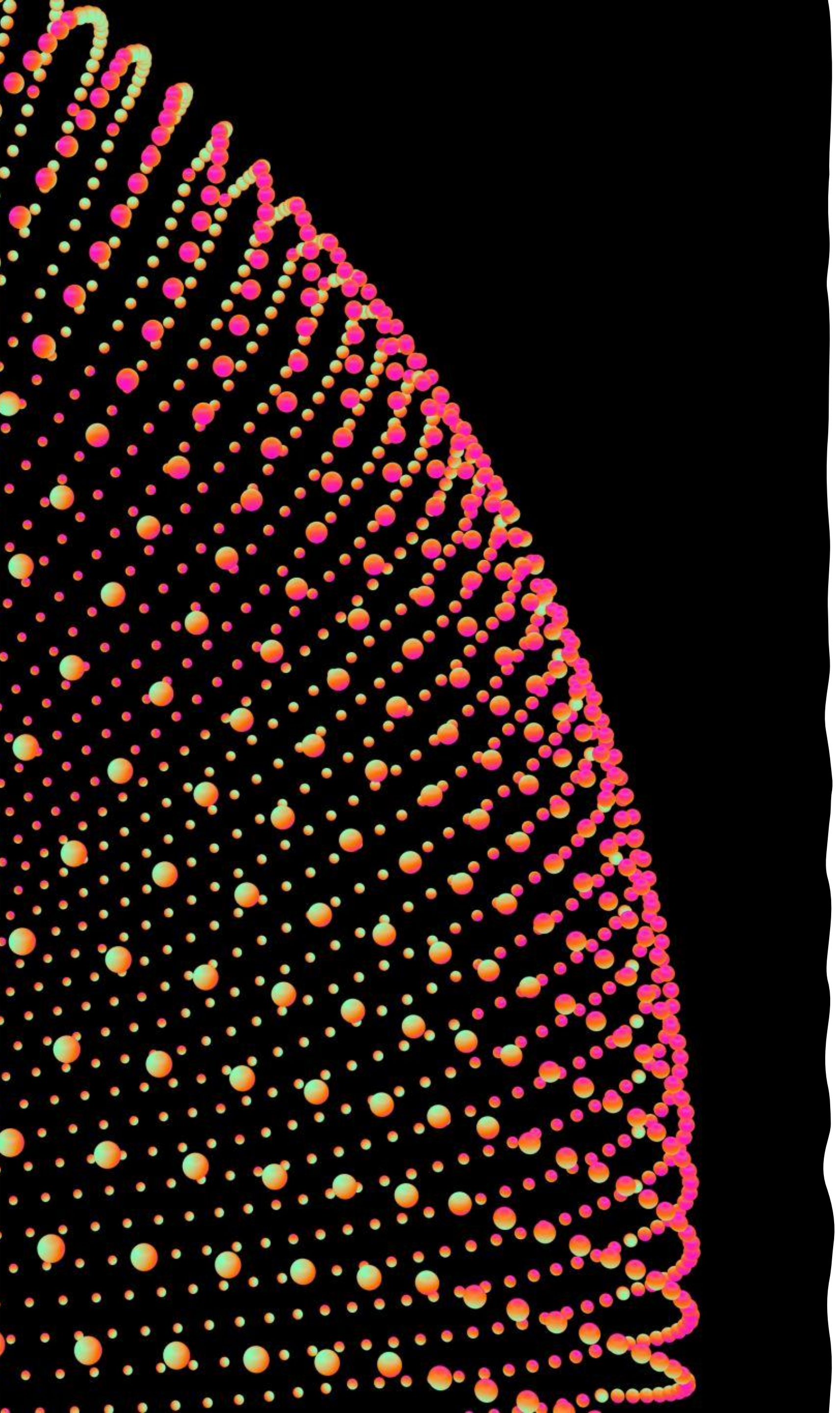


AUTONOMOUS MOBILE ROBOTS

Problem Statement

A 2-Wheeled differential drive robot, equipped with odometer(wheel encoders and/or IMUs) and LIDARs(1 360-lidar or 2 240-lidar), which uses MPC or EKF to avoid dynamically moving obstacles while tracking its base trajectory. The initial setting of the bot is an unknown environment with a set of multiple beacons spread throughout the workspace. All the beacons are connected through a specific trajectory, which the bot has to track. The bot, initially goes through an active-SLAM pipeline to reach out to the nearest beacon. The robot then, starts at a set initial position and has to reach the target position(next beacon) in minimum possible time. The choice of the next beacon would be conveyed through a speech based command.

Such an process would be applicable for an autonomous warehouse management system with multiple robots. Our initial target is implementing the same with a single robot and if time permits we can implement swarm application (hence the first step of our problem involves finding the nearest beacon).

Assumptions made

- The **2-Wheeled Differential drive robot** we are using, has a precise rotational axis about its COM i.e. offsets are ignored. Also the map is assumed to be divided into grids.
- The moving obstacles path is pre-registered into the bot's memory and it precisely knows where the obstacle is at any time point. We have assumed simple trajectories like linear/ circular/ sinusoidal trajectories. At a later stage, we plan to do the same for random obstacle trajectories, which are not known to the bot beforehand.
- We have assumed both our bot and the obstacle to have a **circular** space which makes it easier to model the C-space for our bot.
- The trajectory between any 2 beacons has been developed using **circular paths** making our problem a bit simpler. The **ideal speed input and target beacon** is provided by a speech input to our bot. This speed input would be converted into a "trajectory" data frame within the bot's processor itself.
- We have assumed only a **single obstacle per trajectory** for the sake of simplicity. For multiple obstacles, the process remains similar, but the calculations become lengthy making the algorithms complex.

BASIC OUTLINE

- **Navigation stack** - Initially, the single/multiple robot is placed at the factory's door and it receives the coordinate location from all the beacons. It starts of with implementing active A-star algorithm(incremental learning) (updating the weights for all the beacons at each time step). Parallely it also implements mapping using its LIDAR. This would lead it to the nearest beacon (At a later stage we plan to fuse a particle filtering based localization using the beacon data). Using the local fields, the bot would also create an occupancy grid based map of its environment, which would also help it while tackling the dynamic obstacle.

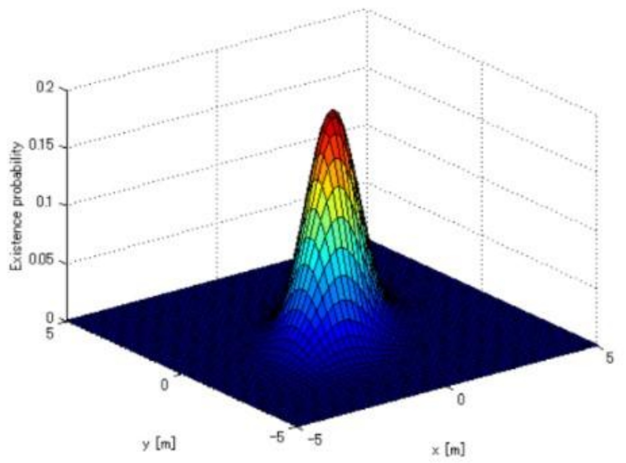
- **Speech Processing stack** - This stack involves the use of a very simple speech2text model that could detect a keyword, process it, refine it and send the required data to all the beacon.

Contd...

- **Trajectory tracking stack -** The circular trajectory can be tracked by the bot by traversing to the next dedicated spot. We would be testing both EKF and MPC for this stack. There exist 2 types of obstacles - along the path obstacle (technically overtaking problem) and through the path obstacle (de-routing/ speed changing problem). Our aim is to tackle both these kind of obstacle but we would start with the overtaking problem as good amount of literature exists for this method.

- **Communication stack** - The beacons are assumed to be targeted workspace in a factory setting. They would be communicating with the bot using a bluetooth/wifi module (which ever is available in Webots). Internally, the "starting and the ending" beacons would also be communicating with each other to keep a track of where the robot is. The speech processing(for commands) unit would also be embedded within the beacons and it would publish these to the bot. In simulation, we would implement this stack by assigning a dedicated data structure for each beacon.The beacons have a specific identifier that would decide which beacon is being reconned.

APF



- In this method, the velocity and position of the obstacle is measured at an instant, using which a probability distribution of the obstacle wrt position is constructed.
- The variance of the probability distribution is measured using which a potential field for the robot is made. This Potential field guides the robot path.
- This Model works really well in cases where the variance is high, i.e. the obstacle motion is very erratic.
- If obstacle motion is nearly uniform, the variance is very low, in which case this model is very inefficient and leads to collisions.
- We won't be using this model in our project on the first priority basis.

Fuzzy Control

- A fuzzy control system is a control system based on fuzzy logic—a mathematical system that analyzes analog input values in terms of logical variables that take on continuous values between 0 and 1
- Fuzzy processing of information leads to a reduction in the control accuracy of the system and deterioration of dynamic quality, with the design of fuzzy control lacking systematicity and unable to define the control goal.

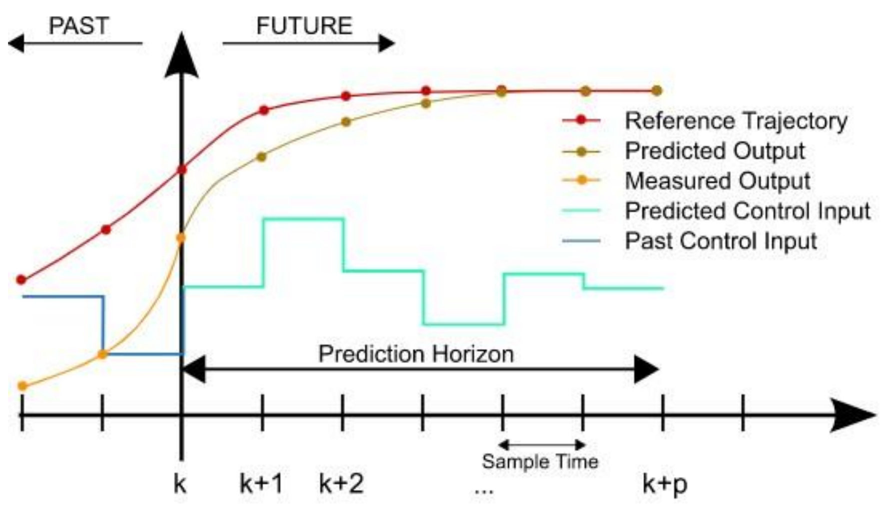
EKF

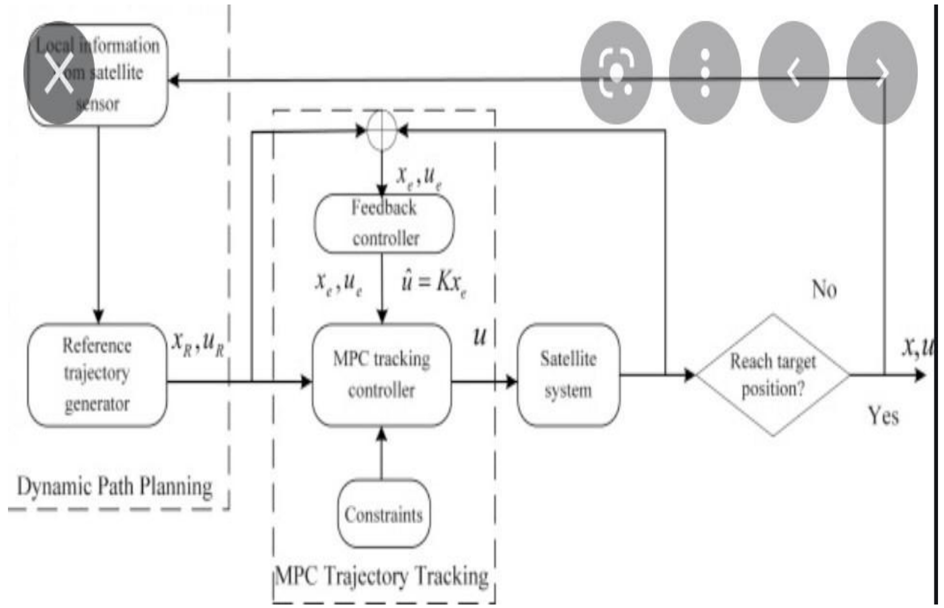
- EKF (Extended Kalman Filters) is used in many robotics applications. Ex: Self-driving cars, drones etc.
- This method is useful when the sensor data can be inaccurate due to noise.
- EKF uses the information about the previous state of the robot and the control input to predict the current state.
- This predicted value of current state is used to predict the sensor values.
- A weighted average of the measured and estimated sensor values is used to get a more accurate representation of the current state.

Model predictive control

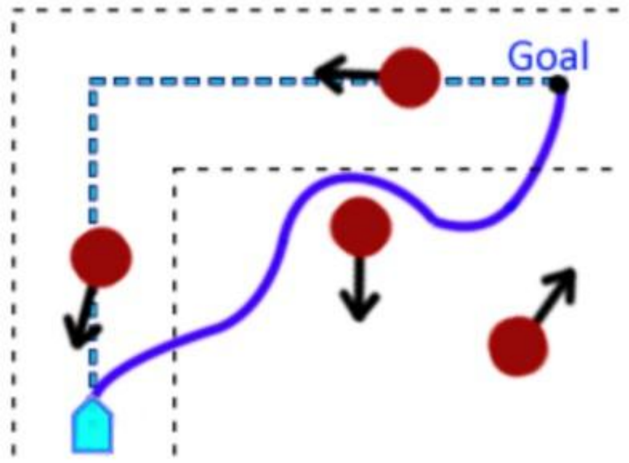
- We will be using this control model to implement trajectory tracking in our project.
- MPC is one of the most powerful control methods with widespread Industrial applications. It is an iterative method which can track and predict the obstacle movement.
- MPC provides a unified optimization based approach which can handle arbitrary constraints on state and control while minimizing a user-defined cost function.
- At a given instant t, the current obstacle velocity and position are known to the bot. These measurements are interpolated for a finite time period, called the predictive period T. For this Predictive period T, the most optimum path for the robot is calculated, based on the interpolated measurements of the obstacle velocity and position. Then, the calculated optimum path is implemented, but, instead of doing it for the entire finite Predictive period, it is done only for the current sampling period, i.e. for a very small period of some milliseconds before another round of measurements could be made. After this small period, the sensors again measure the position and velocity of the obstacle repeating the entire process.

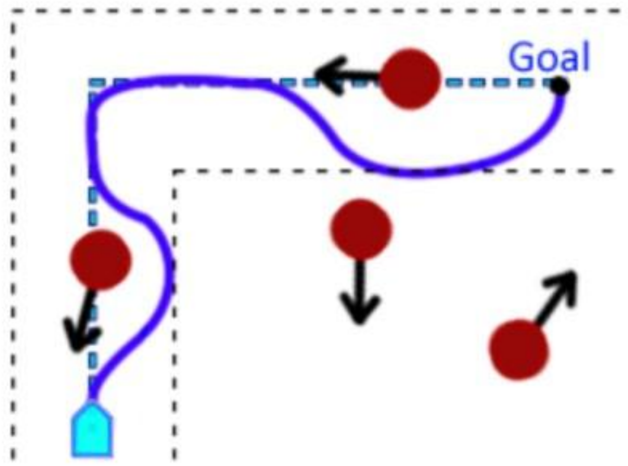
- Before optimizing, we have the option of adding a few constraints. In our case, the constraint will be the prohibition area, which is basically the area occupied by the obstacle at an instant. The bot needs to stay out of this of this prohibition area to avoid collision.
- The constraints can be state constraints as well as control constraints. The prohibition area is a state constraint. Other instances of a state constraint are the Path Boundary condition. On the hand, the if the bot is subjected to velocity constraints, that it can’t exceed Vmax and can’t slow down below Vmin, these are control constraints.
- After optimizing, we get the fastest path the robot can follow, which is then implemented by the robot. This calculation has been made for the predictive period. But, MPC only implements the path for the current sampling period. A few moments after the first sampling measurement, another sampling measurement is made by the sensors. This generates another reference trajectory set. This set is then used for another round of optimization.





- The cost function assumed by us for our MPC model is time taken by the bot to reach the next beacon. This time taken is optimized for the path function using the Euler-Lagrange equations.
- We have chosen Time as the sole optimum route criteria. Whenever the robot encounters an obstacle, it has multiple paths which it can take to avoid the collision, and take the robot to the target point. The path chosen by the robot will be the one which takes it to the target in least possible time. The robot will take the path which will take it to its destination faster (quickest). There are other possible optimum route criteria as well, like the shortest path criteria, where the robot takes the path which is the shortest.
- The Euler Lagrange equations help us in finding a path function by minimizing an associated actional function (cost function), which is the time function in our case, subject to some constraints.





- Let’s assume that at an instant t, the position of the obstacle is given by (x,y), its velocity v and the position of the robot is given by (a,b) and its velocity by V. At this moment, the sensors of the robot take a sample measurement. Based on these measured values, a reference trajectory of the obstacle and the robot is constructed, for a finite time period T, called the Predictive period. This reference trajectory is used by the MPC for calculating the new optimum path of the robot. After time period T, obstacle will be at (x+X,y+Y) while robot will be at (a+A,b+B). If, as per the reference trajectory, there is no collision in the predictive period, the robot will continue to move in the same path which it was moving earlier. If, there is a collision, MPC proposes a new path.
- Assuming circular bot and obstacles make the C-space very simple. The C-space will just have to exclude the the circle occupied by the obstacle at an instant.