SC 627 - Motion Planning and Coordination for Autonomous Vehicles Assignment IV: Balancing Robots

Gagan Jain | 180100043

Implementation Details

The key idea is to construct a velocity control strategy to balance the robots equidistantly. In order to do this, we ignore motion in the y direction and just focus on the distances and the velocities in the x direction. More specifically, our velocity for the i^{th} robot, v_i is given by:

$$v_{ix} = k * [(x_{i+1} - x_i) - (x_i - x_{i-1})] \tag{1}$$

Here, k is a tunable parameter and x_i denotes the x coordinate of the i_{th} robot. Now, all that is left to do is define suitable callback functions and store the data from the robot's self odom as well as left and right robot odoms as well. This is done by using the nav_msgs datatype which provides access to the current pose, velocity and orientation for all the three robots, for each case. These are appropriately stored and updated in a list. The orientation (yaw angle) is retrieved using the euler_from_quaternion function. The velocity convert function is also adjusted to ignore the angular component and allow the robots to only move back and forth. The velocity computed using the above equation is fed to this function with a zero velocity in vertical direction and the current yaw angle, and we get the linear and angular velocity components, that are then fed to the publisher. This is iteratively done until we reach the termination condition, which just checks for the containment of the velocity of a robot as well as it's left and right robot's velocities within a certain threshold (0.0001). As soon as this is met, we terminate. Note that we also add a counter as the initial velocities upon spawning the robots are very small and that might make the termination condition true very early without balancing. So, we do not allow termination until we have iterated for a certain number of steps (in our case 1000). Final tuning led to choosing $k = 1$ for a smooth balancing. Finally, we just instantiate this class, the main algorithm and generate the time and path plots for each of the 6 movable robots.

Simulation results

We get the following results upon simulation:

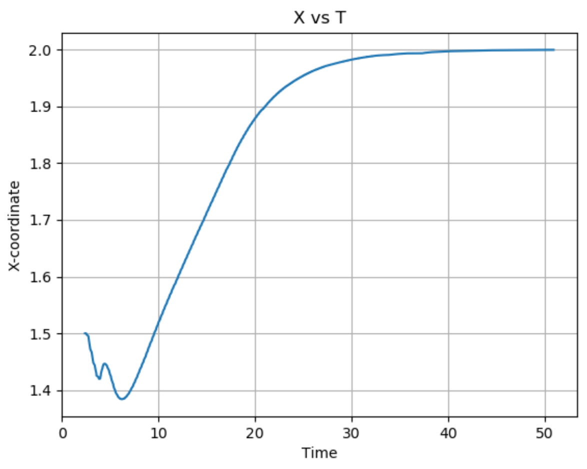

Figure 1: Robot 2: X vs Time

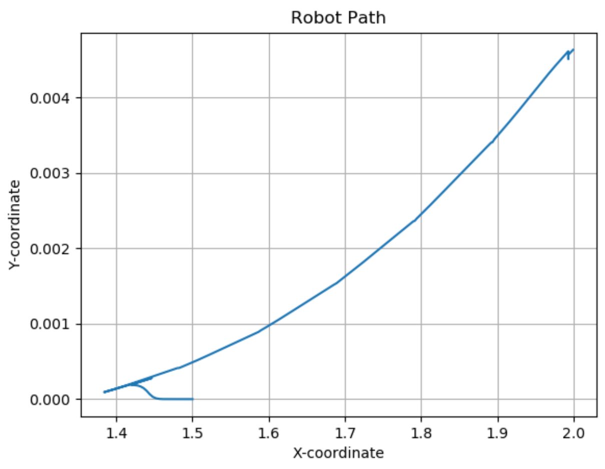

Figure 2: Robot 2: Path Plot

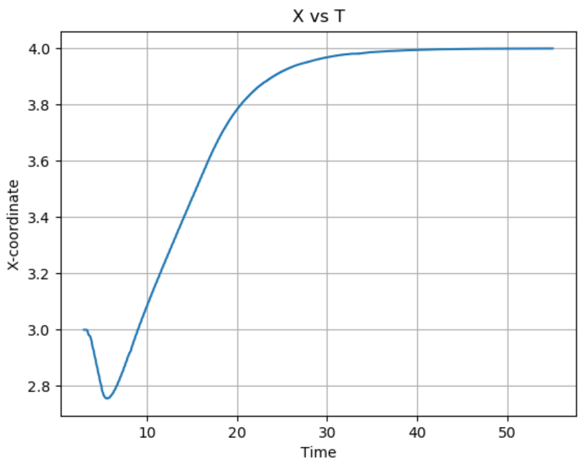

Figure 3: Robot 3: X vs Time

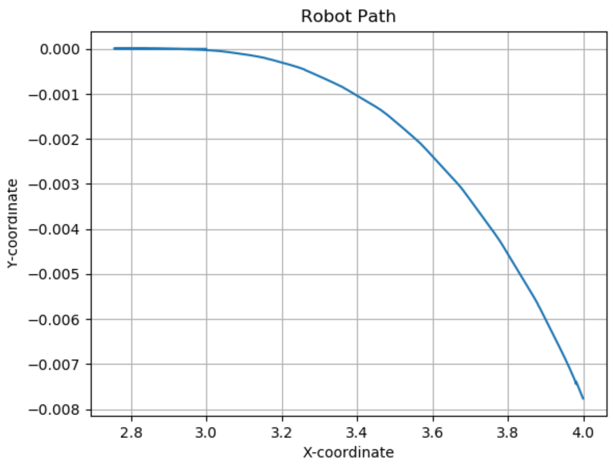

Figure 4: Robot 3: Path Plot

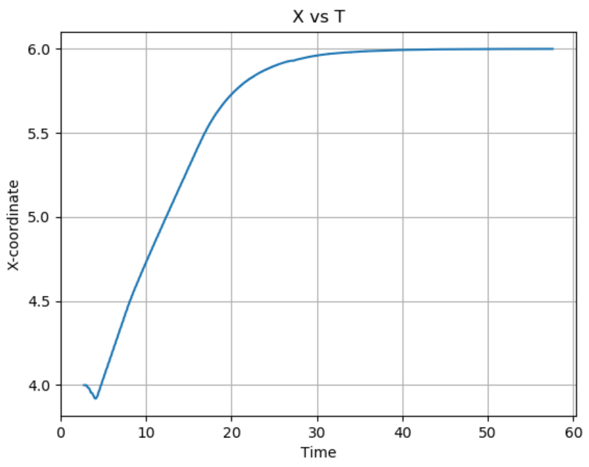

Figure 5: Robot 4: X vs Time

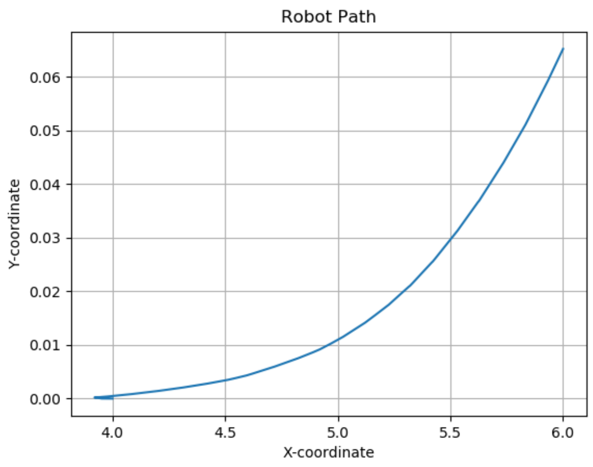

Figure 6: Robot 4: Path Plot

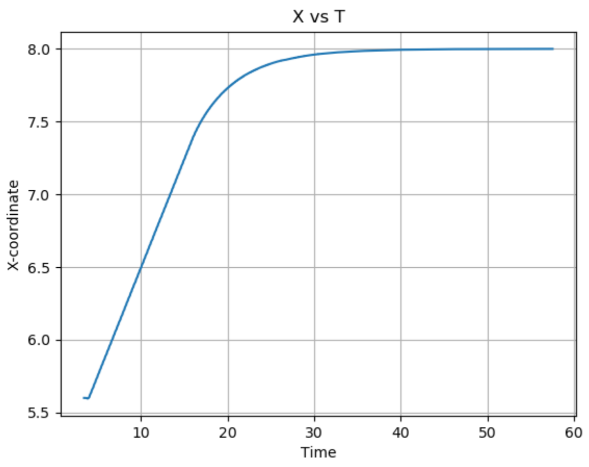

Figure 7: Robot 5: X vs Time

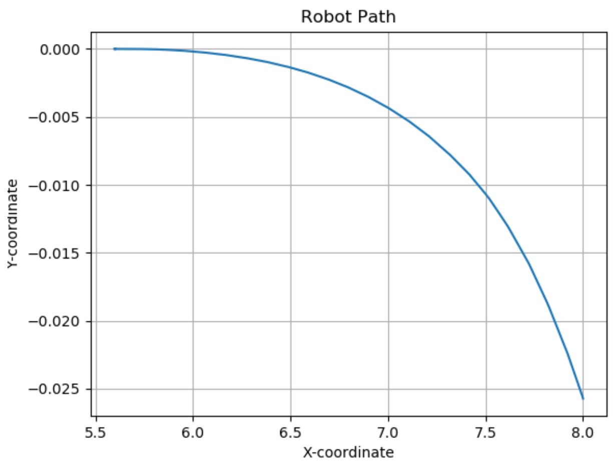

Figure 8: Robot 5: Path Plot

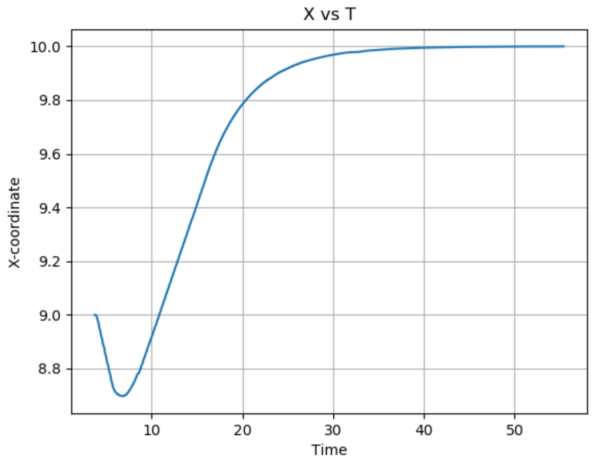

Figure 9: Robot 6: X vs Time

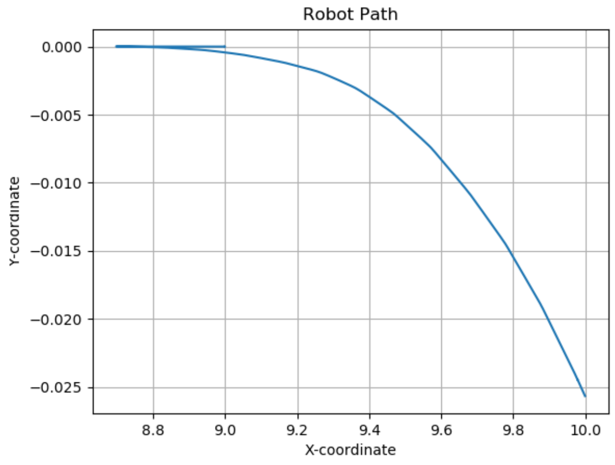

Figure 10: Robot 6: Path Plot

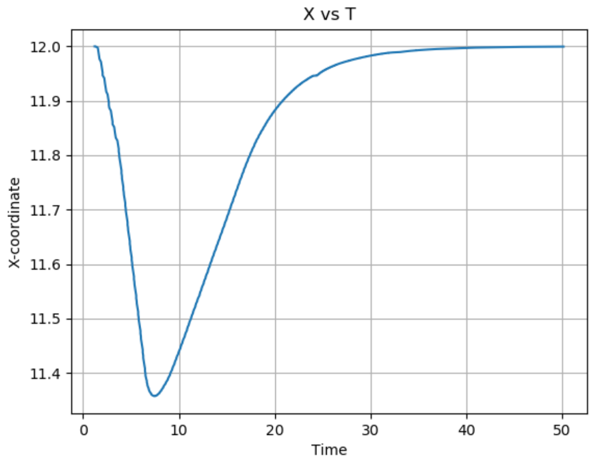

Figure 11: Robot 7: X vs Time

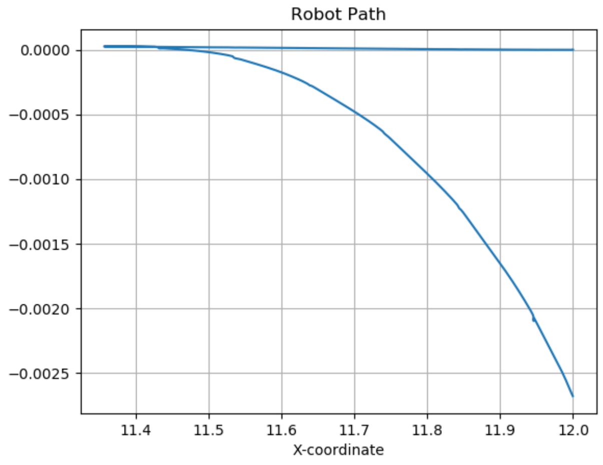

Figure 12: Robot 7: Path Plot