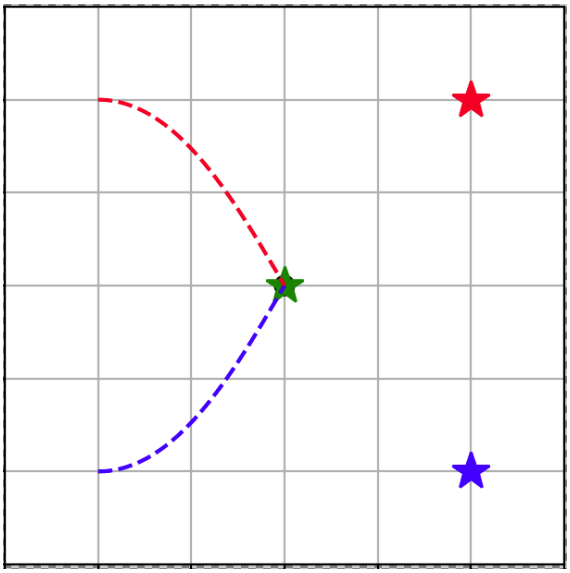
   For every robot and given its kinodynamic model and the current position and velocity of all neighboring robots, **we can compute a collision-free trajectory for the robot under the assumption that all other agents follow the same algorithm for collision avoidance** or their velocity remains constant **during the planning horizon.**

This way as both the robots implement the collision free trajectory planning, **we only have to timescale one robot faster than the other robot by choosing appropriate k value.**

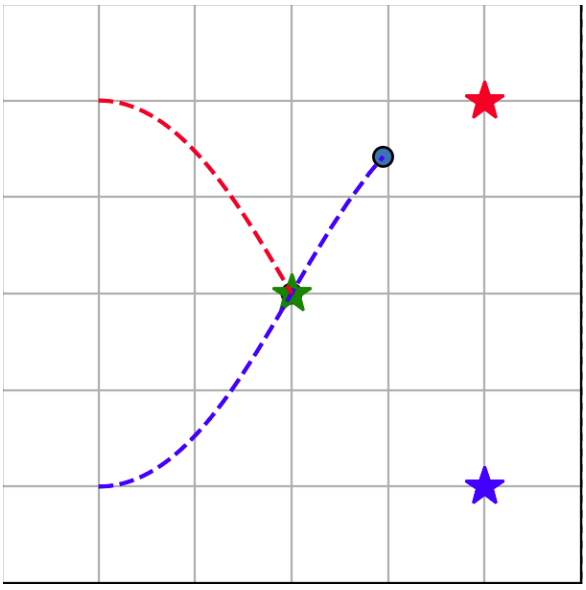
In our code example, k is taken such that the 1st robot is 16 times faster than the second robot by the virtue of time scaling.

NOTE: Code written and verified for bonus

Initial configuration of two robots to show collision will take place normally:



Result:



RED Color: Robot 2

Blue Color: Robot 1

Red Star: Goal Location for Robot 2

Blue Star: Goal Location for Robot 1