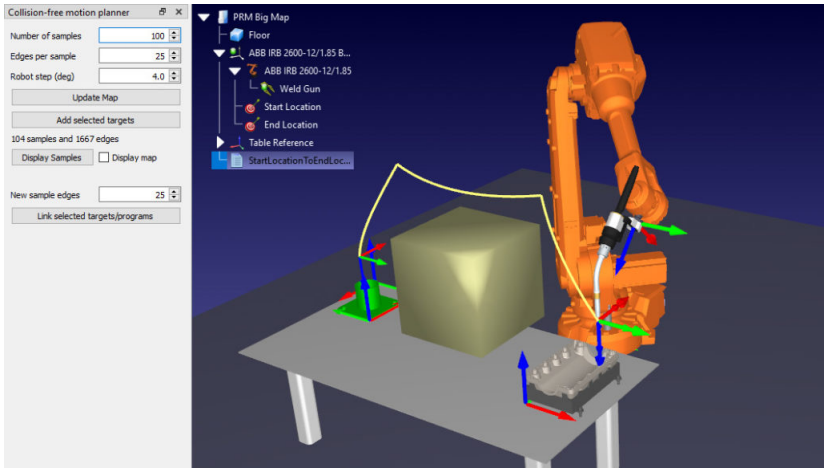
Planning and Navigation

Spring 2021

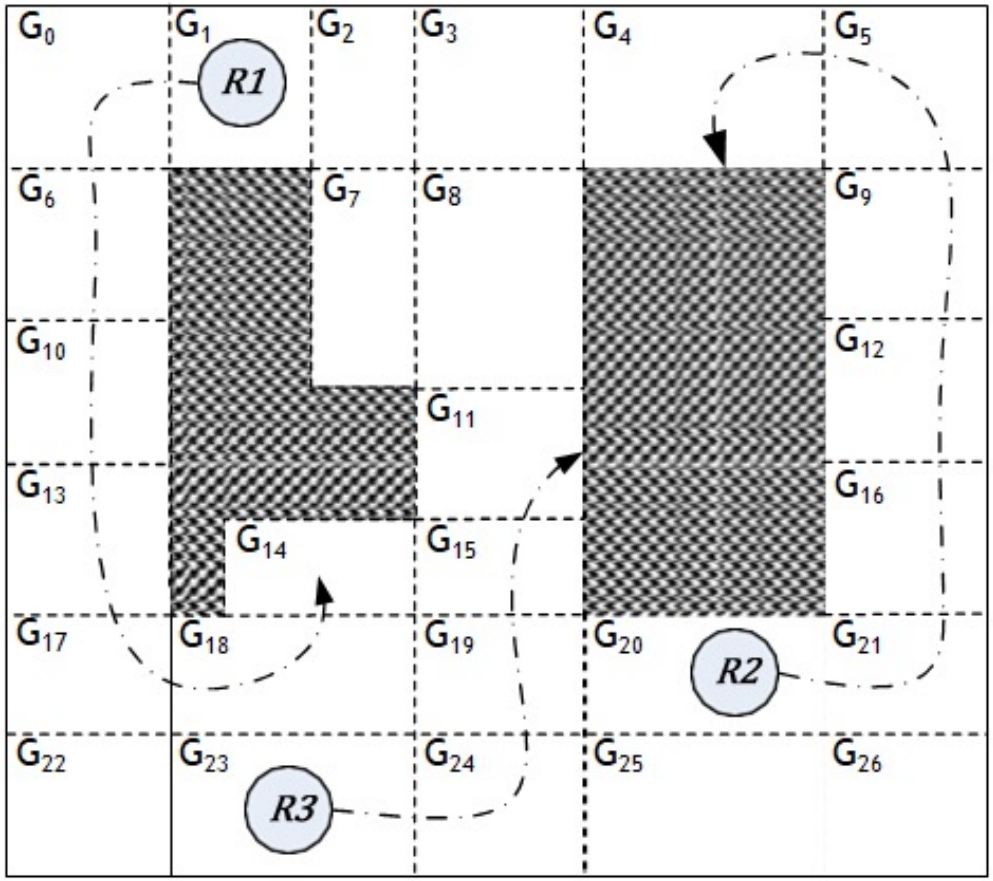
The Idea is Very Commonplace

- https://robodk.com/blog/motion-planning-trend/
  - Why is it such a big trend??

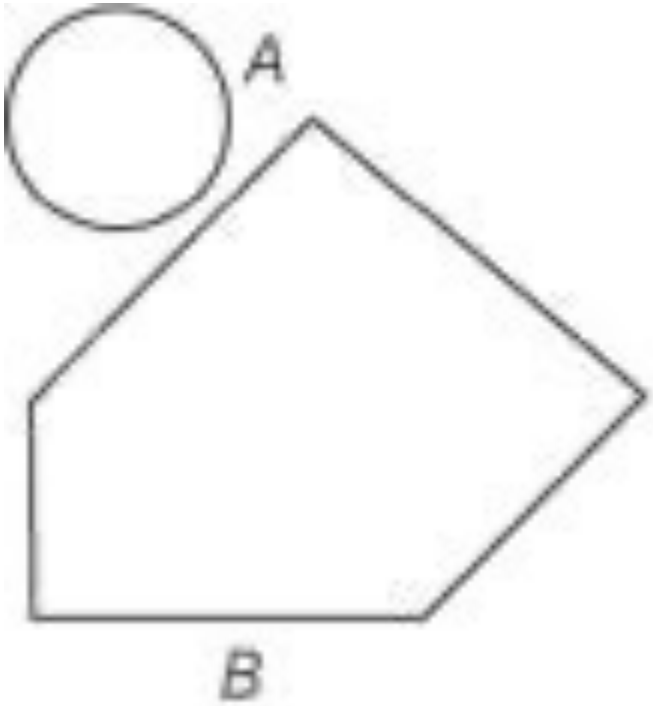


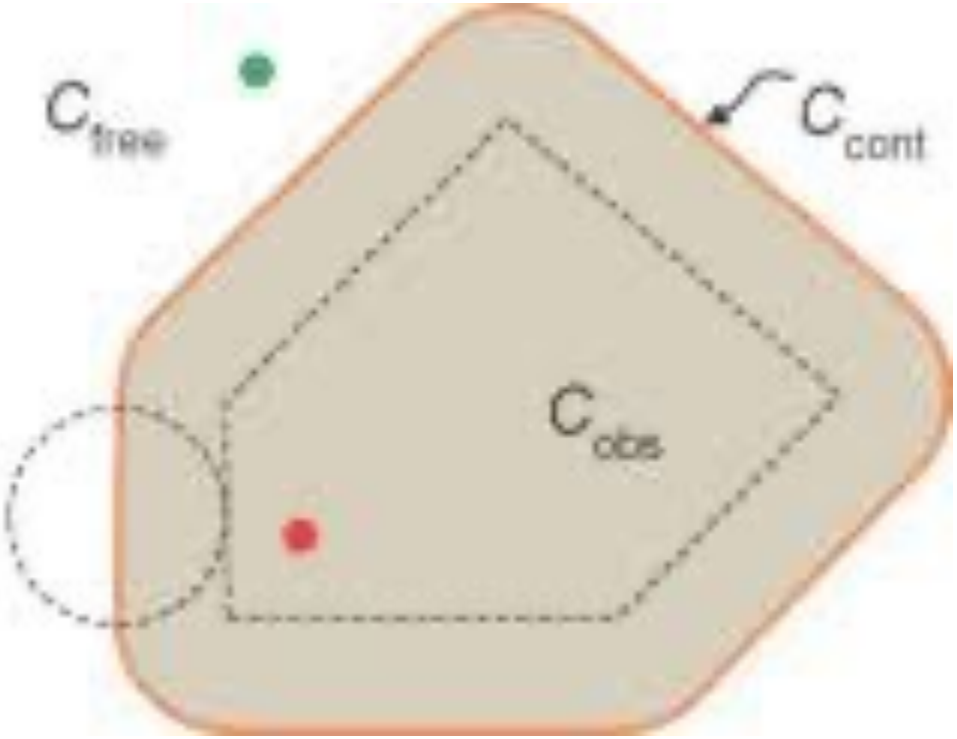
Why is It a Big Trend

Classical AI Methods: Grid Based Planning

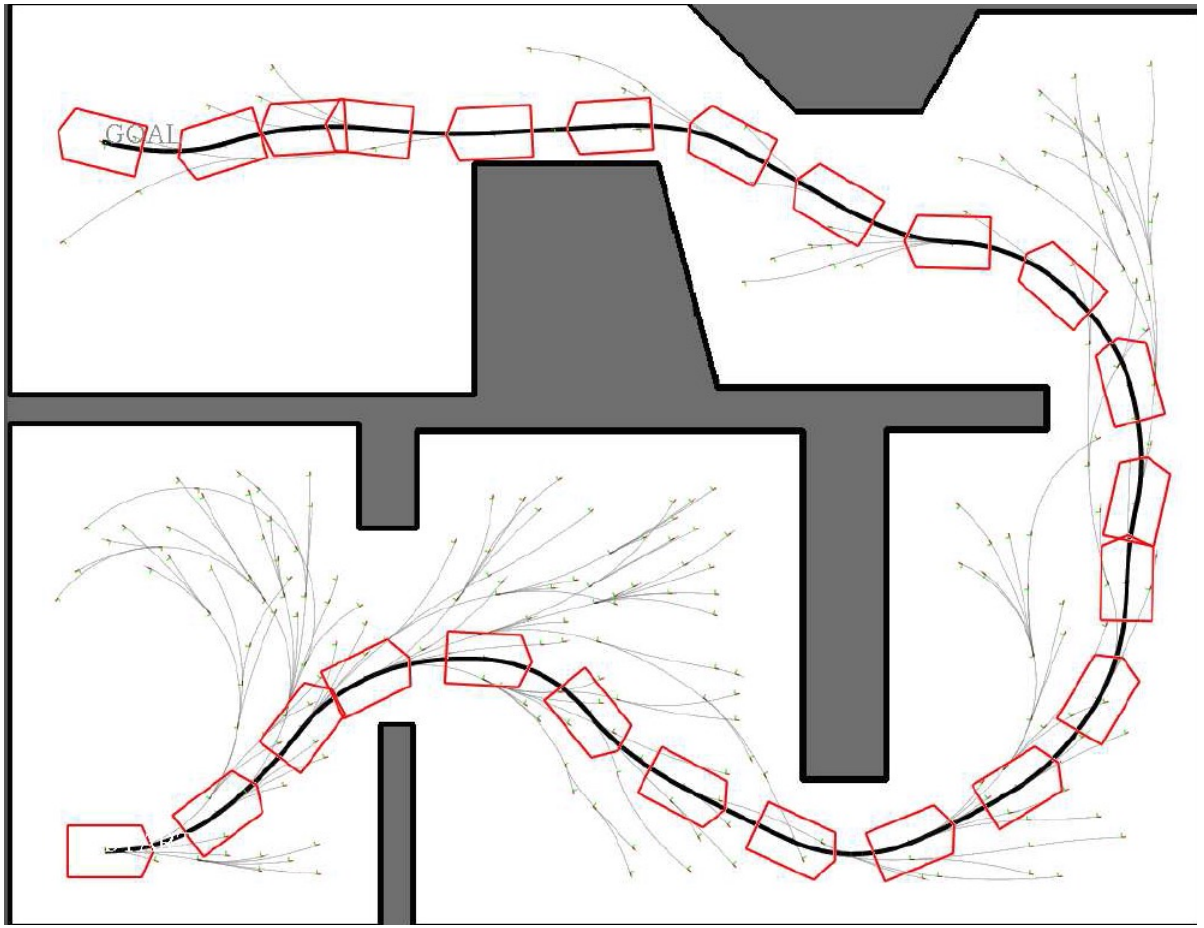


Classical AI Methods:Configuration Space Planning

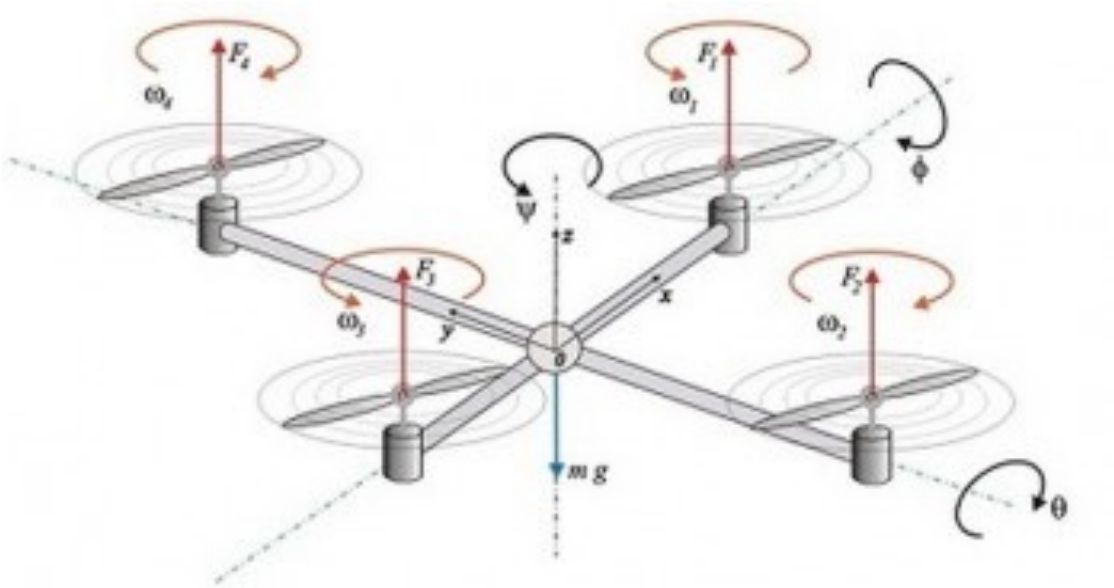




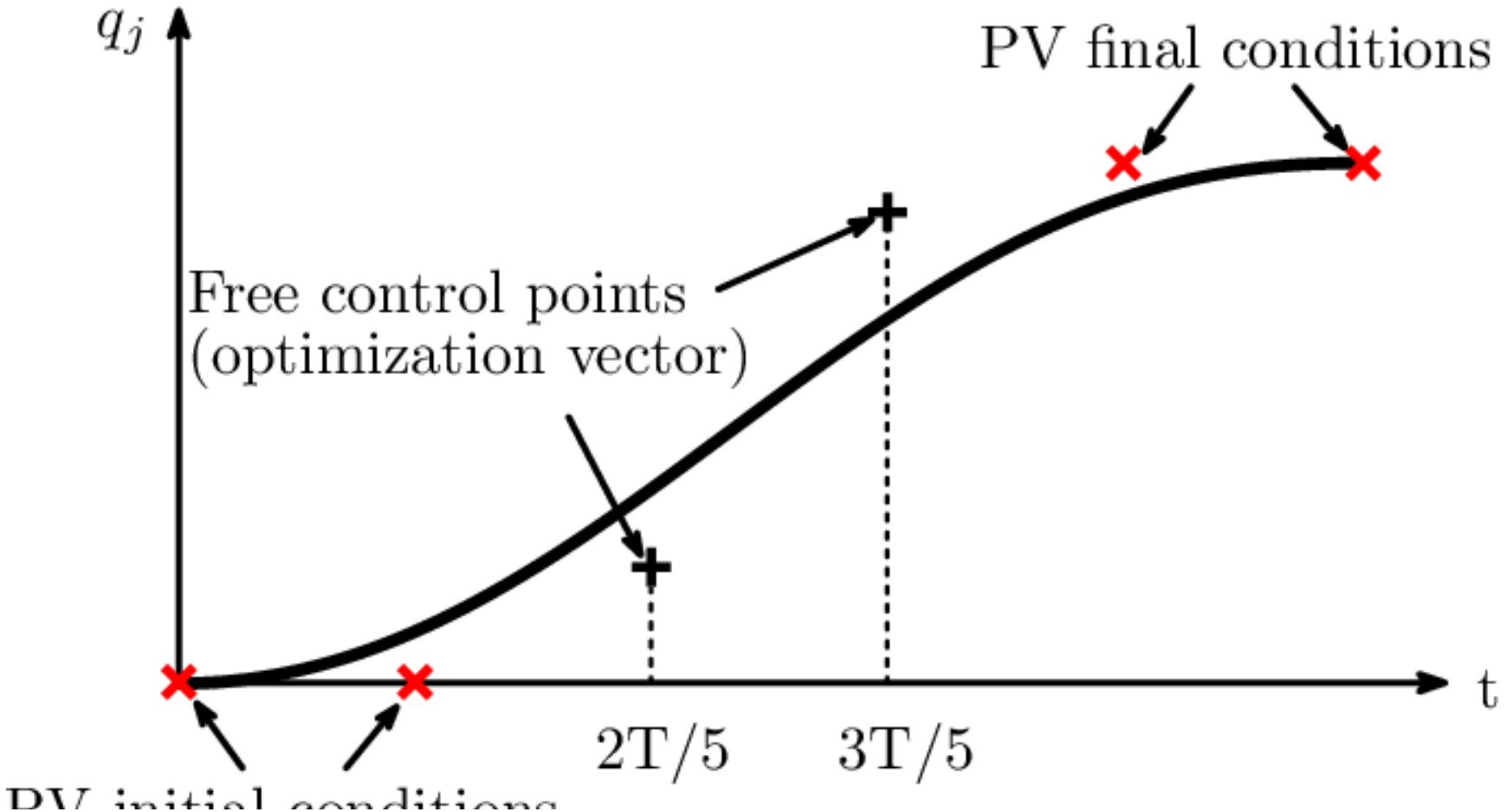
Kinematic Planning



Dynamics Planning: Incorporating Force and Torques



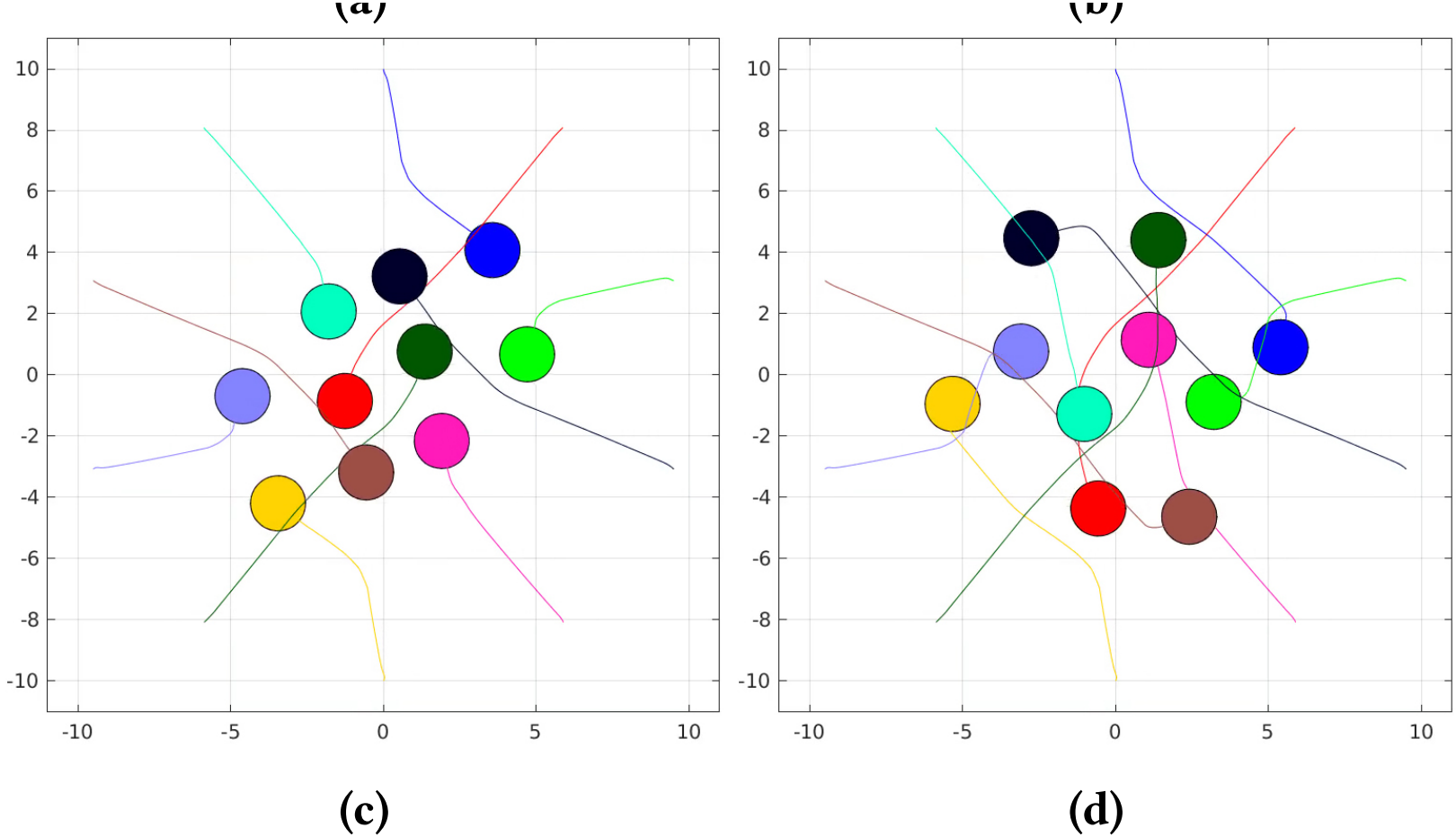
Optimization Methods: Trajectory Parametrization



Optimization Methods: Model Predictive Control

- Play video

Planning in Dynamic Scenes

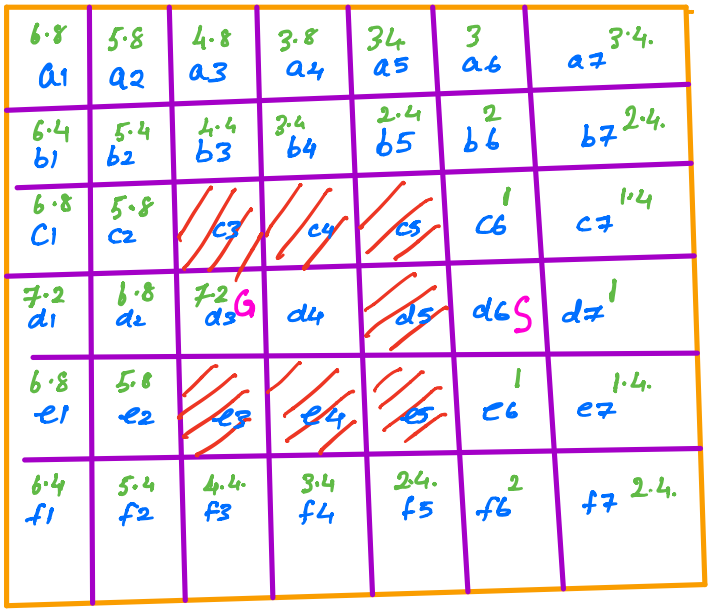


Course Layout

- Quiz – 10%
- Assignments – 40% (2 * 20)
- Project – 20%
- End Sem – 30%

Maybe 4 assignments

Given a start and end location for a point robot in a grid (workspace divided into cells) find a path that connects the start to the goal based on some optimization of a metric.



→ Initialize all cells to very high cost

→ Obstacle cells to ∞ or something higher than the cell costs

→ Make cost of S ← 0 (d6 = 0).

→ Find cost (Nbhrs(S)) → c6 = 1, e6 = d7 = 1. c7 = e7 = 1.4

→ Choose the Nbhr with the least cost from d6. (S). ← c6

→ Expand from c6 or find cost of (Nbhrs(c6)) from S

↳ b6 = 2, b7 = 2.4.

→ Choose the node amongst all expanded / opened nodes, the one with the minimum cost from S, which is e6

→ Expand (e6) → f6, f7, f5

→ Cost(Nbhrs(e6)) → f6 = 2, f7 = 2.4, f5 = 2.4

→ Choose node with the least cost → d7

→ Expand (d7) → No nodes to expand @ d7

→ Continue to expand till the goal node is reached or all nodes in the cell are expanded (Either of the two ways is fine)

→ Find path from G → S by the method of steepest descent.

QUESTIONS:

→ Does the algorithm halt (Find a path if one exists)?

→ Is the path optimal in an eight connected sense from the S?

LIMITATIONS:

1

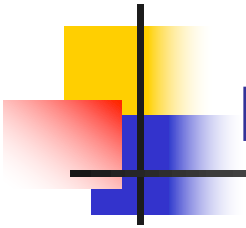
2

3

⋮

Grid-based Navigation for Mobile Robots

Lecture 2



Basic Methodology

- Given an occupancy grid map construct its equivalent cost grid map.
- From the starting position move along the direction of the most negative gradient and reach the next grid along that direction
- Repeat the above step till goal is reached

Constructing the cost-grid

- Each cell in the cost grid denotes the distance from that location to the goal

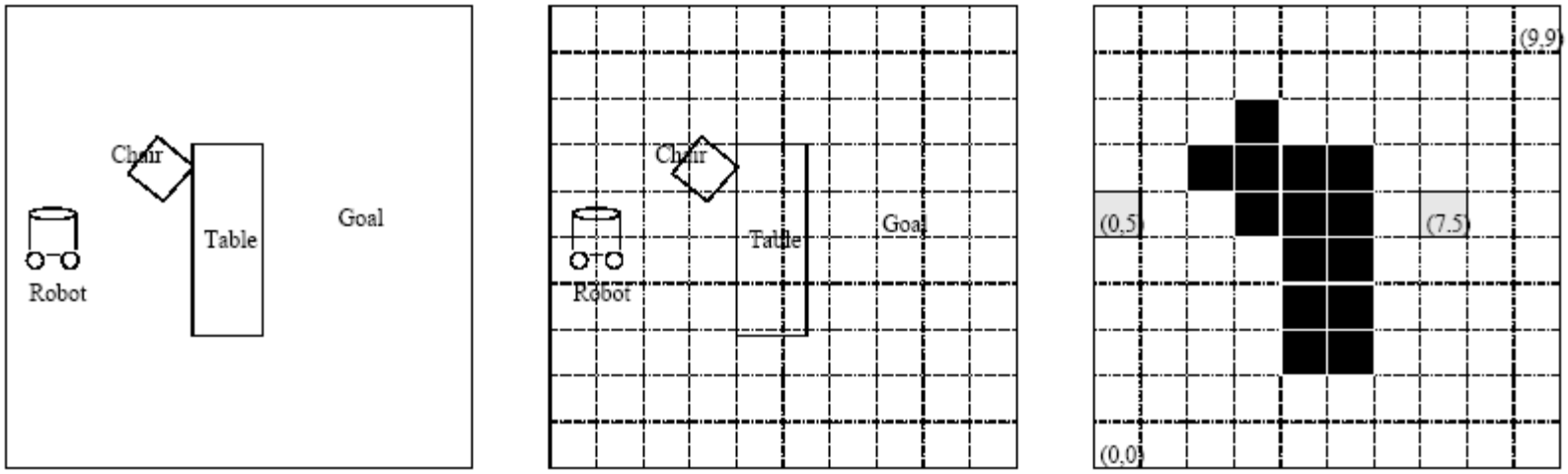


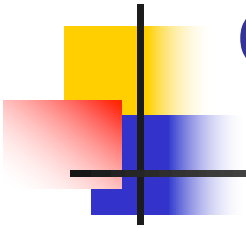
Figure 1: How an example scene (left) is represented in an occupancy grid (right). The black cells are "full" while the white ones are "empty". Locations of the goal and the robot are not usually stored in the occupancy grid, but here they are colored gray for visualization purposes.

Constructing the cost-grid

8.66	7.66	6.66	5.66	5.24	4.83	4.41	4.00	4.41	4.83
8.24	7.24	6.24	5.24	4.24	3.83	3.41	3.00	3.41	3.83
8.66	7.66	6.66	BIG	3.83	2.83	2.41	2.00	2.41	2.83
9.07	8.07	BIG	BIG	BIG	BIG	1.41	1.00	1.41	2.41
9.49	9.07	9.49	BIG	BIG	BIG	1.00	0.00	1.00	2.00
10.49	10.07	9.66	9.24	BIG	BIG	1.41	1.00	1.41	2.41
10.66	9.66	8.66	8.24	BIG	BIG	2.41	2.00	2.41	2.83
10.24	9.24	8.24	7.24	BIG	BIG	3.41	3.00	3.41	3.83
9.83	8.83	7.83	6.83	5.83	4.83	4.41	4.00	4.41	4.83
10.24	9.24	8.24	7.24	6.24	5.83	5.41	5.00	5.41	5.83

Figure 2: The cost grid for the example navigation problem.

Constructing the cost-grid (contd.)

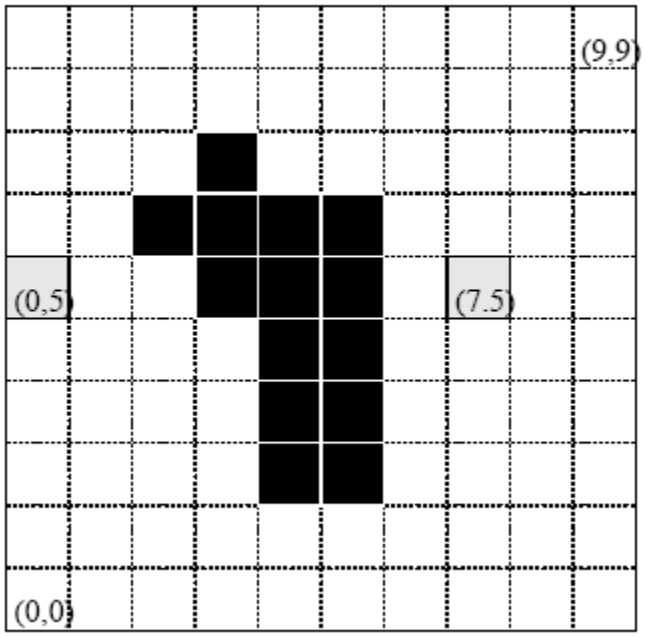


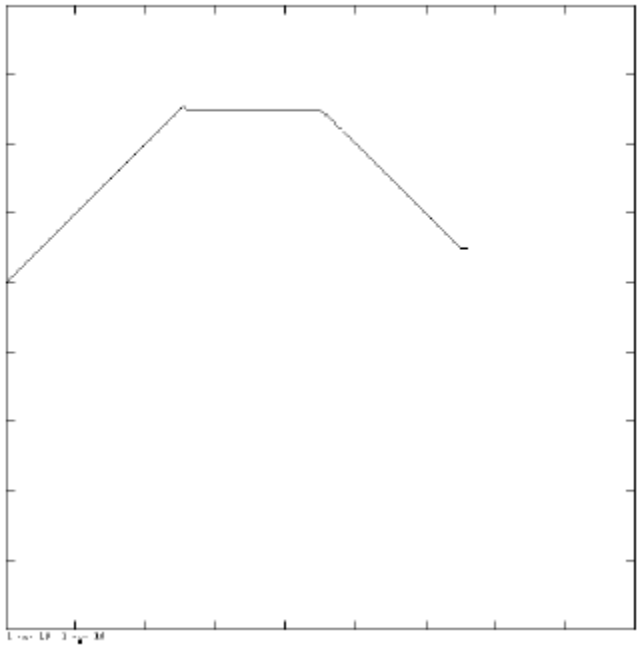
- Initially all cells (occupied and non occupied cells) excepting the goal cell are initialized to BIG values except the goal cell
- A list *"open"* stores the cells whose values have been lowered from their initialized values. The list is sorted with the lowest values at the top of the list that are popped. (Initially the list contains only the goal node)
- The top most node in the list is popped and expanded to its neighbors, whose values are in turn lowered and inserted to the sorted list at their respective places.
- The process continues till all cells in the grid have been expanded once.

Constructing the cost-grid (contd.)

- How the cell costs are lowered: The lower cost is computed by looking at neighboring cells and using an estimate of travel cost from the adjacent cells to the current cell. For laterally adjacent cells the estimate is 1 and for diagonally adjacent cells its 1.414. This estimate is added to the previously computed cost at the adjacent cell to give the cost at the current cell. The minimum of such costs becomes the cost at the current cell.

An Example Path:





Grid-based Navigation for Mobile Robots

Tucker Balch

1 Mobile Robots, Navigate!

Navigation is a special task for mobile robots; after all, isn't getting somewhere what being mobile is all about? And getting somewhere in a complex environment means *navigation*. You may not have given much thought to how you manage to walk across a crowded parking lot, but programming a robot to do the same thing is challenging. Coordinating sensor and motor skills are not the least of the problems, but we'll leave those issues for others to solve for now. This article will examine path planning: how a robot can select a path to a goal.

There's a section on equipment your robot will need to navigate and a step-by-step explanation of how to implement an efficient cost-based algorithm. Tested C source code is included; it's also available by ftp (see the end of the article for ftp information).

2 What A Robot Needs to Navigate

There are a few capabilities a robot must have to navigate. Most importantly it must have some way of sensing where it is supposed to go. This information may be provided by an infrared beacon at the goal, it might be an (x, y) coordinate, or if the robot is equipped with a positional sensor like GPS, a point on the earth's surface defined by latitude and longitude. It is also important for the robot to know where it is in relation to the goal. I'll assume this information is available and that it has been converted into cartesian coordinates. The robot's location is given by `(robot_x, robot_y)` and the goal is `(goal_x, goal_y)`.

You might be wondering how accurate these values need to be. It depends. If your robot directly senses the goal as it moves about, homing on an IR beacon for instance, the values can be somewhat coarse. The accuracy of this type of sensory information does not degrade over time. On the other hand, if the robot depends on internal sensors for position, using timing or shaft encoders for instance, the data must be more precise. As the robot moves about its estimate of position will get worse and worse with this type of sensor. In the end you'll

have to experiment with your particular robot to see if its position sensors are good enough.

Next, the robot must be able to detect obstacles. Sonar range sensors, infrared proximity sensors and laser ranging devices are all excellent, but a good old bump sensor works just fine.

Last is the issue of computing power and memory. The code listings here were implemented and tested on a Unix system with lots of memory. I realize many robots are running around with a 6811 and only 512 bytes of RAM. There's no reason the code shouldn't work on a 6811 (provided you have a compiler), but you'll probably need at least 10K of RAM. The code is in C; but translation to assembly, BASIC, or other languages should be straight forward. Just remember: grid-based path planners are traditionally compact in code, but fat in RAM.

Now on to path planning.

3 Representation for Navigation

Lots of approaches to robot path planning have been proposed and implemented. An important distinction between them is in how they *represent* the world. "Representation" refers to how the data is stored in a computer's memory and how that corresponds to objects in the outside world. In this article, we'll look at one type of grid-based representation. A good reference for information on other approaches is the book *Robot Motion Planning*, by Jean-Claude Latombe. He covers this approach, and others, in great detail.

With respect to navigation, the world consists of open areas, where a robot may travel freely and closed areas (obstacles) where the robot cannot travel. We'll represent these two types of space in a two-dimensional occupancy grid. Each cell in the grid corresponds to a small section of the real world. The occupancy grid is filled in so that a cell is marked "empty" if the corresponding part of the world is free space, and "full" if it contains an obstacle. Figure 1 shows how an example scene is represented in an occupancy grid. Occupancy grids are convenient for a robot to update when sensors indicate changes in the world. They are also easy to use for planning. The primary disadvantage is memory consumption; a high resolution map might require several megabytes.

The occupancy grid is usually read in from a file at start-up time. But such a map may not always be available, or even worse, it might be available but wrong. Luckily, occupancy grids are easy to update if the robot discovers a discrepancy. Suppose, for instance, that the robot discovers a new obstacle just to the north of itself. Since north is in the $+Y$ direction, the occupancy grid is corrected like this:

```
occupancy[robot_x][robot_y + 1] = FULL;
```

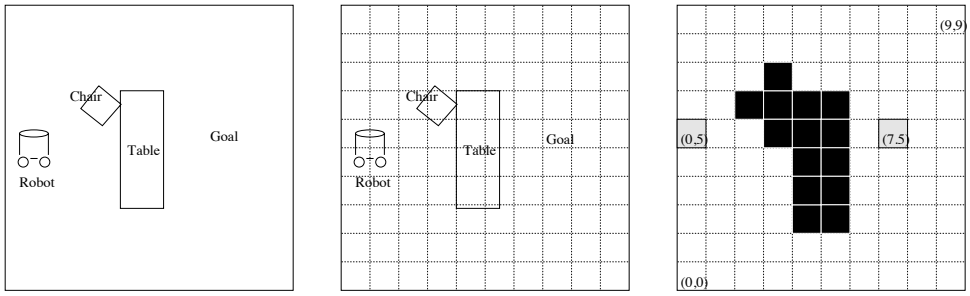


Figure 1: How an example scene (left) is represented in an occupancy grid (right). The black cells are "full" while the white ones are "empty". Locations of the goal and the robot are not usually stored in the occupancy grid, but here they are colored gray for visualization purposes.

Similarly, if space to the east was previously thought to be occupied but turns out to be free, the correction is made by:

```
occupancy[robot_x + 1][robot_y] = EMPTY;
```

Note that changes in the occupancy grid will require a replanning step since a new obstacle might obstruct the planned route, or newly discovered open space might offer a short cut.

One final issue regarding the occupancy grid is *resolution*. The grid's resolution refers to how large an area in the real world is represented by one cell in the grid. The example uses a 1 foot resolution on 10 by 10 foot grid (Figure 1). For most robot applications this resolution is too low. Low resolution may lead to a less optimal path and jerky robot motion. The selection of a resolution should depend on the accuracy of the robot's position sensors, how fast it will move and how much memory is available. In general, it is best to use as high a resolution as possible since this will result in the most accurate representation of obstacles and the "smoothest" plan. But there are reasons to avoid too high a resolution. It doesn't make sense, for instance, to use a higher resolution than the precision of the robot's position sensor. Also, if the robot moves so quickly that it skips over several cells between computation cycles, the resolution is probably too high. Finally, higher resolution maps will take longer to plan over and use more space. A good starting point is to set the resolution to the distance your robot will travel in one computation cycle.

4 Representation of the Plan

The plan is a cost grid. Each cell in the grid is an estimate of the shortest travel distance from that point to the goal. Usually the cost grid is the same

8.66	7.66	6.66	5.66	5.24	4.83	4.41	4.00	4.41	4.83
8.24	7.24	6.24	5.24	4.24	3.83	3.41	3.00	3.41	3.83
8.66	7.66	6.66	BIG	3.83	2.83	2.41	2.00	2.41	2.83
9.07	8.07	BIG	BIG	BIG	BIG	1.41	1.00	1.41	2.41
9.49	9.07	9.49	BIG	BIG	BIG	1.00	0.00	1.00	2.00
10.49	10.07	9.66	9.24	BIG	BIG	1.41	1.00	1.41	2.41
10.66	9.66	8.66	8.24	BIG	BIG	2.41	2.00	2.41	2.83
10.24	9.24	8.24	7.24	BIG	BIG	3.41	3.00	3.41	3.83
9.83	8.83	7.83	6.83	5.83	4.83	4.41	4.00	4.41	4.83
10.24	9.24	8.24	7.24	6.24	5.83	5.41	5.00	5.41	5.83

Figure 2: The cost grid for the example navigation problem.

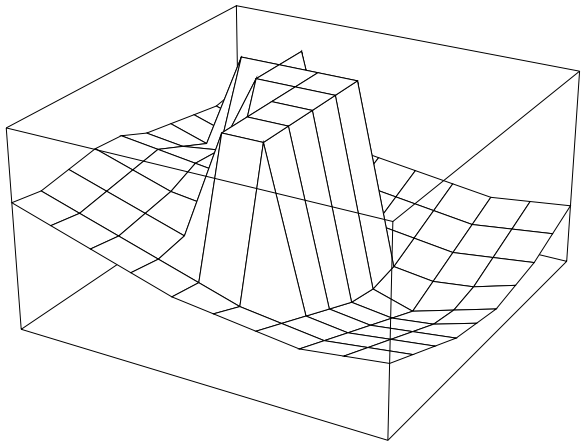


Figure 3: The cost grid viewed as a three dimensional surface.

resolution as the occupancy grid. This makes referencing one while using the other more convenient.

For the moment don't worry about how cost cells are filled in. You'll see how to do that in the next section. Look at Figure 2. This is a cost grid for the example in Figure 1. Note that the cost at the goal cell is 0.0, and the cost at other cells increases the further they are from the goal. To get a better idea of what the cost grid looks like we can view a three dimensional surface generated by plotting the cost at each point as a height. Figure 3 shows the plot for this cost grid. The high spots on the plot are obstacles. The goal is at the low point at the front right. You can see that following the plan is just like rolling a ball down hill.

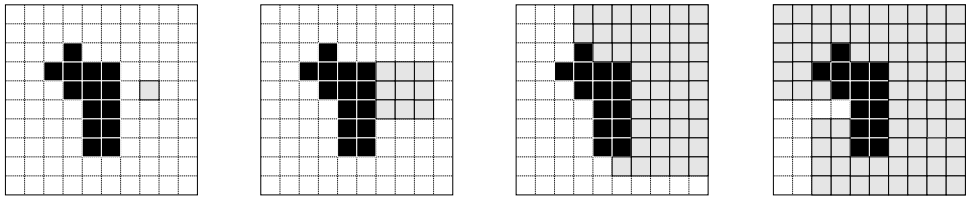


Figure 4: This sequence shows how cost computation expands outwardly from the goal. Initially (left), only the cost at the goal is known. The other images, from left to right, show the cells that have been evaluated after 1, 4 and 7 iterations.

5 How the Cost Grid is Computed

As you may realize by now, the hard part is computing the cost grid. Recall that the number in each cell is an estimate of the cost of traveling from that cell to the goal.

First the program sets all the cells in the grid to initial values. Goal and obstacle cells have constant values throughout the computation. Since the cost of traveling from the goal to the goal is 0 the goal cell is always set to 0.0. Since traveling through obstacles is not normally desired (!) we discourage this by setting the cost at obstacle cells to an arbitrarily large value called `BIG_COST`. "Empty" cells are also initially set to `BIG_COST`, but lower values for them will be computed later.

After setting the cells to initial values, the program repeatedly scans through the grid looking for cells it can reset to lower values. The lower cost is computed by looking at neighboring cells and using an estimate of travel cost from the adjacent cells to the current cell. For laterally adjacent cells the estimate is 1.0, and for diagonally adjacent cells it's $\sqrt{2}$. This is just the distance from the center of one cell to the center of the next, assuming each cell measures one unit on a side. Note that if the cost estimates were multiplied by the resolution of the grid the values at each cell would reflect the true distance to the goal. At each sweep through the grid one more layer of cells is re-evaluated until, eventually, the entire grid is minimized. Figure 4 shows how the evaluation spreads outwardly from the goal.

At heart of the planner is the procedure `cell_cost()` (listing at end of article). `cell_cost()` evaluates the cost at one cell by inspecting the cost of each cell adjacent to it. For the neighboring cells, it adds appropriate lateral or diagonal travel costs and notes the lowest value. That lowest value is recorded in the current cell. `cell_cost()` returns a 0 if there was no change in the cost, 1 otherwise.

We now have all the pieces needed to build a cost-based grid path planner. Here it is:

```
int count = 1;
while (count != 0)
        {
        count = 0;
        for (i = 1; i < GRID_SIZE-1; i++)
                {
                for (j = 1; j < GRID_SIZE-1; j++)
                        {
                        count = count + cell_cost(i,j);
                        }
                }
        }
```

Yes, that *really* is the entire planner! It repeatedly cycles through the grid to recompute each cell's cost until it makes a full pass with no changes. Now for the bad news: as it stands, this planner is *extremely* inefficient. If the grid has N cells along each side, and we plan over a complicated map, it might cycle through the grid N^2 times and make N^4 cell evaluations. This will use up a lot of CPU cycles for a large grid. In a later section we'll see how to make it faster.

6 Using the Cost Grid as a Plan

Let's look at how to employ the cost grid as a plan. Provided the robot knows its own location, using the plan is as simple as

```
new_direction = check_plan(robot_x, robot_y);
```

The function `check_plan()` knows how to consult the cost grid and return a heading for the robot. It looks at the region in the cost grid corresponding to where the robot is in the world. Then it choses the direction along the shortest path to the goal (i.e. "down hill" on the cost grid).

To do this, `check_plan()` looks at a sample of nearby cells to compute the *gradient* at the position of the robot. The gradient is down hill on the cost grid. X and Y components of the gradient are computed separately, then the `atan2()` function is used to convert them into a direction between 0 and 2π. If one of the nearby cells contains an obstacle, `check_plan()` uses the direction towards the lowest cost neighbor instead of using the gradient. This avoids jittering when the robot is near obstacles.

7 Pulling it All Together

A robot using a grid-based planner should follow a cycle of planning and acting something like this:

1. Initialize variables and read the map.
2. Plan.
3. Check sensors to find robot position, goal and nearby obstacles.
4. Update map if sensors show a discrepancy.
5. If the map has changed, recompute the plan.
6. Check plan and initiate movement along shortest path.
7. Go to 3.

This sequence is used in the C code below. But this program is just a simulation until you replace the sensing and movement "stub" procedures with appropriate subroutines for your hardware.

The program includes a procedure called `readmap()` that will read a map of obstacles into the occupancy grid and initialize the robot and goal locations. If you want to use this capability, you can make a map of the environment in a text file using spaces for open space, the letter 'O' for obstacles, the letter 'G' for the goal, and 'R' for the robot. The example from Figure 1 can be coded in a text file as:

```
␣␣␣␣␣␣␣␣␣␣<CR>
␣␣␣␣␣␣␣␣␣␣<CR>
␣␣␣␣O␣␣␣␣␣␣<CR>
␣␣OOOO␣␣␣␣<CR>
R␣␣␣OOO␣G␣␣<CR>
␣␣␣␣OO␣␣␣␣<CR>
␣␣␣␣OO␣␣␣␣<CR>
␣␣␣␣OO␣␣␣␣<CR>
␣␣␣␣␣␣␣␣␣␣<CR>
␣␣␣␣␣␣␣␣␣␣<CR>
```

The ␣ symbol indicates a space, and $< CR >$ indicates the end of a line. If you run the program using this input file, it will simulate moving the robot from its initial location $(1, 6)$ to the goal, $(8, 6)$. The resulting path is shown in Figure 5. If the computer on your robot cannot read files, you'll obviously have to avoid using `readmap()`.

Also, if you'd like, you can print out the cost grid using the function `printcost()`

8 Making it Faster

Earlier, I pointed out that the present routine for computing the cost grid is inefficient. For the example problem 1000 cell evaluations are made. This means

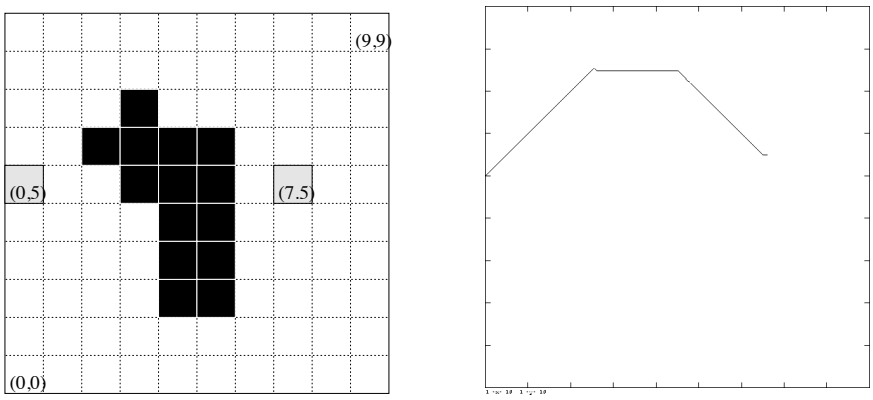


Figure 5: A simulated robot navigates across the scene on the left. The resulting path is shown on the right.

each cell in the 10 by 10 grid is evaluated 10 times. It should really only be necessary to evaluate each cell once. But if efficiency is not important, you might want to stick with the slower version since it is less complicated.

The main problem with the old planner is that cells are not evaluated in an efficient order. Let's look a better way to order the evaluations. Consider the first time a cell is evaluated. Recall that all the cells are initially set to `BIG_COST`. If all the cells adjacent to the current cell contain `BIG_COST` as well, there is no way the cell can be lowered. A cell should not be evaluated until after one of its neighbors has been lowered, otherwise we're wasting time. Initially, the only cells with a low cost neighbor are the cells next to the goal, so they should be evaluated first. Also, if we can be sure the first evaluation is correct, there is no need to evaluate a cell again.

One way to do this is to arrange for cells to trigger the evaluation of neighboring cells after their cost values have been lowered. To do this we keep a list, called the *open list*, of cells whose cost has been lowered. It is automatically sorted from lowest to highest using linked list routines. Cells are "popped" off the top of the list by a routine called `expand()`. After `expand()` pops a cell off the list, it looks at each of the cell's neighbors to determine if they have been evaluated. Each cell that has not been evaluated is evaluated at that point, then pushed onto the open list for later expansion. This ensures that the lowest cost cells are expanded first.

The new planner works by first pushing the goal onto the open list. Next it repeatedly pops cells off the list and expands them until the open list is empty. The computation eventually terminates since each cell is expanded only once and cells outside the boundaries are not pushed onto the list.

This planner makes only 85 cell evaluations on the example problem, so it runs about 10 times as fast as the old planner for that case. In most situations

the faster planner will run N times faster where N is the number of cells along one side of the grid. Due to space considerations, listings for fast version are not not included below, but they are available by e-mail or ftp.

9 Wrapping Up

There are ways to make an even faster grid-based planner. The A* (pronounced "A star" with a long "a") algorithm works by only expanding nodes along a direct path between the goal and the robot. D* (pronounced "dee star"), developed by Anthony Stentz at CMU, initially computes the grid as outlined here. But D* keeps additional information so that when errors are found in the map the plan can be corrected without recomputing the entire grid. For dynamic or unknown situations D* is better since it does not require a complete replanning step when errors are found.

If you'd rather not type in the program by hand, you can find it via anonymous ftp at `ftp.cc.gatech.edu` in the directory `people/tucker/gridnav1.0` Another directory, `gridnav2.0` contains the fast planner. Get all the files from one directory or the other. If you are familiar with `tar` and `uncompress` you can grab `gridnav.tar.Z` to get them all at once. If you don't have access to ftp, send me e-mail indicating which version you want and I'll e-mail the files back to you (`tucker@cc.gatech.edu`). After you have the files on your local system, a simple `make gridnav` should compile it for you.

10 About the Author

Tucker Balch was born in Miami, Florida in 1962. He received the B.S. Degree from Georgia Tech in 1984 and the M.S. Degree from U.C. Davis in 1988. He is currently pursuing a Ph.D. in Autonomous Robotics at Georgia Tech. From 1984 to 1988 he supported research at the Lawrence Livermore National Laboratory as a computer scientist. He entered the Air Force as a Pilot Candidate in 1988 and completed fighter training in 1991. He now flies F-15 Eagles at the Georgia Air National Guard.

His research interests include integration of deliberative planning and reactive control, communication in multi-robot societies, and parallel algorithms for robot navigation.

Tucker can be reached by e-mail at `tucker@cc.gatech.edu` . His world-wide-web page is `http://www.cc.gatech.edu:/grads/b/Tucker.Balch` .

References

[1] Latombe, Jean-Claude, *Robot Motion Planning*, Kluwer Academic Publishers, Boston, 1991.

[2] Stentz, A., "Optimal and efficient path planning for partially-known environments", *Proceedings 1994 IEEE International Conference on Robotics and Automation*, p.3310-17 vol.4, 1994.

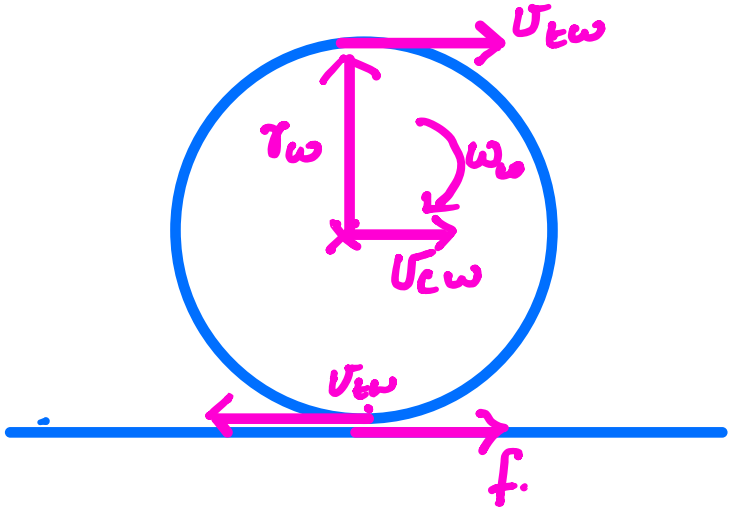
11 Source Code Listings

Forward Kinematics:

If the motors of a robot rotate with commanded angular velocities where would the robot reach in a given time interval.

Differential Drive:

→ Rolling NO Slipping

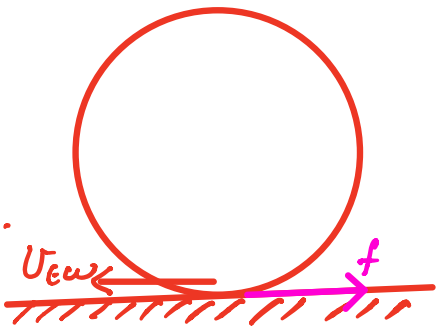


Consider a wheel that rotates freely about the center in air. When this wheel is placed on the ground, aided by the force of friction, f, begins to move forward.

In other words in the absence of such a force the wheel would rotate, but its velocities at

the top and bottom are in opposite direction and of equal magnitude that it would not move. $\vec{v}_{tw}$ direction at the top of the wheel is → and at the bottom (point of contact with the ground) is ←, cancelling each other and preventing any translation

However at the point of contact with the ground the tendency of the wheel is to move in ← direction.



f opposes this tendency by acting in → direction

This opposing nature of f causes the wheel to move in → direction.

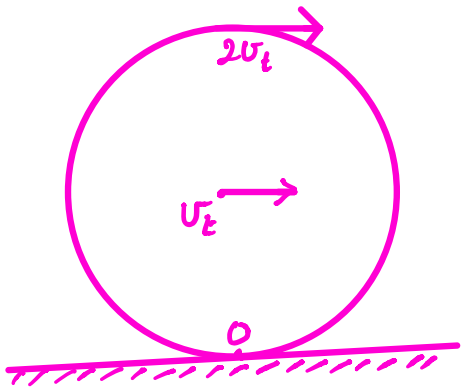
Let $\vec{v}_{cw}$ be the translational velocity of the wheel center. Then velocity at the top of the wheel is $\vec{v}_{cw} + \vec{v}_{tw}$. Note $\vec{v}_{cw}$ is the effect of f.

The velocity at the point of contact with the ground is $\vec{v}_{cw} - \vec{v}_{tw}$

Rolling without slipping or pure rolling entails: $v_{cw} - v_{tw} = 0$

Or $v_{cw} = v_{tw} = r_w \omega_w$. or $r\omega$ dropping the suffix w denoting wheel.

Hence

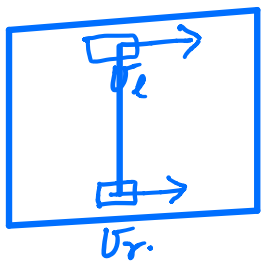


at the point of contact with the ground the wheel is instantaneously at rest.

NOTE: The translational velocity of the wheel is v_{cw} is the same at all points on the wheel. However the velocity due to the rotation of the wheel center v_t is different at different location of the wheel.

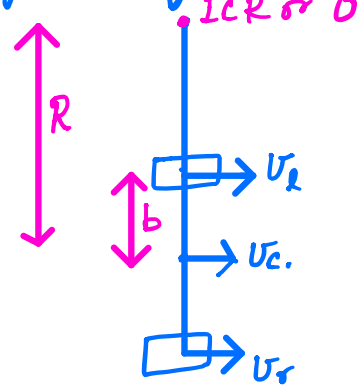
DIFFERENTIAL DRIVE ROBOT:

Differential drive kinematics: left and right wheel independently controlled.



Let the linear velocity of the left wheel center (translational velocity) be v_l and the right wheel center be v_r.

The entire robot rotates about an instantaneous center of rotation ICR that lies on the line joining the left and right wheel (the wheel base)



Let ω be the angular velocity with which it rotates about O. Then

$\vec{v_c} = \vec{\omega} \times \vec{r}$ or $|v_c| = R\omega$. $\longrightarrow (1)$

$v_l = (R-b)\omega \longrightarrow (2)$

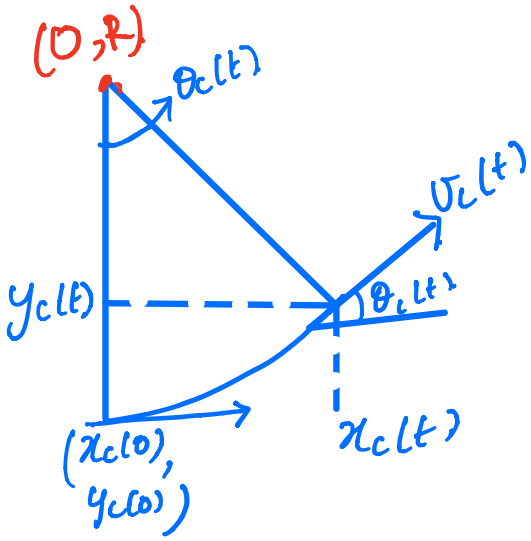
$v_r = (R+b)\omega \longrightarrow (3)$

$\begin{bmatrix} v \\ \omega \end{bmatrix} = f(v_l, v_r)$

$\downarrow$

$v_p = f(v_w)$

$v_c = \frac{v_l + v_r}{2} \longrightarrow (4)$. $\omega = \frac{v_r - v_l}{2b}$. $\longrightarrow (5)$



$x_c(t) = \int_0^t v_c(t) \cos(\theta_c(t)).dt.$

$= \int_0^t v_c \cos(\theta_c)\, dt \longrightarrow (6).$

$y_c(t) = \int_0^t v_c \sin(\theta_c)\, dt \longrightarrow (7).$

$\theta_c(t) = \int_0^t \omega\, dt \longrightarrow (8).$

$\theta_c(t) = \omega t \longrightarrow (9).$

Then $x_c(t) = \int_0^t v_c \cos(\omega t)\, dt = \frac{v_c \sin(\omega t)}{\omega}\Big|_0^t = \frac{v_c}{\omega}\sin(\omega t)$

$\longrightarrow (9).$

$y_c(t) = \int_0^t v_c \sin(\omega t)\, dt = -\frac{v_c}{\omega}\cos(\omega t)\Big|_0^t = -\frac{v_c}{\omega}[\cos(\omega t) - 1]$

$\longrightarrow (10)$

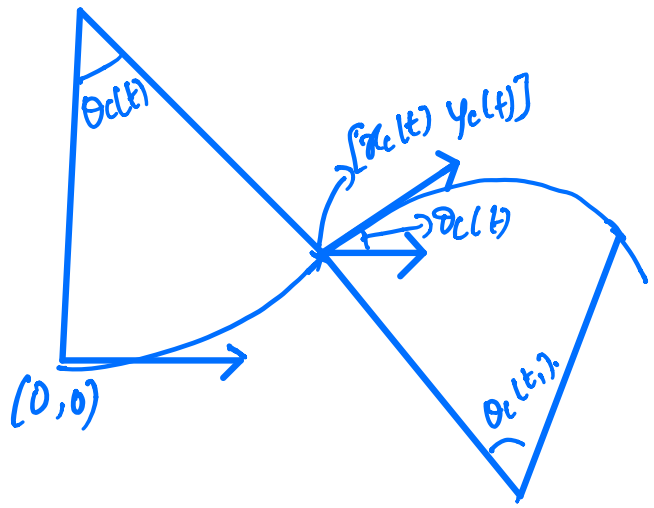
From (9) & (10) we get

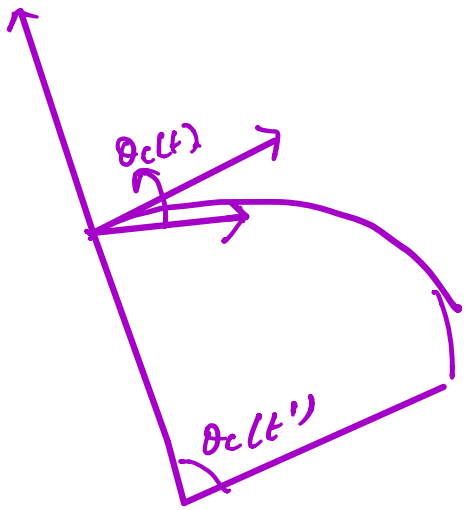
$$x_c^2 + \left(y_c(t) - \frac{v_c}{\omega}\right)^2 = \frac{v_c^2}{\omega^2}\left(\cos^2\omega t + \sin^2\omega t\right) = R^2$$

or $\underline{x_c^2(t) + (y_c(t) - R)^2 = R^2}$

How do you aggregate multiple such segment over time?

Use coordinate transform.





Let $t=0$ translate the spatial and temporal origin to $(x_c(t), y_c(t), t)$.

$t' = t_1 - t$.

Then $x_c(t') = \frac{v_c'}{\omega'}\sin(\omega' t') \longrightarrow (11)$.

$y_c(t') = -\frac{v_c'}{\omega'}\left[1 - \cos(\omega' t')\right] \longrightarrow (12)$.

Where (v_c', ω') is the linear and angular velocity from $t \to t'$.

Where is the robot now with respect to the original origin?

$$\begin{bmatrix} x_c(t_1) \\ y_c(t_1) \end{bmatrix} = \begin{bmatrix} \cos\theta_c & -\sin\theta_c \\ \sin\theta_c & \cos\theta_c \end{bmatrix} \begin{bmatrix} x_c(t') \\ y_c(t') \end{bmatrix} + \begin{bmatrix} x_c(t) \\ y_c(t) \end{bmatrix}$$

$\longrightarrow$ (13).

$$\theta_c(t_1) = \theta_c(t) + \theta_c(t') \longrightarrow (14).$$

Kinematics in the presence of accelerations:

$$x_c(t) = x_c(0) + \int_0^t (V_c + at)\cos\left(\theta_c(0) + \omega t + \frac{\alpha t^2}{2}\right) dt.$$

$\longrightarrow$ (15).

$$y_c(t) = y_c(0) + \int_0^t (V_c + at)\sin\left(\theta_c(0) + \omega t + \frac{\alpha t^2}{2}\right) dt$$

$\longrightarrow$ (16).

The above integrals need to be numerically computed by the method of Fresnal Integrals.

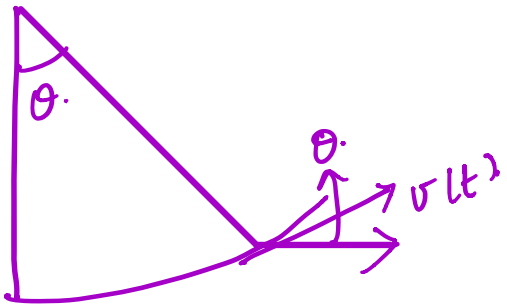
The resulting curve obtained from numerical integration is called a CLOTHOID.

Non Holonomic Robot:

The Differential Drive robot is Non Holonomic.

$$\left.\begin{aligned} \dot{x}_c(t) &= v\cos(\omega t). \\ \dot{y}_c(t) &= v\sin(\omega t). \end{aligned}\right\} \boxed{\dot{y}_c(t) = \dot{x}_c(t)\tan\theta.}$$

v_y and v_x are not **Decoupled**. They are coupled through the robots instantaneous direction / heading θ.



The robot's tangential velocity $v(t)$ is always along the heading direction $\theta(t)$

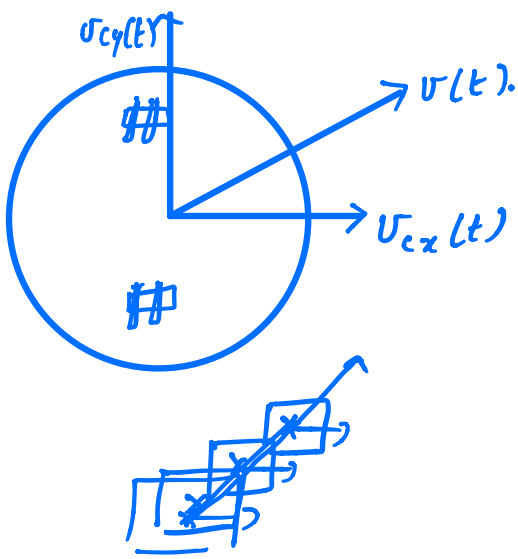
NO INDEPENDENT CONTROL of V_x and V_y.

Control degrees of freedom (v, ω).

Configuration / cartesian degrees of freedom (x, y, θ).

The degrees of control is less than the degrees of configuration accessible by the robot.

HOLONOMIC KINEMATICS:

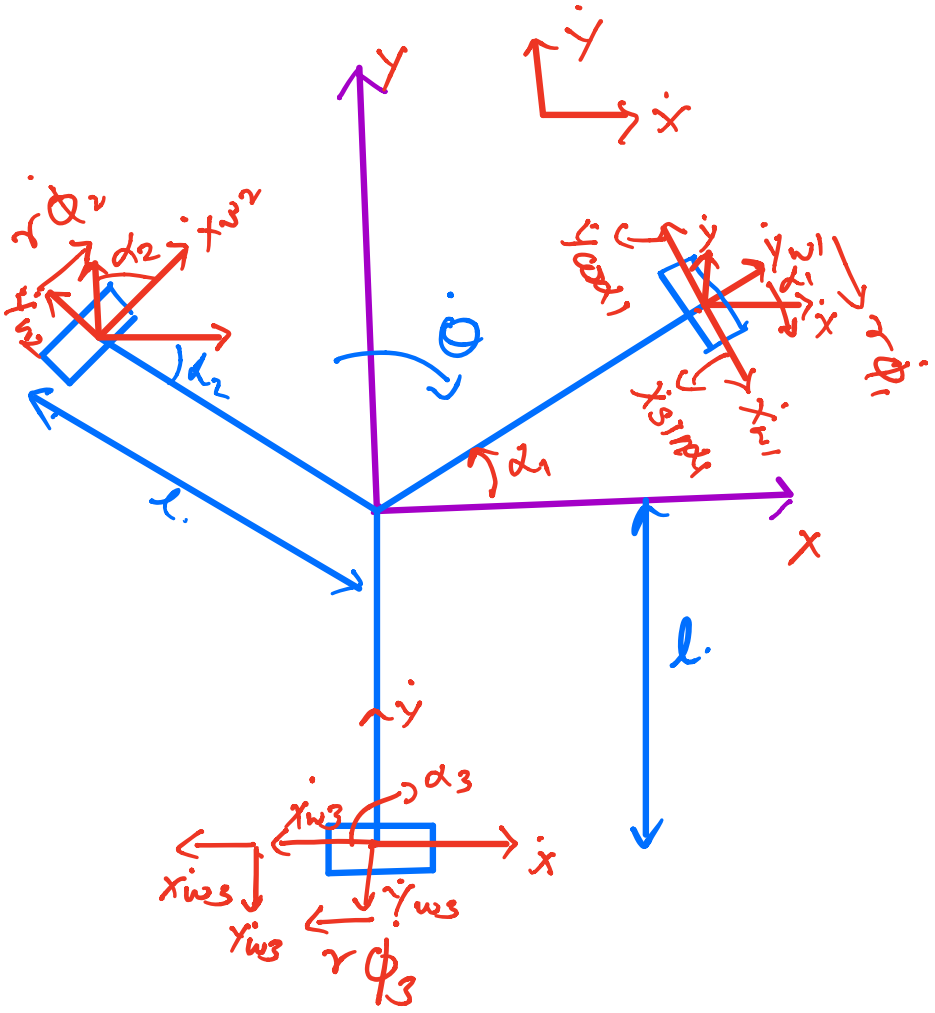


Independent control of v_x, v_y.

$$x(t) = x(0) + \int_0^t v_x \, dt \longrightarrow (17)$$

$$y(t) = y(0) + \int_0^t v_y \, dt \longrightarrow (18).$$

Omnidirectional Kinematics:



$\dot{X} = [\dot{x}\ \dot{y}\ \dot{\theta}]$ is the body center velocities.

$\dot{x}_{wi}, \dot{y}_{wi}$ is the wheel velocity of the ith wheel.

$\dot{\phi}_i$ is the angular velocity of the ith wheel.

Wheel 1: $\dot{x}\sin\alpha_1 - \dot{y}\cos\alpha_1 + l\dot{\theta} = r\dot{\phi}_1 \longrightarrow (1)$.

$\dot{x}\cos\alpha_1 + \dot{y}\sin\alpha_1 = 0 \longrightarrow (2)$.

(No lateral sliding constraint).

Wheel 2: $\dot{x}\sin\alpha_2 + \dot{y}\cos\alpha_2 + l\dot{\theta} = r\dot{\phi}_2 \longrightarrow (3)$.

$\dot{x}\cos\alpha_2 - \dot{y}\sin\alpha_2 = 0 \longrightarrow (4)$.

(No lateral sliding constraint)

Wheel 3: $\dot{X}\cos(90+\alpha_3) + \dot{Y}\cos\alpha_3 + l\dot{\theta} = r\dot{\phi}_3 \rightarrow (5)$

$\dot{X}\sin(90+\alpha_3) + \dot{Y}\sin\alpha_3 = 0 \rightarrow (6)$

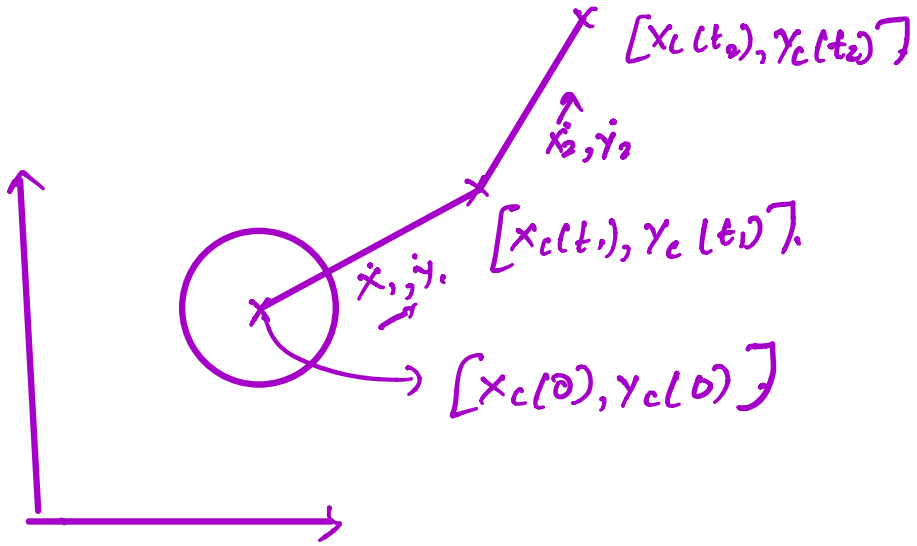
Taking together 1, 3 & 5 we get

$$\begin{bmatrix} \sin\alpha_1 & -\cos\alpha_1 & l \\ \sin\alpha_2 & \cos\alpha_2 & l \\ \cos(90+\alpha_3) & \cos\alpha_3 & l \end{bmatrix} \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} r\dot{\phi}_1 \\ r\dot{\phi}_2 \\ r\dot{\phi}_3 \end{bmatrix}$$

$\rightarrow (7)$.

or $J\dot{X}_p = \dot{X}_w \rightarrow (8)$.

$\dot{X}_p = [\dot{X}\ \dot{Y}\ \dot{\theta}]^T \quad \dot{X}_w = [r\dot{\phi}_1\ r\dot{\phi}_2\ r\dot{\phi}_3]^T$.



$\dot{X}_1, \dot{Y}_1 \xrightarrow{a_x, a_y} \dot{X}_2, \dot{Y}_2$

$\dot{X}_1(t) = \dot{X}_1(t_0) + a_x(t - t_0)$ until you reach $\dot{x}_2$

$\dot{Y}_1(t) = \dot{Y}_1(t_0) + a_y(t - t_0)$ until you reach $\dot{y}_2$

$$X_1(t) = \int_{t_0}^{t} \dot{x}_1(t)\,dt = x_1(t_0) + \dot{x}_1(t-t_0) + \frac{a_x}{2}(t-t_0)^2$$

$$Y_1(t) = \int_{t_0}^{t} \dot{y}_1(t)\,dt = y_1(t_0) + \dot{y}_1(t-t_0) + \frac{a_y}{2}(t-t_0)^2$$

Kinodynamic Motion Planning

Lecture 4

Kinodynamic Motion Planning

- Planning in robotics is often not just a geometrically feasible path from one location to another. When kinematic or dynamic constraints are present, planning becomes non-trivial. The simple car-parking problem illustrates this.

Introduction (contd.)

Kinematics

kinematics

The effect of a robot's geometry on its motion.

If the motors move ***this*** much, where will the robot be?

Assumes that we control ***encoder readings***…

dynamics

The effect of all forces (internal and external) on a robot's motion.

If the motors apply ***this*** much force, where will the robot be?

Assumes that we control ***motor current***…

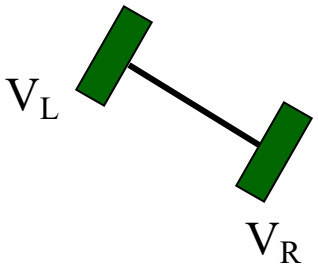
Differential drive

Most common kinematic choice

Most miniature robots…

ER1, Pioneer, Rug warrior

- difference in wheels' speeds determines its turning angle



Questions (forward kinematics)

Given the wheel's velocities or positions, what is the robot's velocity/position ?

Are there any inherent system constraints?

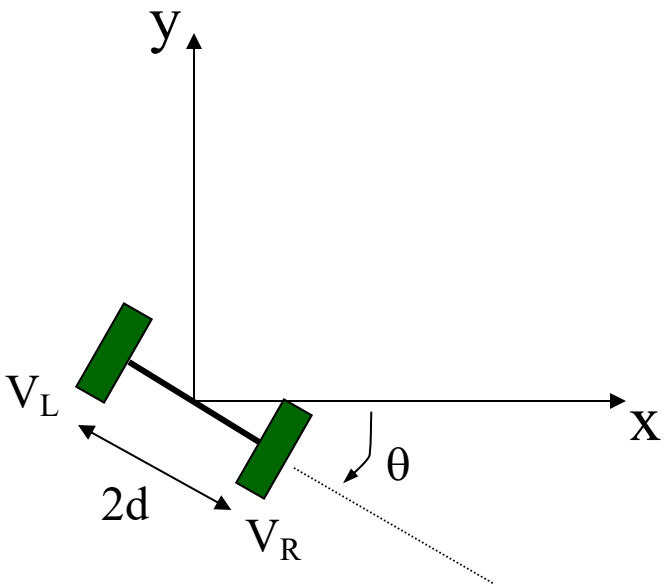
1) Specify system measurements

2) Determine the point (the radius) around which the robot is turning.

3) Determine the speed at which the robot is turning to obtain the robot velocity.

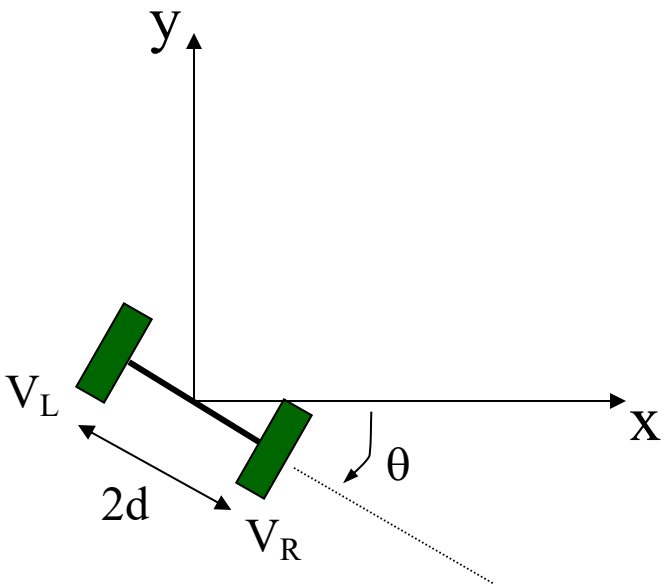
4) Integrate to find position.

Differential drive



1) Specify system measurements
   - consider possible coordinate systems

Differential drive

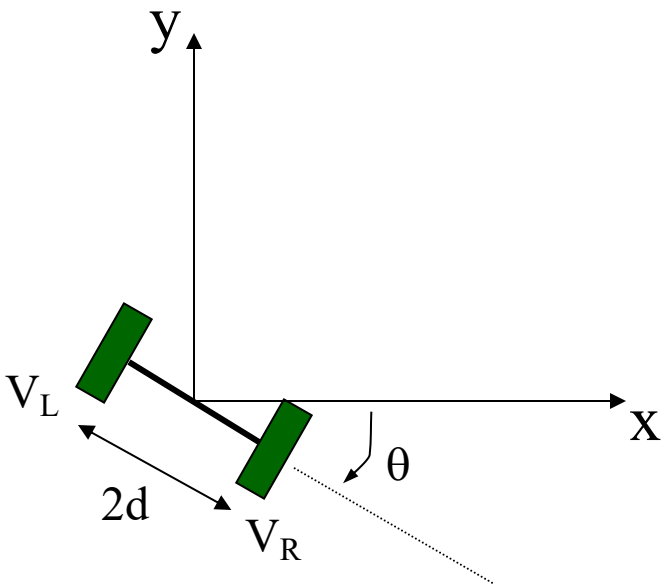


1) Specify system measurements
   - consider possible coordinate systems

---

2) Determine the point (the radius) around which the robot is turning.

Differential drive



1) Specify system measurements
   - consider possible coordinate systems

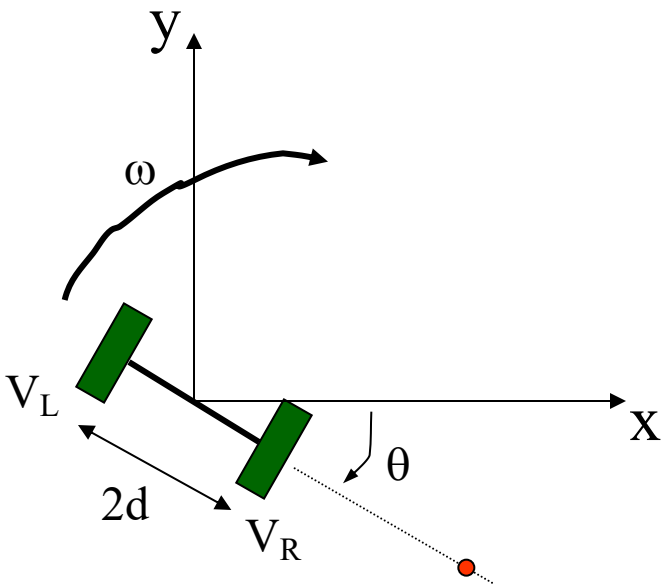
---

2) Determine the point (the radius) around which the robot is turning.

- to minimize wheel slippage or lateral motion, the instantaneous center of curvature (the **ICC**) must lie at the intersection of the wheels' axles

- each wheel must be traveling at the same angular velocity

Differential drive



1) Specify system measurements
   - consider possible coordinate systems

---

2) Determine the point (the radius) around which the robot is turning.

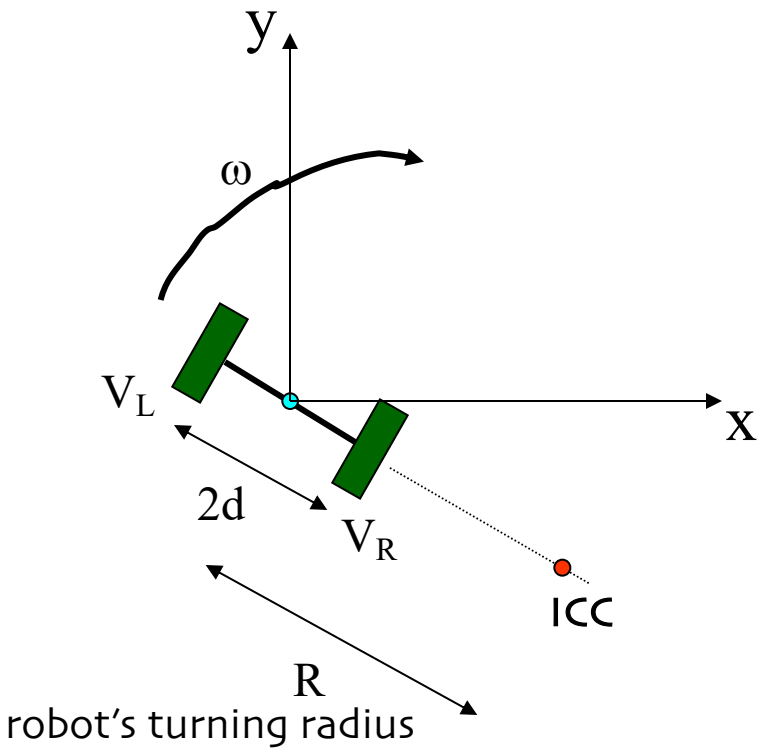
- to minimize wheel slippage, this point (the **ICC**) must lie at the intersection of the wheels' axles

- each wheel must be traveling at the same angular velocity **around the ICC**

ICC "instantaneous center of curvature"

(assume the wheel diameter is accounted for already)

Differential drive



1) Specify system measurements
   - consider possible coordinate systems

---

2) Determine the point (the radius) around which the robot is turning.

   - each wheel must be traveling at the same angular velocity around the ICC

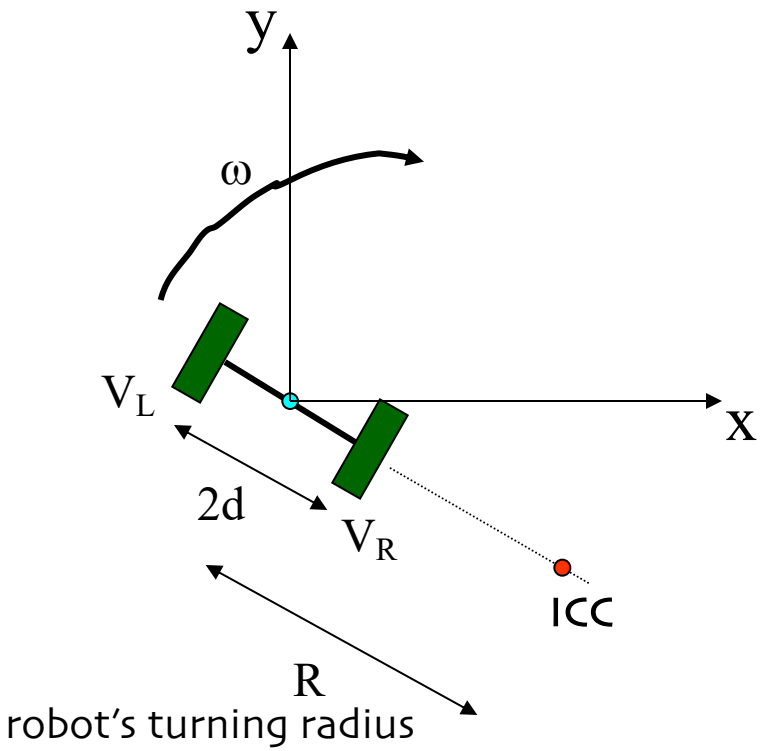
---

3) Determine the robot's speed around the ICC and its linear velocity

$$\omega(R+d) = V_L$$

$$\omega(R-d) = V_R$$

Differential drive



1) Specify system measurements
   - consider possible coordinate systems

---

2) Determine the point (the radius) around which the robot is turning.

   - each wheel must be traveling at the same angular velocity **around the ICC**

---

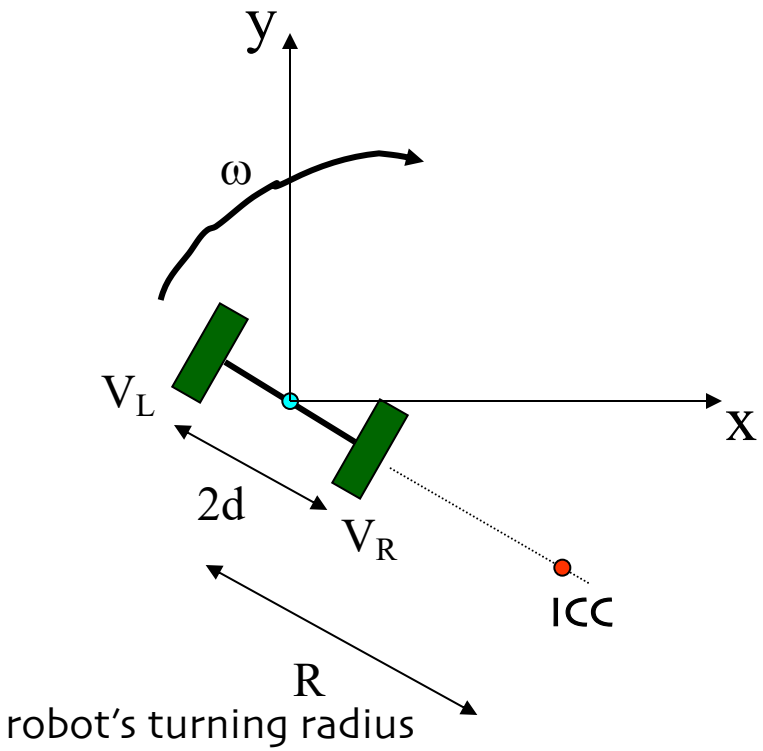
3) Determine the robot's speed around the ICC and its linear velocity

$$\omega(R+d) = V_L$$

$$\omega(R-d) = V_R$$

of these five, what's known & what's not?

Differential drive



1) Specify system measurements
   - consider possible coordinate systems

2) Determine the point (the radius) around which the robot is turning.

   - each wheel must be traveling at the same angular velocity around the ICC

3) Determine the robot's speed around the ICC and its linear velocity

$$\omega(R+d) = V_L$$

$$\omega(R-d) = V_R$$

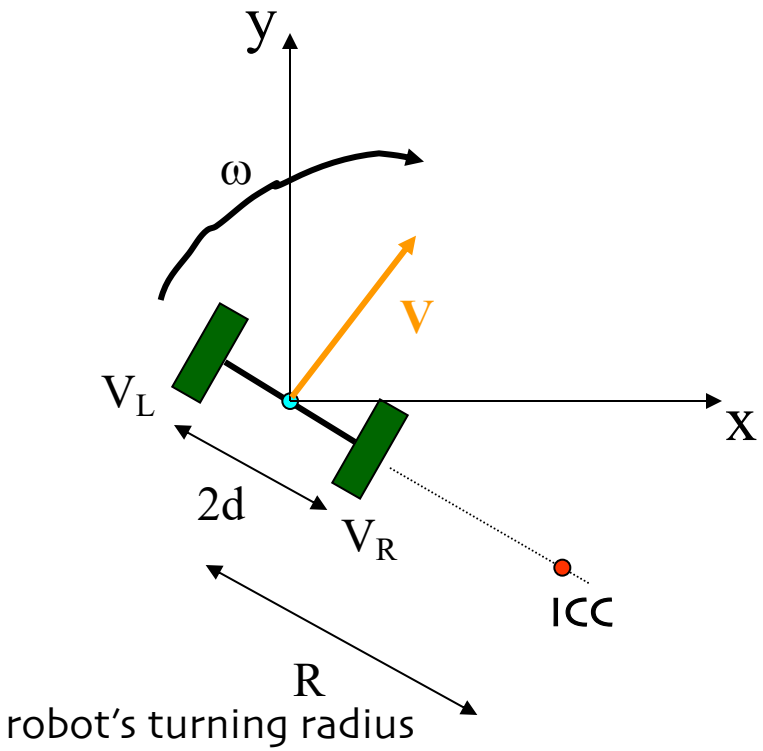
Thus,

$$\omega = (V_R - V_L) / 2d$$

$$R = d(V_R + V_L) / (V_R - V_L)$$

are there interesting cases?

Differential drive



1) Specify system measurements
   - consider possible coordinate systems

2) Determine the point (the radius) around which the robot is turning.

   - each wheel must be traveling at the same angular velocity around the ICC

3) Determine the robot's speed around the ICC and its linear velocity

$$\omega(R+d) = V_L$$

$$\omega(R-d) = V_R$$

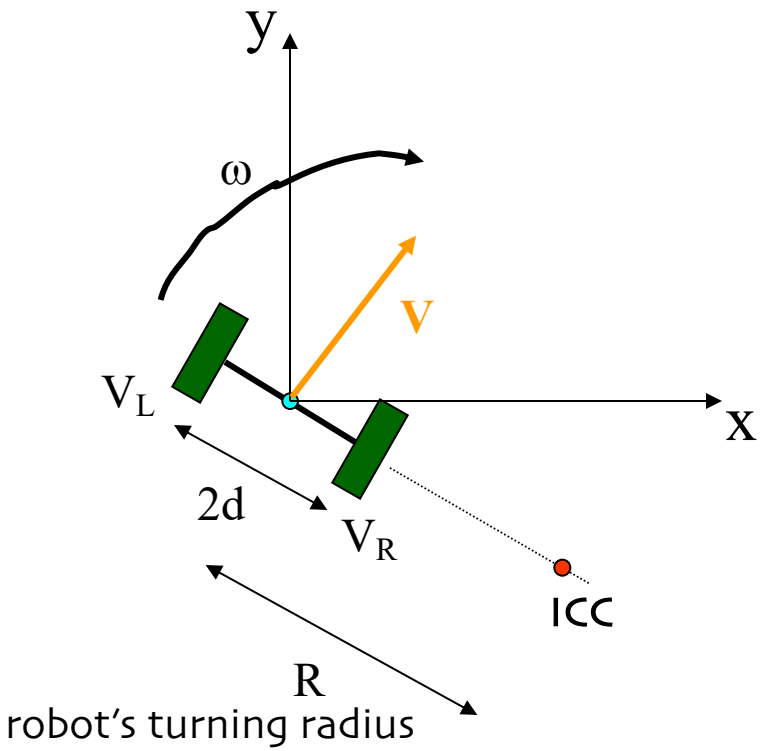
Thus,

$$\omega = (V_R - V_L) / 2d$$

$$R = d(V_R + V_L) / (V_R - V_L)$$

So, what is the robot's velocity?

Differential drive



1) Specify system measurements
   - consider possible coordinate systems

2) Determine the point (the radius) around which the robot is turning.

   - each wheel must be traveling at the same angular velocity around the ICC

3) Determine the robot's speed around the ICC and its linear velocity

$$\omega(R+d) = V_L$$

$$\omega(R-d) = V_R$$

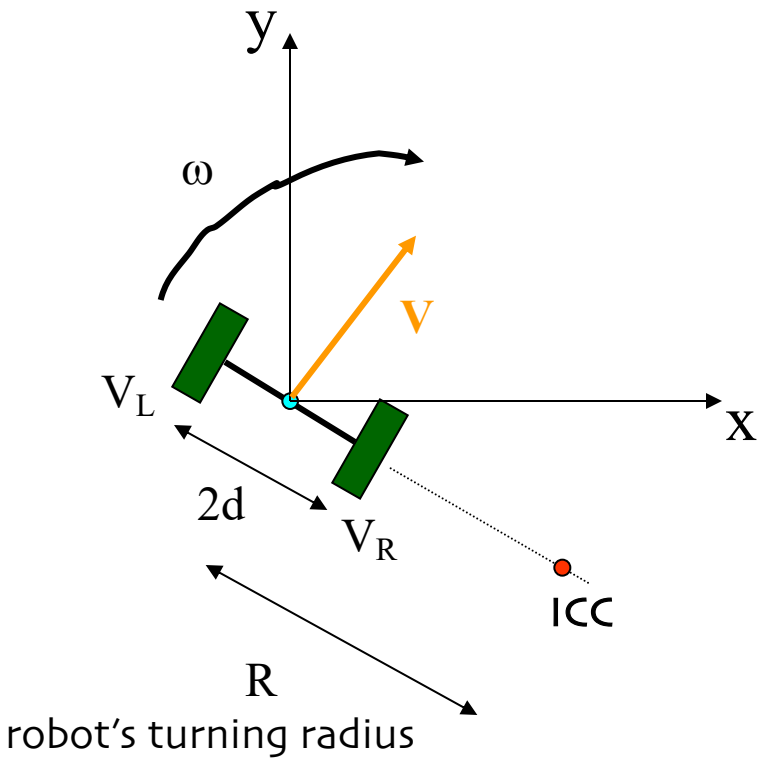
Thus,

$$\omega = (V_R - V_L) / 2d$$

$$R = d(V_R + V_L) / (V_R - V_L)$$

So, the robot's velocity is $V = \omega R = (V_R + V_L) / 2$

Differential drive



4) Integrate to obtain position

$$V_x = V \cos(\theta)$$

$$V_y = V \sin(\theta)$$

with

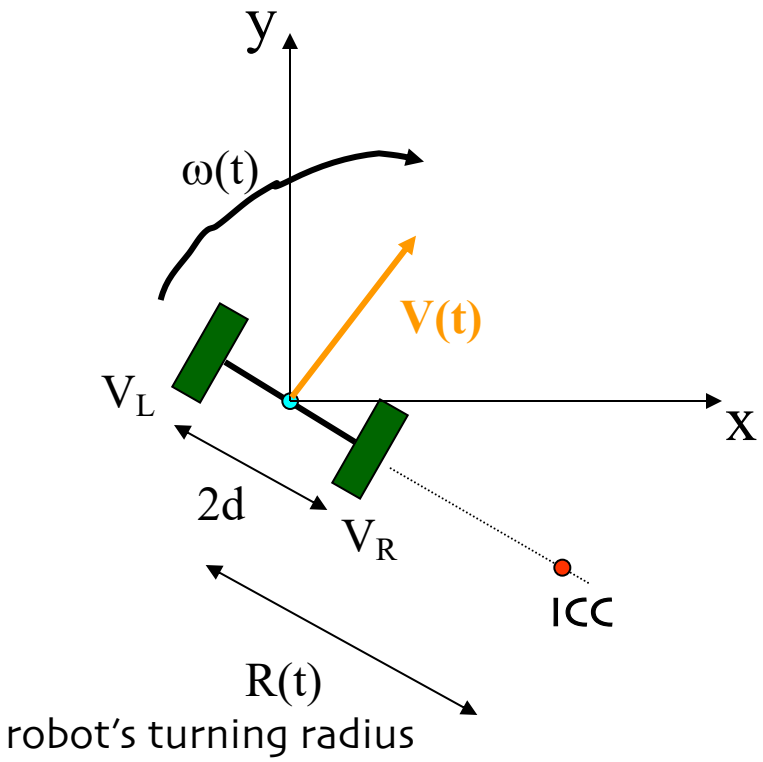
$$\omega = (V_R - V_L) / 2d$$

$$R = d(V_R + V_L) / (V_R - V_L)$$

$$V = \omega R = (V_R + V_L) / 2$$

What has to happen to change the ICC ?

Differential drive



4) Integrate to obtain position

$$V_x = V(t)\cos(\theta(t))$$

$$V_y = V(t)\sin(\theta(t))$$

with

$$\omega = (V_R - V_L)/2d$$

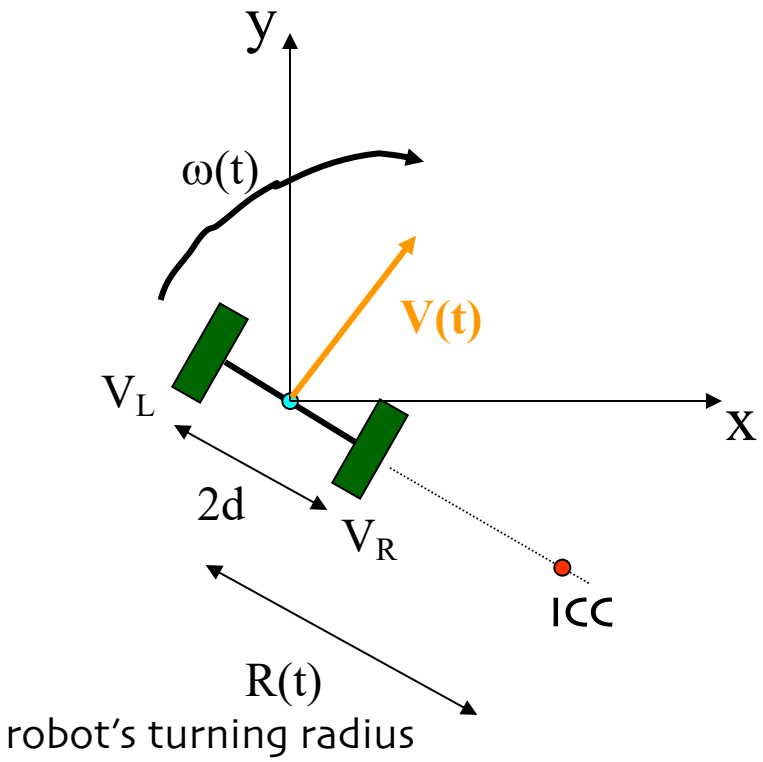
$$R = d(V_R + V_L)/(V_R - V_L)$$

$$V = \omega R = (V_R + V_L)/2$$

What has to happen to change the ICC ?

things have to change over time, **t**

Differential drive



4) Integrate to obtain position

$$V_x = V(t)\cos(\theta(t))$$

$$V_y = V(t)\sin(\theta(t))$$

Thus,

$$x(t) = \int V(t)\cos(\theta(t))\,dt$$

$$y(t) = \int V(t)\sin(\theta(t))\,dt$$

$$\theta(t) = \int \omega(t)\,dt$$

with

$$\omega = (V_R - V_L)/2d$$

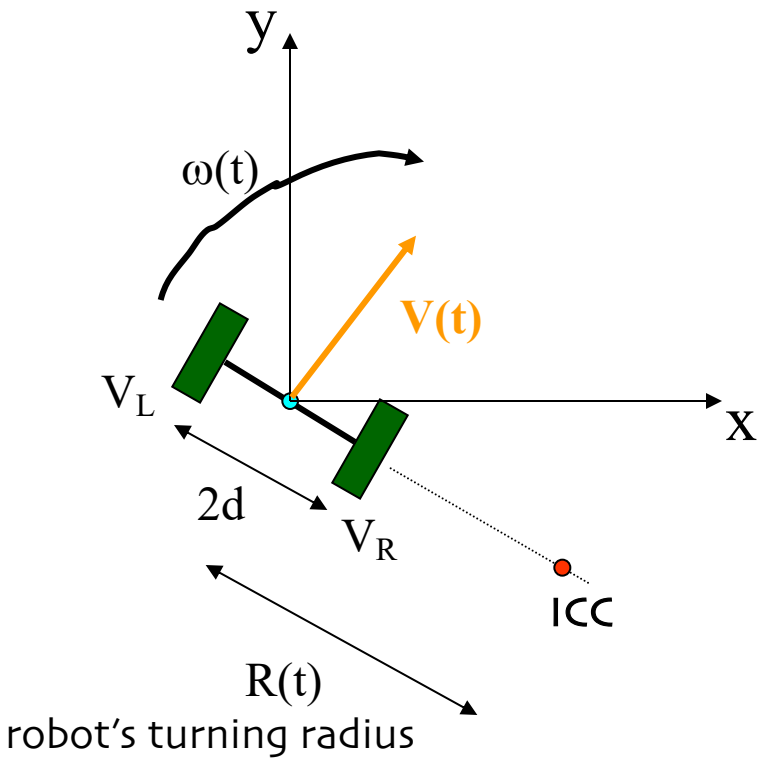
$$R = d(V_R + V_L)/(V_R - V_L)$$

$$V = \omega R = (V_R + V_L)/2$$

What has to happen to change the ICC ?

things have to change over time, **t**

Differential drive



4) Integrate to obtain position

$$V_x = V(t)\cos(\theta(t))$$

$$V_y = V(t)\sin(\theta(t))$$

Thus,

$$x(t) = \int V(t)\cos(\theta(t))\,dt$$

$$y(t) = \int V(t)\sin(\theta(t))\,dt$$

$$\theta(t) = \int \omega(t)\,dt$$

Kinematics

with

$$\omega = (V_R - V_L)/2d$$

$$R = d(V_R + V_L)/(V_R - V_L)$$

$$V = \omega R = (V_R + V_L)/2$$

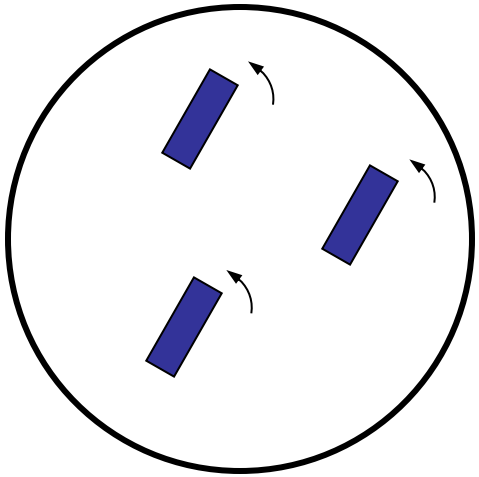
What has to happen to change the ICC ?

things have to change over time, **t**

Synchro drive

Nomad 200

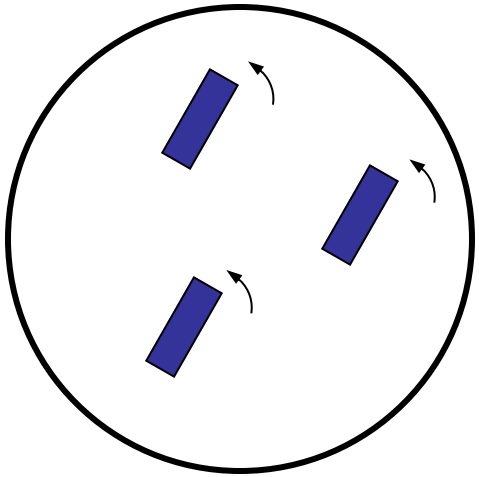
wheels rotate in tandem and remain parallel
all of the wheels are driven at the same speed



Synchro drive

Nomad 200

wheels rotate in tandem and remain parallel
all of the wheels are driven at the same speed



Where is the ICC ?

Questions (forward kinematics)

Given the wheel's velocities or positions,
what is the robot's velocity/position ?

Are there any inherent system constraints?

1) Specify system measurements

2) Determine the point (the radius) around which the robot is turning.

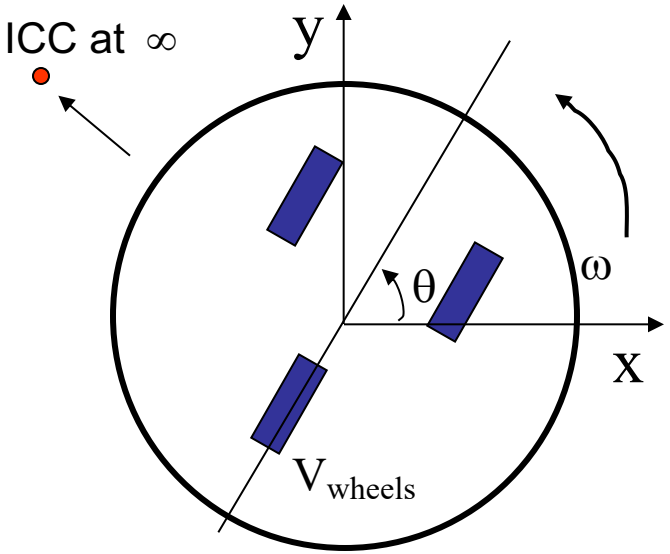
3) Determine the speed at which the robot is turning to obtain the robot velocity.

4) Integrate to find position.

Synchro drive

Nomad 200

wheels rotate in tandem and remain parallel
all of the wheels are driven at the same speed



$$V_{robot} = V_{wheels}$$
$$\omega_{robot} = \omega_{wheels}$$

} velocity

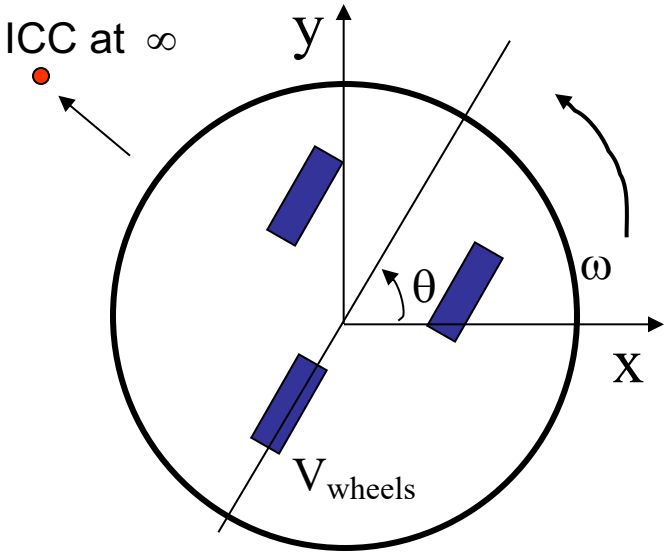
$$\theta(t) = \int \omega(t)\, dt$$
$$x(t) = \int V_{wheels}(t) \cos(\theta(t))\, dt$$
$$y(t) = \int V_{wheels}(t) \sin(\theta(t))\, dt$$

} position

Synchro drive

Nomad 200

wheels rotate in tandem and remain parallel
all of the wheels are driven at the same speed



$$V_{robot} = V_{wheels}$$
$$\omega_{robot} = \omega_{wheels}$$

velocity

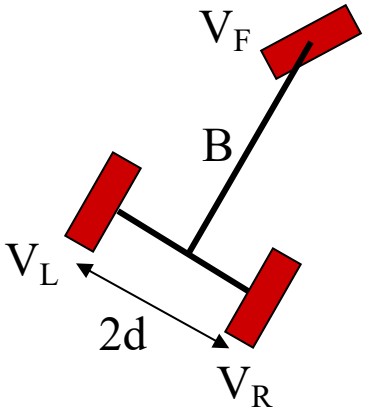
Kinematics

$$\theta(t) = \int \omega(t)\, dt$$
$$x(t) = \int V_{wheels}(t) \cos(\theta(t))\, dt$$
$$y(t) = \int V_{wheels}(t) \sin(\theta(t))\, dt$$

position

Tricycle drive

Mecos tricycle-drive robot



- The velocity of the front wheel is V_F

  (i.e., the conversion from angular velocity with wheel diameter is already done…)

- We know the distances 2d and B

- What else do we need to know?
- Where is the ICC?
- How fast is the trikebot rotating around it?
- What are V_L and V_R ?

Tricycle drive

Mecos tricycle-drive robot

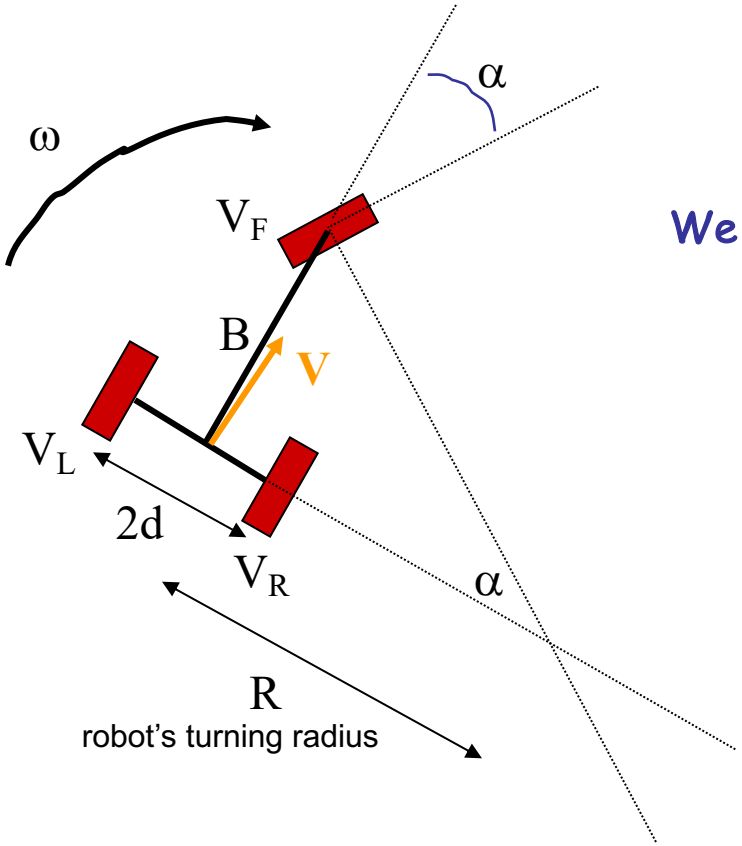
- front wheel is powered and steerable
- back wheels tag along...

Starting from

α, the robot's steering angle

$V_F = \omega B / \sin(\alpha)$

We conclude



$\omega = V_F \sin(\alpha) / B$

$R = B / \tan(\alpha)$

Kinematics

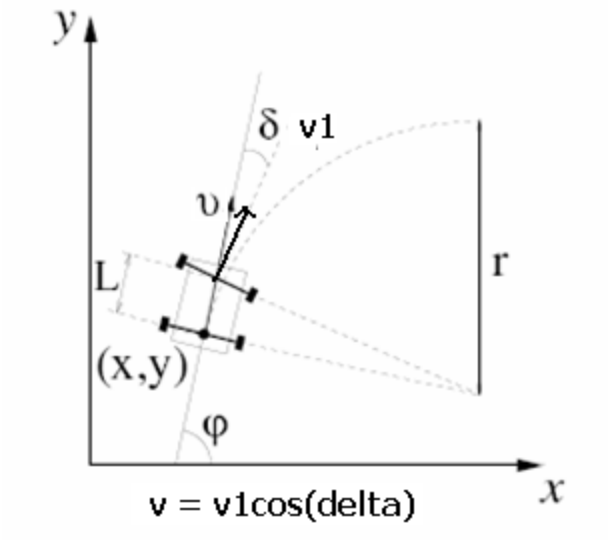
$V = \omega R = V_F \cos(\alpha)$

$V_L = \omega(R+d) = V + dV_F \sin(\alpha) / B$

$V_R = \omega(R-d) = V - dV_F \sin(\alpha) / B$

(velocity only)

Kinematics of the Car



$$\begin{aligned} \dot{x} &= v\cos\varphi \\ \dot{y} &= v\sin\varphi \\ \dot{\varphi} &= \frac{v}{L}\tan\delta. \end{aligned}$$

Integrating above equations you will find the car moves in a circle of radius Lcot(delta)

Delta = 0 corresponds to an infinite turning radius or a straight line,

NOTE:

Since the linear velocity appears in the time evolution of all the state variables, this car model does not have the possibility to make arbitrary rotations while standing still in $\mathbb{R}^2$. It is only able to follow paths that are at least continuously differentiable.

Solution 1: Minimum Reversals Method

1. Let initial state or configuration of the robot be X0. Add this state to a tree T
2. Generate discrete successors for the current state X (a leaf node of T) by varying the steering angle delta in the kinematic equations for the car. Each delta gives a successor node, call it Xn.
3. Among the generated successors Xn (all the leaf nodes of T so far) choose the node with minimum number of reversals and insert it into a heap it is collision free. This step is repeated till a collision free successor node is found
4. Now X = top most node in the heap
5. If X is the goal state or near the goal state STOP else repeat 2 to 4 till goal is reached or heap is empty

Basic RRT Algorithm

- Suitable for rapidly exploring space
- Inefficient for one-time queries

ISSUES:

1. Choice of distance metric
2. Choice of thresholds – how much error in final position to allow
3. Choice of data-structure for search

RRT-Explore (Tree T, Vertex **src**)

1: T = {**src**}

2: for k = 1 to K //(K number of random nodes in workspace W)

3: $\mathbf{X_k}$ = Random-State( )//one such random node

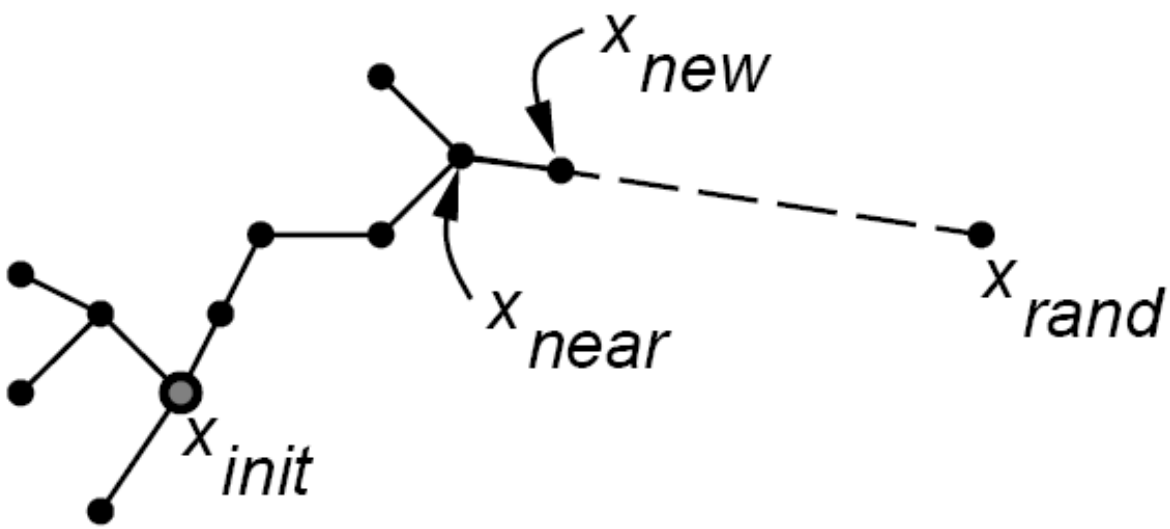
4: **Xn** = Nearest-Neighbour(T, **X**)// finds the node among the leaf nodes that is closest to X_k

5: **Xs** = SuccessorState(**Xn**, **X**, **U**) //generates a successor to Xn, Xs, that takes Xs closer to X_k on some metric

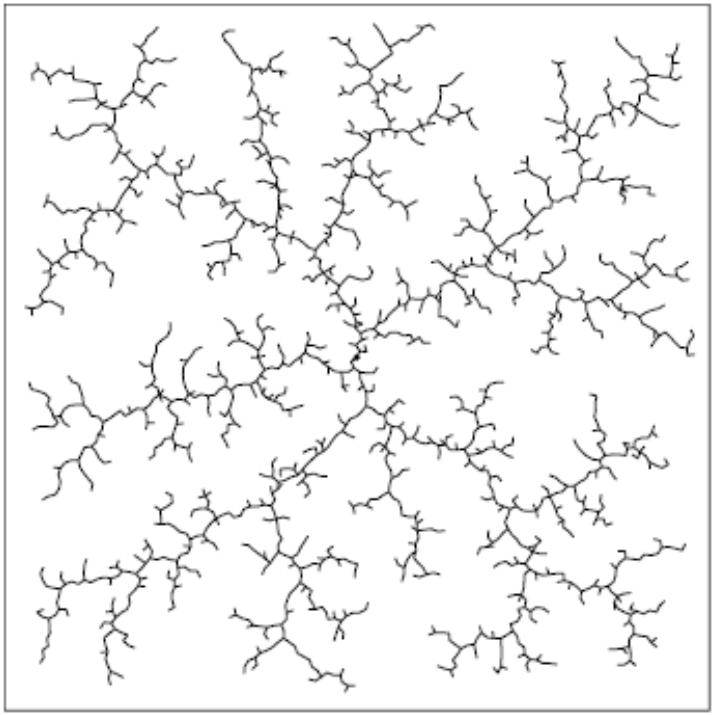
6: if(collision-free(**Xs**))

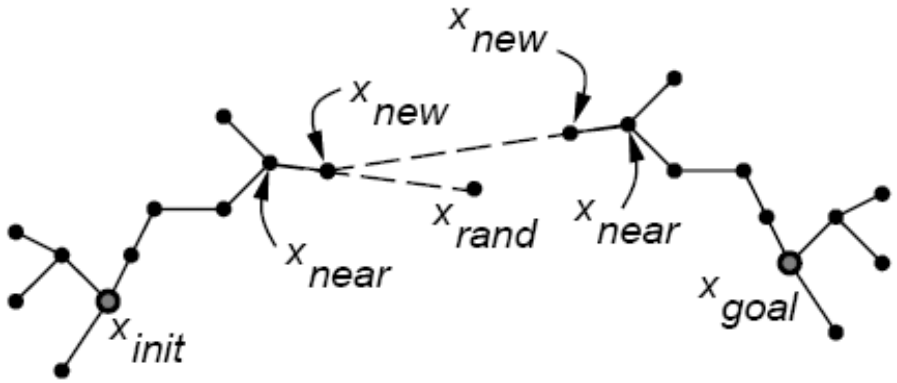
7: then T = T U { **Xs** }

RRT (illustration)



RRT and Dual RRT (Bidirectional Search)





Dual RRT with bi-directional search

Bidirectional RRT Algorithm

- Efficient for one-time queries
- Multiple rendezvous points
- Faster search

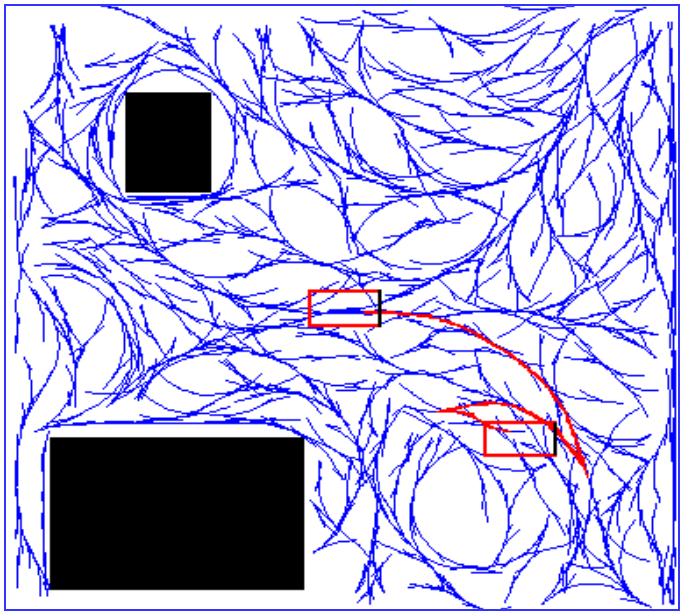
HEURISTICS USED:

Goal-bias – by a coin flip, either a random state or the target state is used.

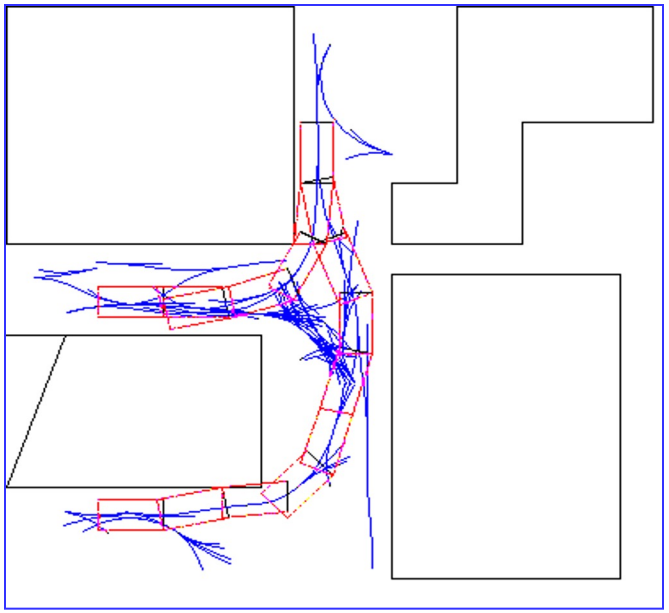
```
RRT-Create (Tree T, State src, State dest)
1:  T1 = {src};  T2 = {dest}
2:  while( true )
3:          X = Random-State( );
4:          if( Extend( T1, X, Xn) != TRAPPED )
5:          then if( Extend(T2, Xn, Xn') == REACHED)
6:                      then break;
7:          swap( T1, T2 );
```

```
Extend( Tree T, State Xr, State &Xs )
1:   Xn = Nearest-Neighbour(T, Xr)
2:   Xs = SuccessorState(Xn, X, U)
3:   if( collision-free( Xs ) )
4:   then T = T U { Xs };
5:   else return TRAPPED;
6:   if( Xs is-close-enough-to Xr )
7:   then  return REACHED;
8:   else return ADVANCED;
```

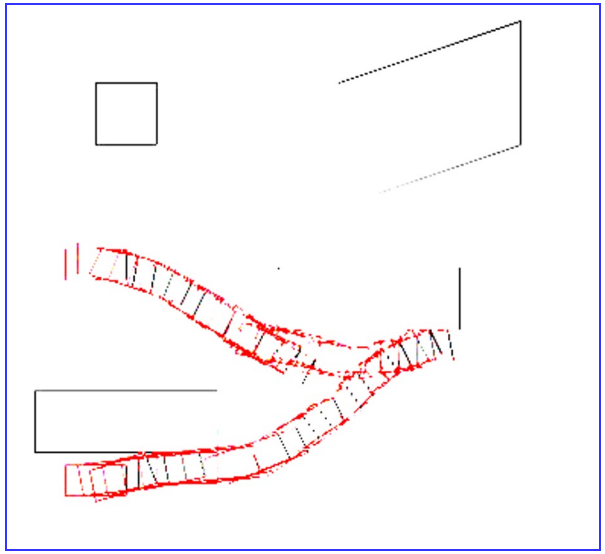
Results with RRTs



Exploring with RRTs

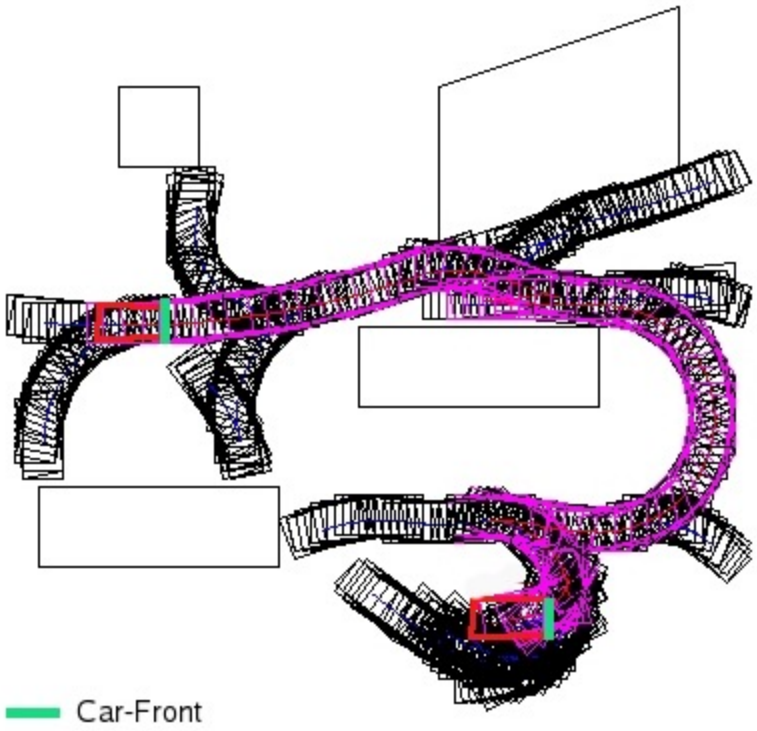


Unbiased RRT search

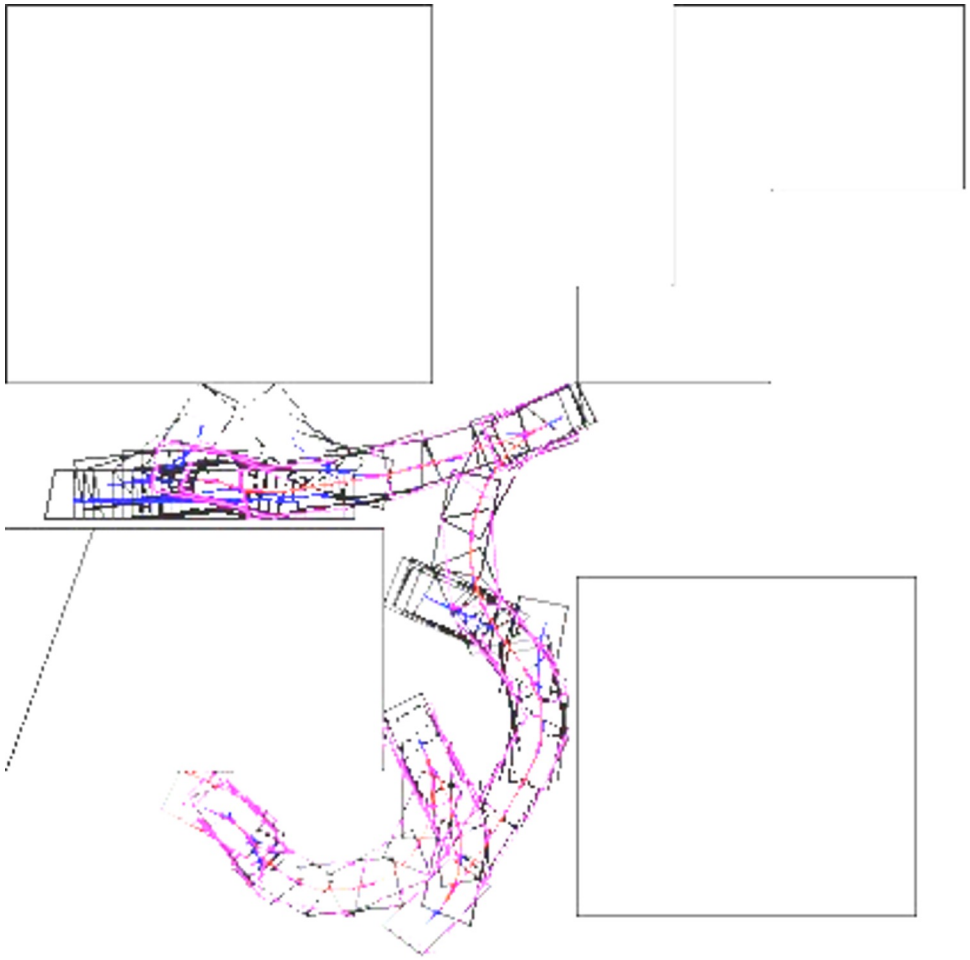


Goal-biased RRT search

Some Results



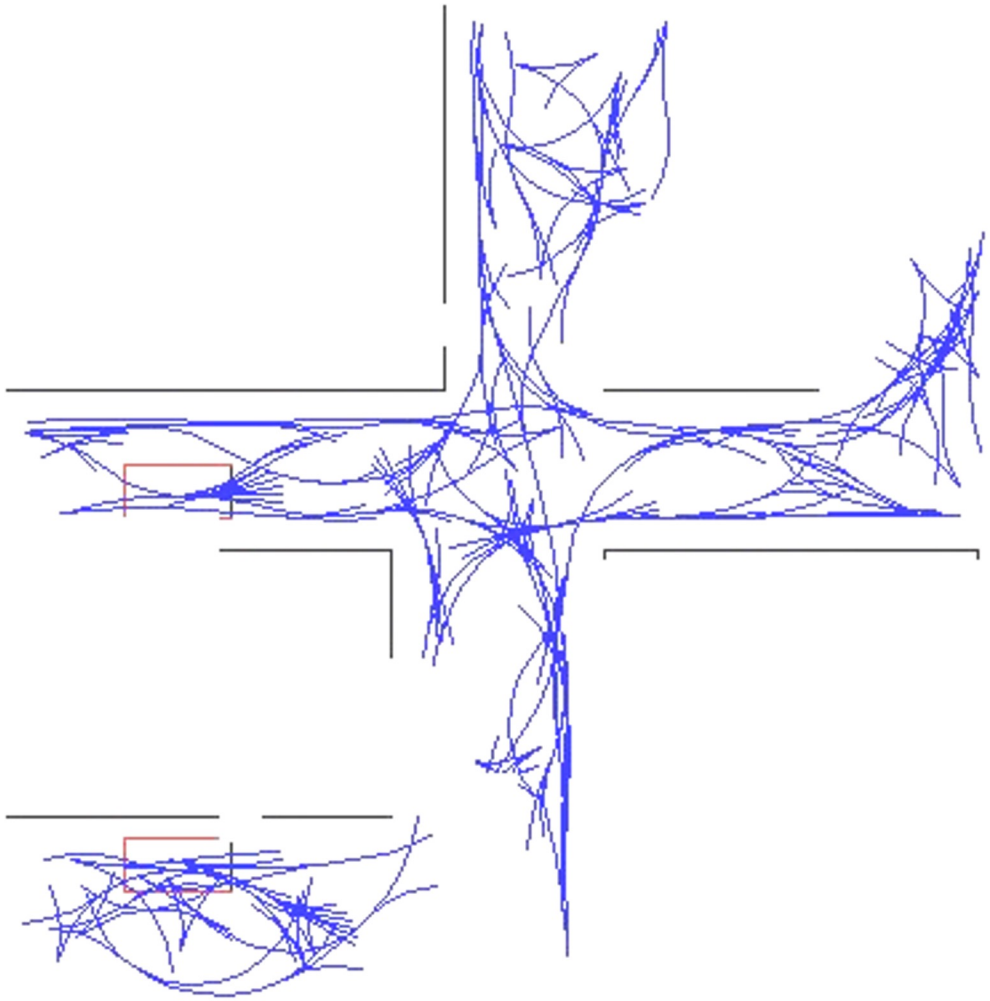
Results with RRTs



Unbiased bi-directional search usually yields such sub-optimal paths as the one shown in the figure.

The black pink represents the actual car path, while he black represents the exploration conducted by the tree

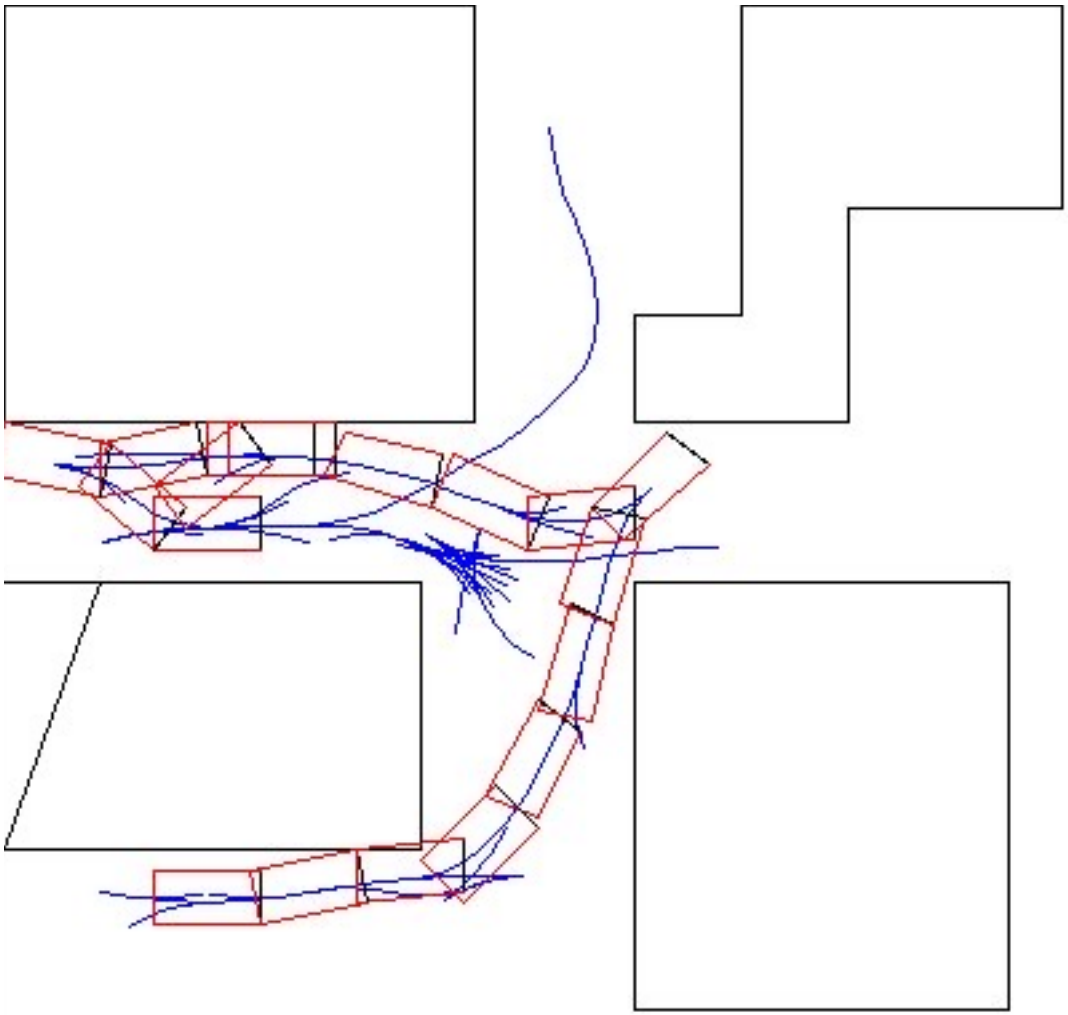
Results with RRTs



Unbiased bi-directional search is great for exploring in the state-space.

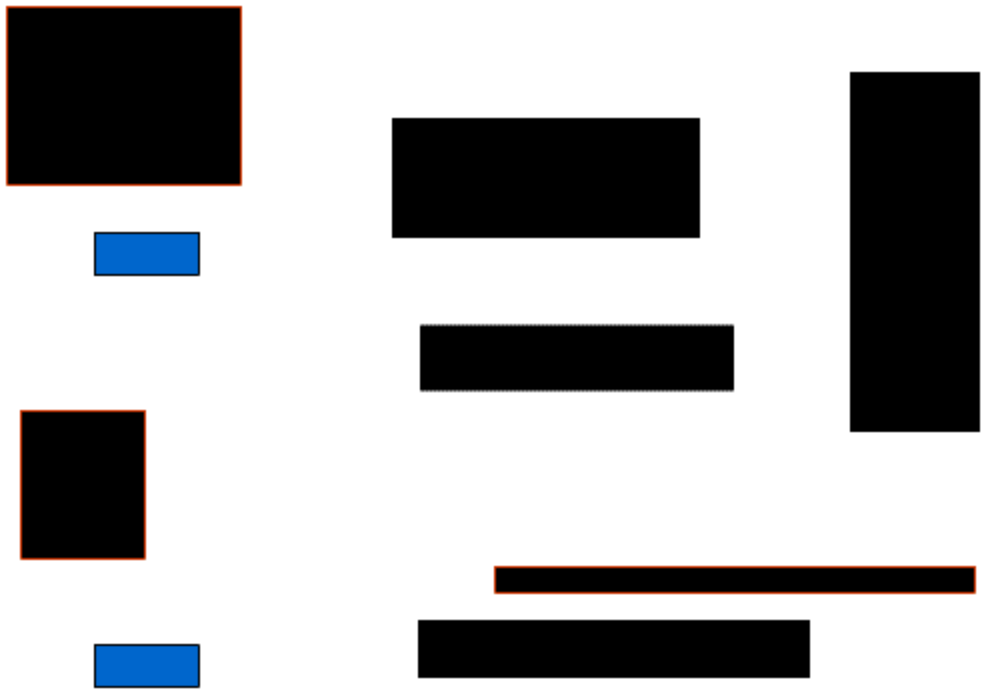
A possible application could be store the pre-explored map as an incomplete planner.

Results with RRTs



Biased (goal-biased) bi-directional search gives good results for difficult cases.

Results with RRTs



Problems with RRT

- The RRT bypasses the need for a local planner. Hence it is always difficult to reach the final state exactly or even nearly exactly through a discrete search
- The RRT uses a metric by which it decides the node closest to the random state. However many a time deciding the metric is equivalent to solving the motion planning problem as such. In other words there is no suitable or appropriate metric often to decide closeness between two states.
- Bidirectional RRT: Find it out☺

The FR SteerBot at iRL

- Has both front and reer wheel steer
- Gives rise to a smaller turning radius (can navigate in tighter spaces)
- Also has the property of parallel steer

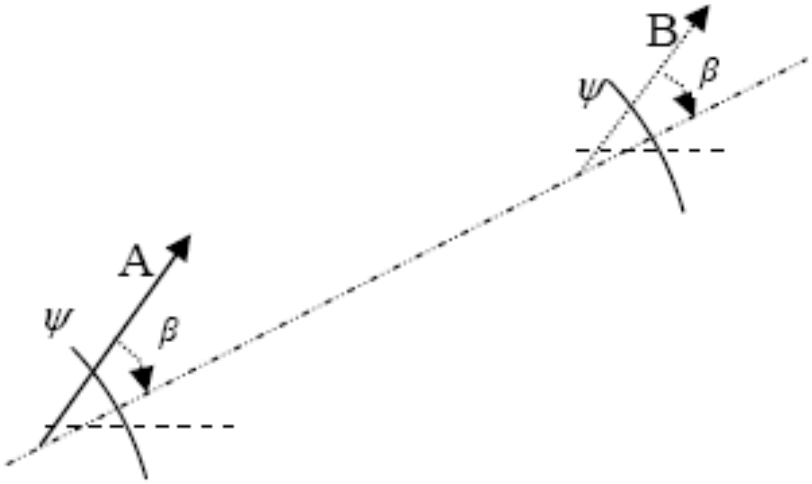
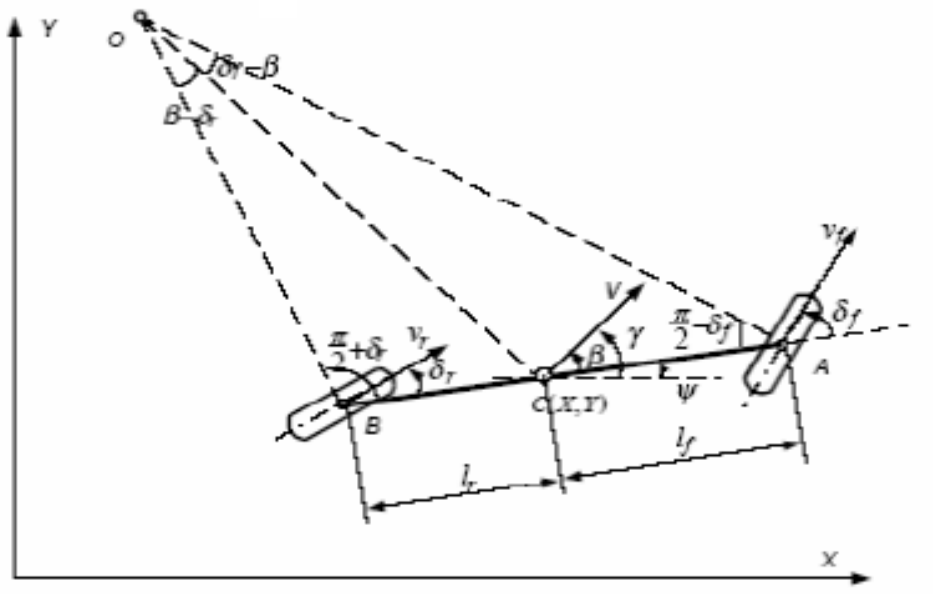


Fig. 2 Parallel steer at β slip

Kinematics of the FR Bot

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} v\cos(\psi+\beta) \\ v\sin(\psi+\beta) \\ \dfrac{v\cos\beta(\tan\delta_f - \tan\delta_r)}{l_f + l_r} \end{bmatrix}$$

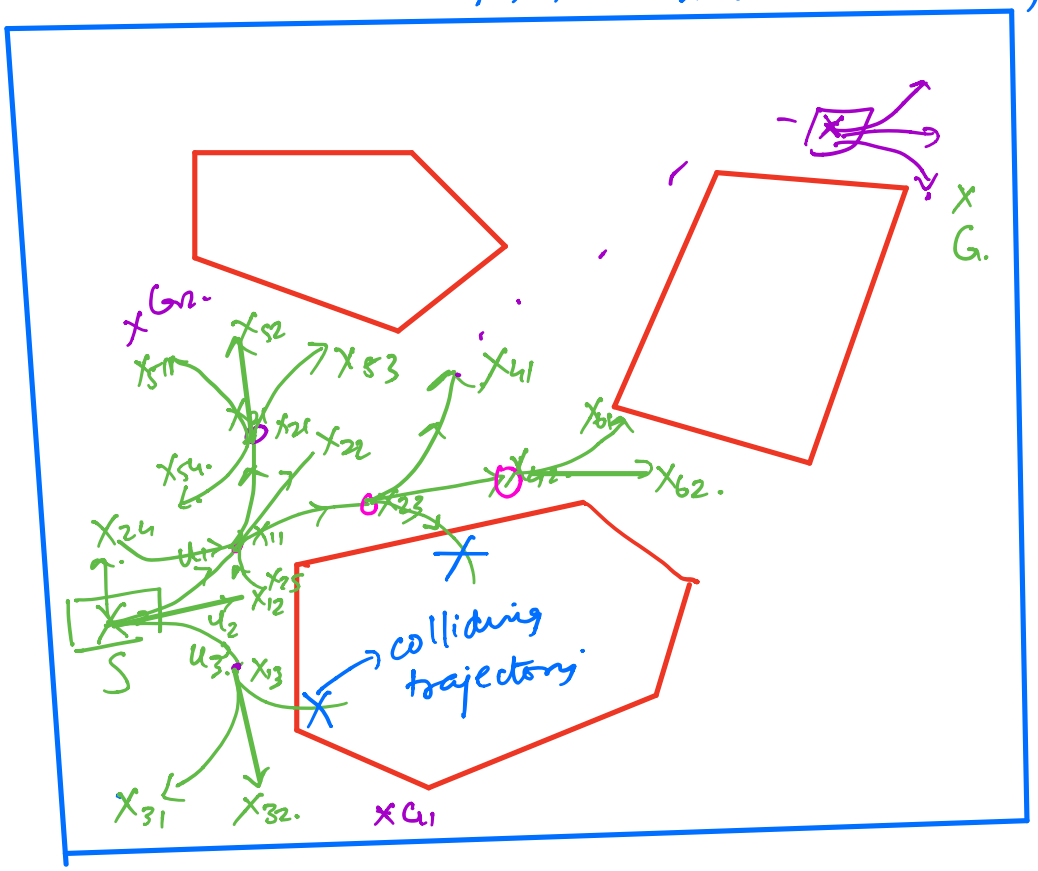
$$\frac{l_f + l_r}{\cos\beta(\tan\delta_f - \tan\delta_r)}$$



The radius of the car is given by $\dfrac{l_f + l_r}{\cos\beta(\tan\delta_f - \tan\delta_r)}$ centered at O

1). This radius is the distance from O to C. The parallel steer condition occurs when $\delta_f = \delta_r$ resulting in an infinite turn radius. This implies that the car translates without changing its orientation.

RRT → bi directional RRT, RRT*



S, G → start and goal configuration

$G_1, G_2, \ldots$ → random goals

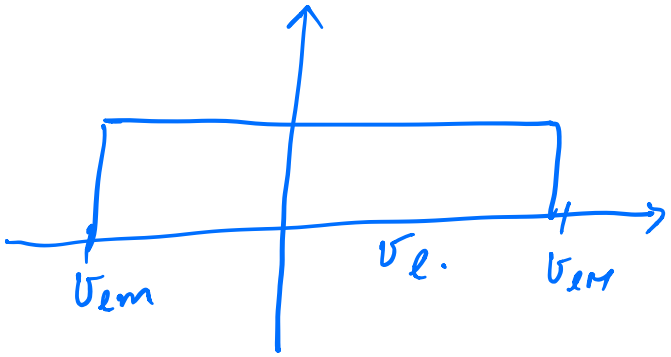
U → set of feasible controls, we can discretize and represent it as

$$U = [[v_{l1}, v_{r1}], \ldots, [v_{ln}, v_{rn}]]$$

where each v_i is s.t

$v_m \leq v_i \leq v_M$

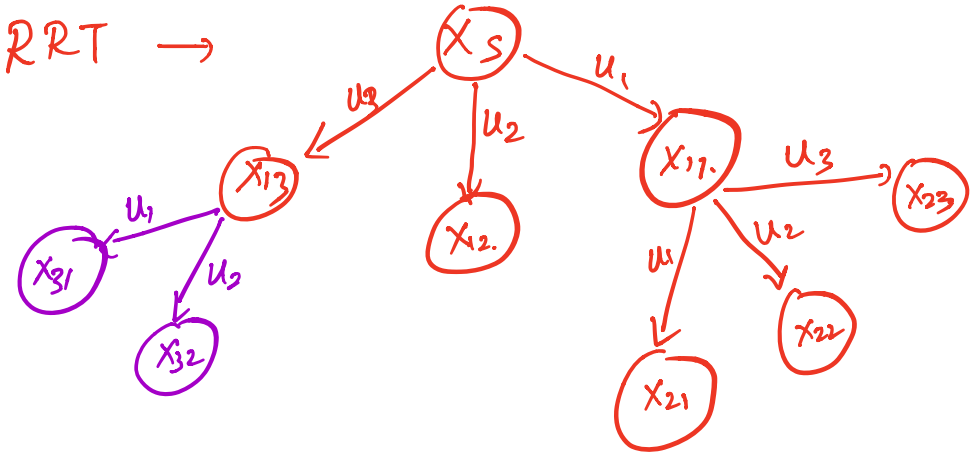
One can also think of this set as a distribution such as

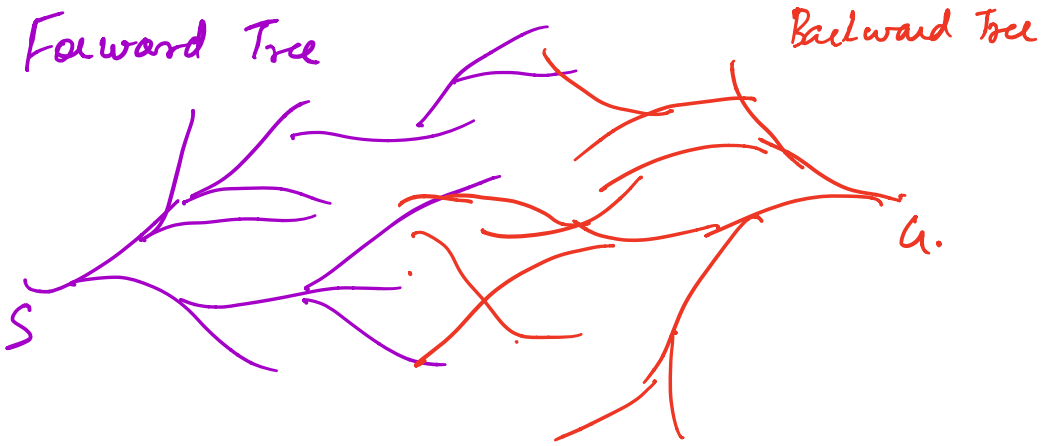


$S \rightarrow \begin{bmatrix} x_s \\ y_s \\ \theta_s \end{bmatrix}$ $\quad G \rightarrow \begin{bmatrix} x_g \\ y_g \end{bmatrix}$

$X_{11} = f(X_s, u_1, dt) \rightarrow$ differential drive kinematics

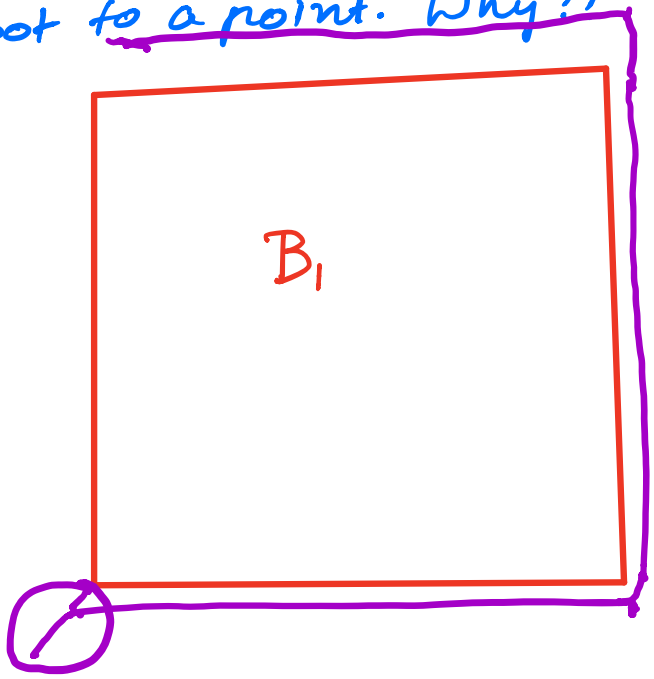
RRT $\rightarrow$

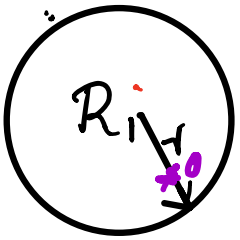


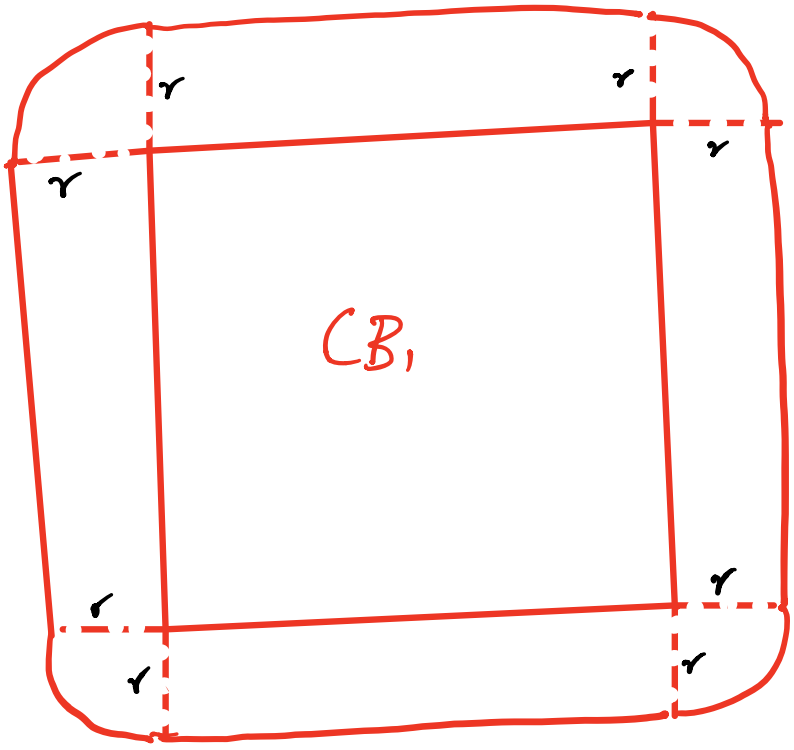

Forward Tree
Backward Tree
s
a.

CONFIGURATION SPACE:

Main Idea: Grow the obstacle by the robot's size. Reduce the robot to a point. Why??







CB_1: The set of all points for which $B_1 \cap R_1 = \phi$.

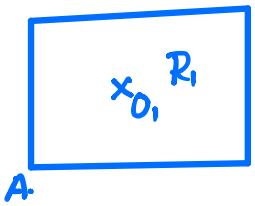
CB_1: The configuration space of the obstacle B_1

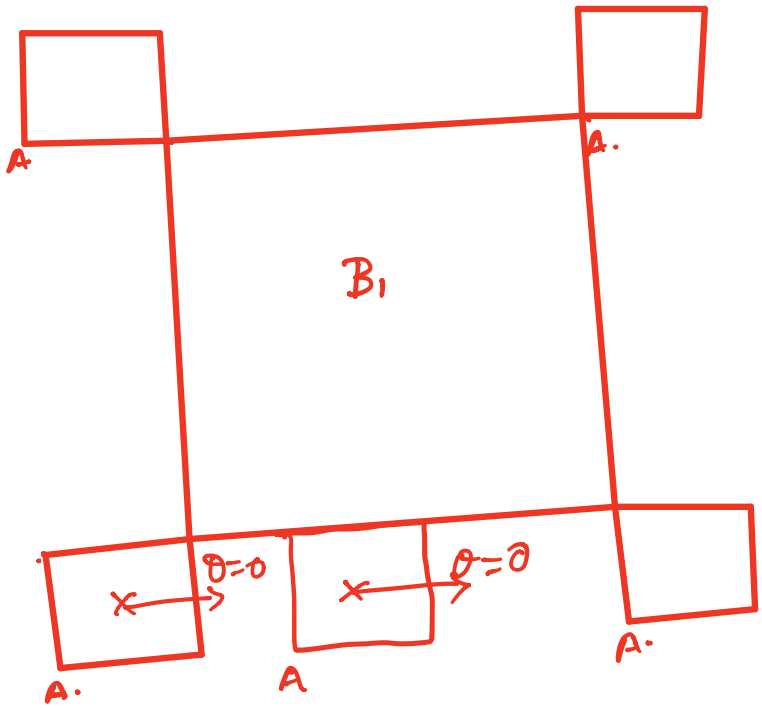
How to obtain CB_1: Trace the locus of all points for which the robot R_1 just grazes the obstacle B_1. The locus is always traced with respect to a fixed point in R_1. In

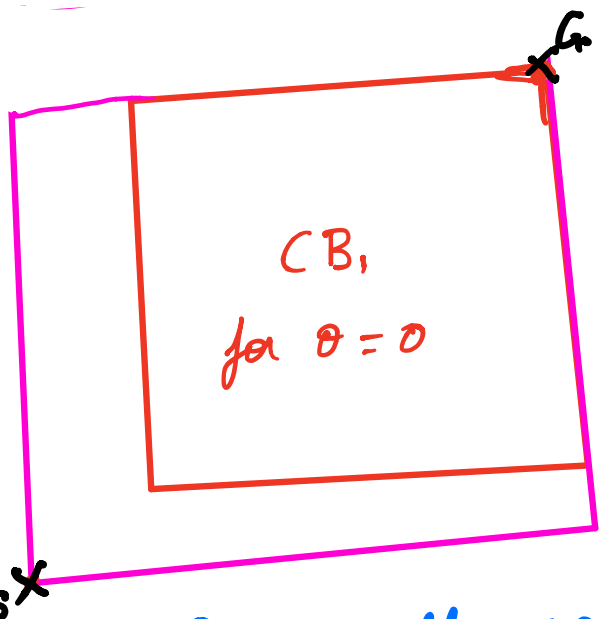
this case it is traced with respect to the center of R_1, say O_1.

How to obtain CB_1 for a non-circular robot.

Say rectangular :







The red quad is the original obstacle B_1

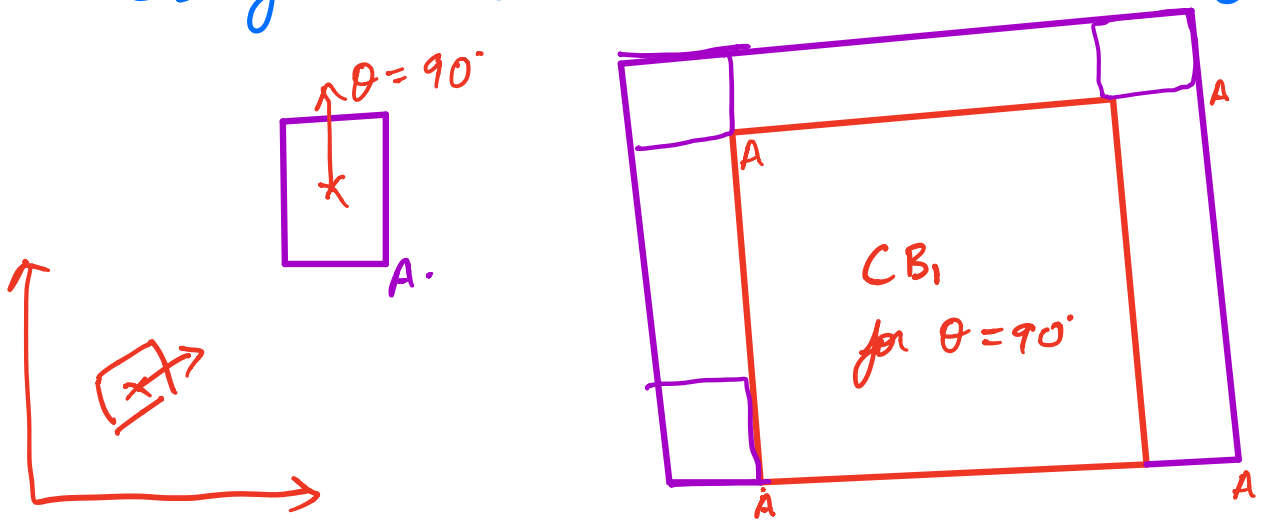
The pink lines trace the locus of the point A for which R_1 grazes B_1

CB_1 is the grown obstacle.

How is CB actually coded: There are libraries that compute CB for a two dimensional /planar robot that can be considered a circle or a polygon.

What is the dimension in which CB resides?

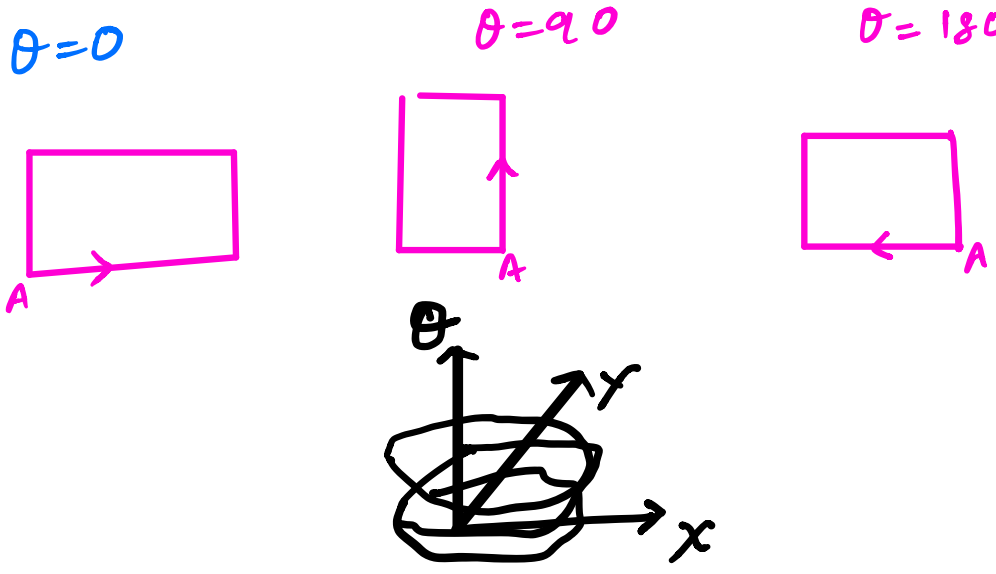
CB for a different orientation of R_1



Now what is the dimension of CB_1

- The set of all points of A for which R_1 grazes B_1.

- For different orientations of R_1 we get different $CB(\theta(R_1))$ where $\theta(R_1)$ represent the orientation of R_1 wrt a global reference frame



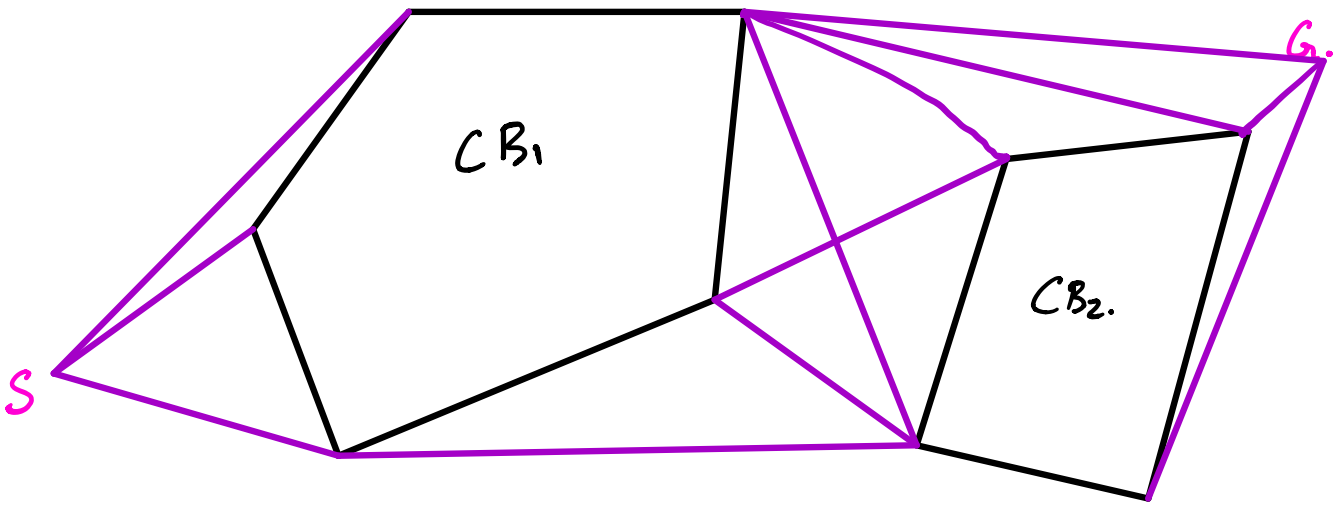
So what is the dimension of CB for a circular R_1. ?
polygonal (convex) R_1 ?

- The dimension of CB is the same as the number of degrees of freedom of R.

→ So what is the problem / are the problems due to the CS approach?

→ How do you make use of the CBs to compute a trajectory for a point robot?

The VISIBILITY GRAPH:



Connect all vertices of the CB's that are visible to one another.

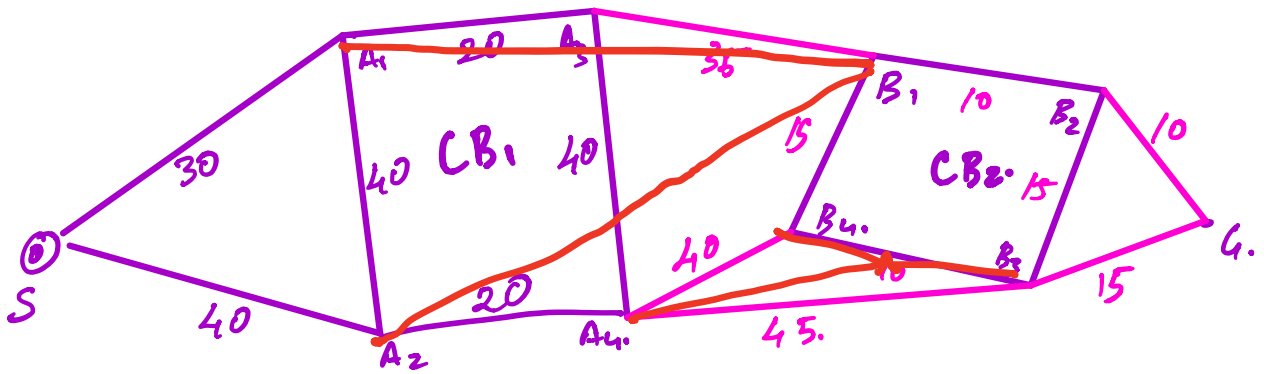
Connect the S & G vertices to the vertices of CB that are visible from S & G

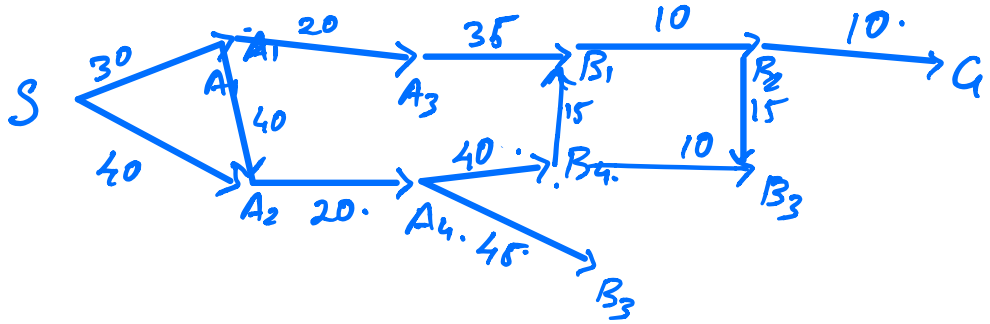
→ The resulting data structure is **Visibility Graph.**

→ Let the weight of the edges be the Euclidean distance between the two visible nodes (visible to one another).

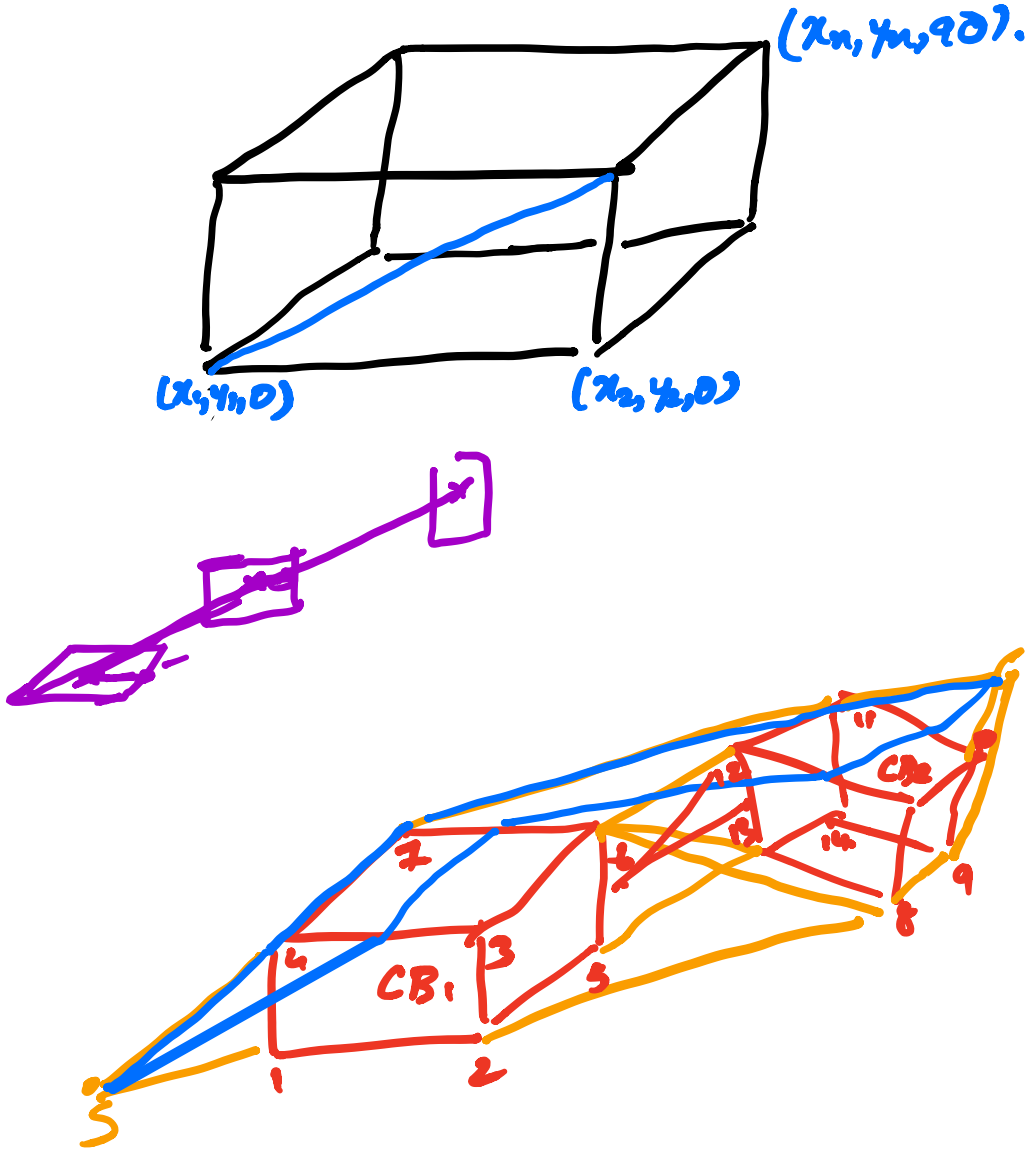
→ The shortest path lies on this graph.

→ Can be obtained by Djkstra's algorithm.



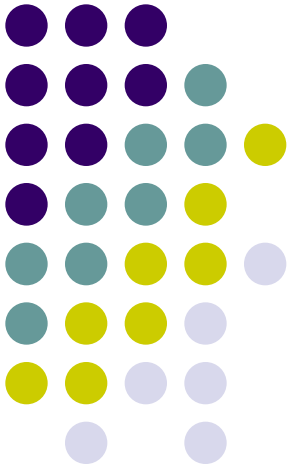


$S \rightarrow A_1 \rightarrow A_3 \rightarrow B_1 \rightarrow B_2 \rightarrow G.$

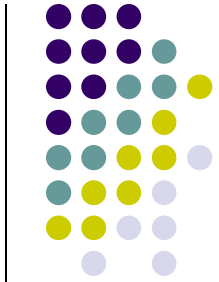

(x1, y1, 0)
(x2, y2, 0)
(xn, yn, 90).
CB1
CBn
1
2
3
4
5
6
7
8
9
10
11
12
13
14

Motion Planning by Configuration Space and Visibility Graphs

Lecture 9

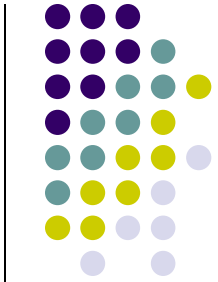


The Robot Motion Planning Problem



Most General Form: Given an a-priori known workspace with stationary objects of arbitrary shape whose representations are known and moving objects whose future trajectories are known find a collision free path that is optimal in some metric for a robot with arbitrary degrees of freedom between any two given points if one exists. (Generally unsolvable if optimality is required!)

Solvable form with optimality: Stationary objects of regular shapes (convex/concave polygonal), moving objects whose trajectories are linear and velocities uniform and a robot with limited degrees of freedom (2 or 3 in number)



COMPLEXITY

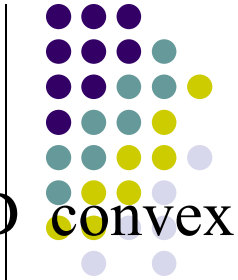
• Complexity of motion planning problem in presence of moving objects is inherently harder than the stationary case

• For stationary objects and a robot with two degrees of freedom the motion planning problem can be solved in polynomial time generally

• The problem of finding a path for a point robot in a plane with ***bounded*** velocity in presence of moving objects that are convex polygons moving with linear velocities ***without*** rotation the problem is classified as **NP Hard**

• For stationary objects and robots with arbitrary degrees of freedom the notion of probabilistic completeness is used for ascertaining complexity

Configuration Space Approach



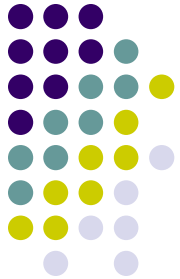
• This method of 2-D motion planning assumes a set of 2-D convex polygonal obstacles and a 2-D convex polygonal mobile robot.

• The general idea is grow the obstacles by the size of the mobile robot, thereby reducing the analysis of the robot's motion from a moving area to a single moving point.

• The point will always be a safe distance away form each obstacle due the growing step of each obstacle. Once we shrink the robot to a point, we can then find a safe path for the robot using a graph search technique.

• Reference: *An Algorithm for Planning Collision Free Paths Among Polyhedral Obstacles* by T. Lozano-Perez and M. Wesley. *(Communications of the ACM 22: 560-570)*

Configuration Space Approach



- Please note the following:
  - The planning is done for a point on the robot. The CS approach makes sure that if a path is collision free for this point it is collision free for the whole robot
  - The number of degrees of freedom of the robot is the dimension of the CS
  - The configuration obstacle (usually denoted as CB) is constructed by tracing the locus of the reference point of the robot for those configurations where the robot just graces the obstacle (Boundary between collision and no collision) or the limiting case

The Overall Algorithm with Visibility Graph

• Grow each obstacle in the scene by the size of the mobile robot. This is done by finding a set of vertices that determine the grown obstacle.

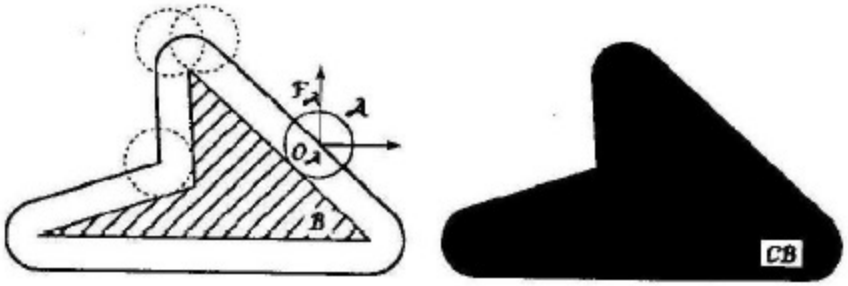
•We can now create a visibility graph. A visibility graph is an undirected graph G = (V,E) where the V is the set of vertices of the grown obstacles plus the start and goal points, and E is a set of edges consisting of all polygonal obstacle boundary edges, or an edge between any 2 vertices in V that lies entirely in free space except for its endpoints. Intuitively, if you place yourself at a vertex, you create an edge to any other vertex you can see (i.e. is visible).

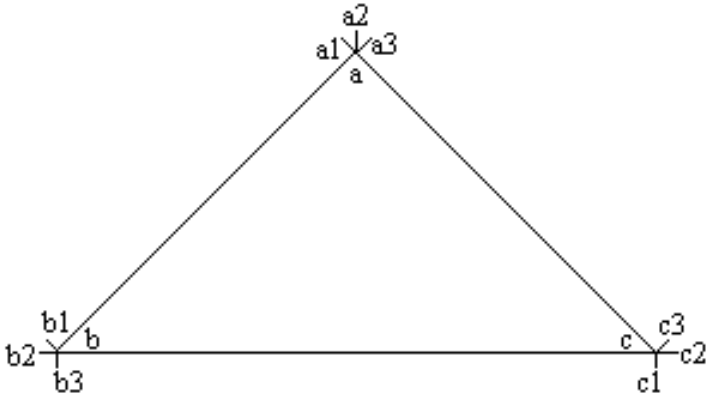
•

• The shortest path in distance can be found by searching the graph G using the shortest path search (Dijkstra's Algorithm) or other heuristic search method

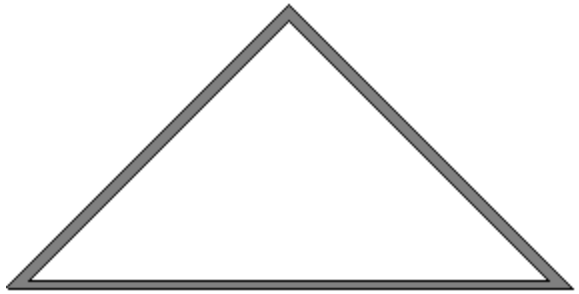
Computing the Configuration Space

- For the simple case of a circular robot





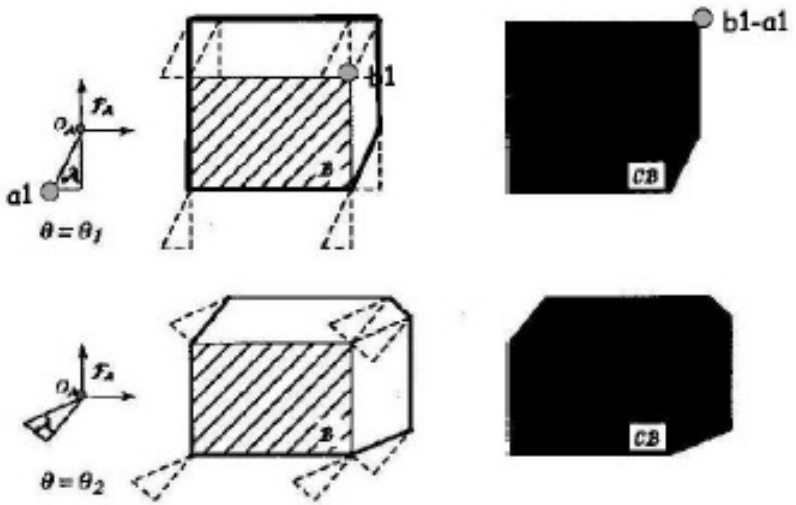
Points for the convex hull

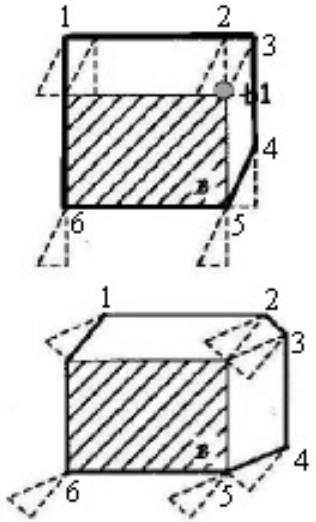


Simplification that you would use for your algorithms

Computing the Configuration Space (contd.)

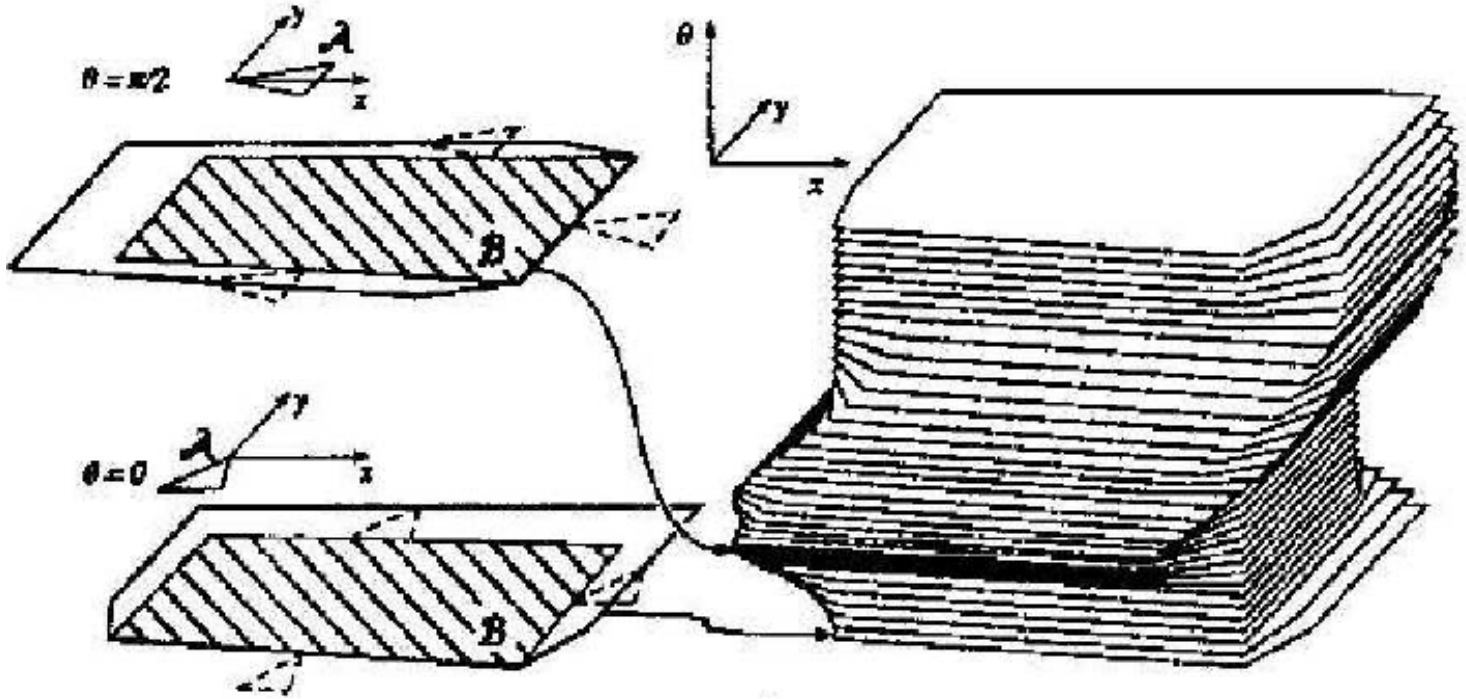
- For a convex polygonal robot



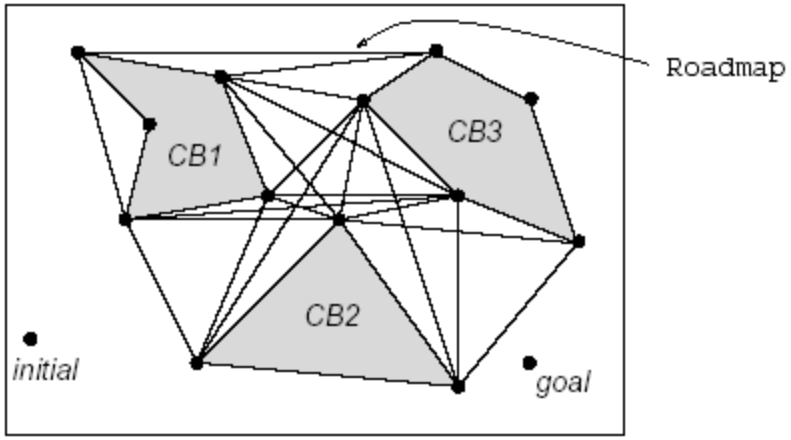


Enumerating the vertices

Problems with CS approach

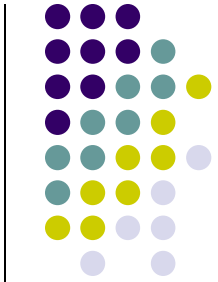


Constructing the Visibility Graph



The visibility graph is an undirected graph G = (V,E) where the V is the set of vertices of the grown obstacles plus the start and goal points, and E is a set of edges consisting of all polygonal obstacle boundary edges, or an edge between any 2 vertices in V that lies entirely in free space except for its endpoints. Intuitively, if you place yourself at a vertex, you create an edge to any other vertex you can see (i.e. is visible).

An Algorithm for Visibility Graph



Brute Force Method:

A simple algorithm to compute G is the following. Assume all N vertices of the G are connected. This forms N(N-1)/2 edges. Now check each edge to see if it intersects (excepting its endpoints) any other edge in the graph. If so reject this edge. The remaining edges are the edges of the visibility graph. This algorithm is brute force and slow but simple to compute. Faster algorithms are known

Fact:

If the initial and final locations of the robot are vertices on the G, then shortest path is contained on the visibility graph. In other words if a computation between S and T involves the visibility graph then it needs to be the shortest path as long as S and T are connected to the vertices of G by straight line segments

Robotics: Planning and Navigation - Motion planning for UAV

Harikumar Kandath

Robotic Research Center,
IIIT Hyderabad

harikumar.k@iiit.ac.in

Jan 29, 2021.

I. Rotary wing UAV

Figure: Multi-rotor VTOL

II. Fixed wing UAV

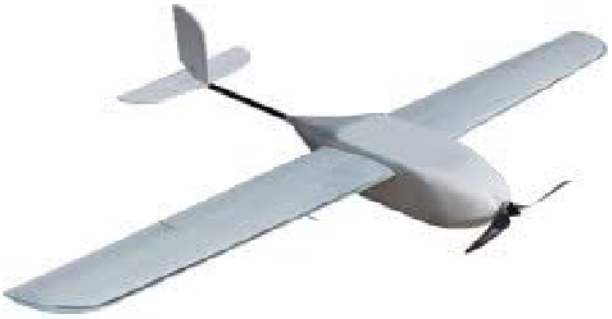


Figure: Fixed wing UAV

Key elements in Unmanned operation

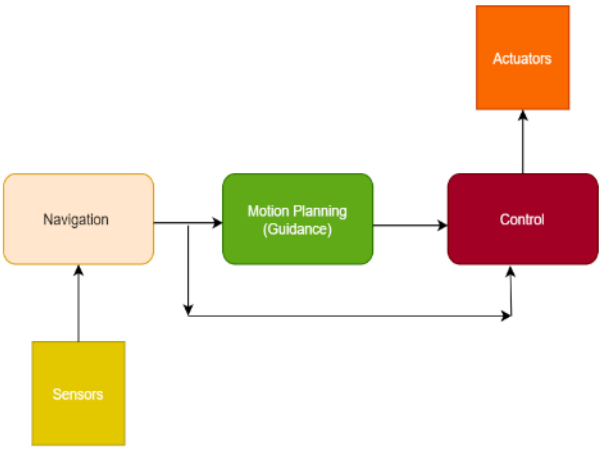


Figure: Navigation, Guidance and Control

Inertial Frame

Position (x, y, z), Velocity (v_x, v_y, v_z), Acceleration (a_x, a_y, a_z).

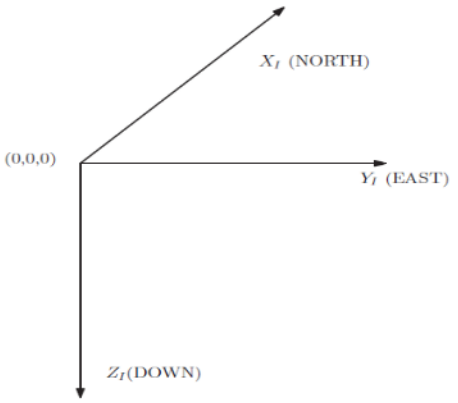


Fig. 1. Diagram showing inertial $X_I Y_I Z_I$ co-ordinate sytem

Velocity Control

Motion in 2D plane $(X_I - Y_I)$.
Altitude $(h = -z)$ is constant, i.e. $\frac{dh}{dt} = \dot{h} = 0$.
Current position (x, y) and the desired position is (x_d, y_d).

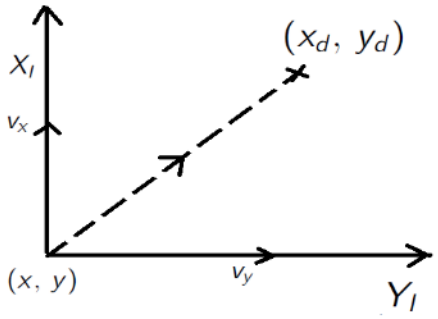
$$\frac{dx}{dt} = \dot{x} = v_x \tag{1}$$

$$\frac{dy}{dt} = \dot{y} = v_y \tag{2}$$

Input is (v_x, v_y) and output is (x, y).
NB: The desired (v_x, v_y) is tracked by inner loop control system.

Velocity Control - Reaching a point

$$v_x = k(x_d - x) \quad (3)$$
$$v_y = k(y_d - y) \quad (4)$$



Acceleration Control

$$\frac{dx}{dt} = \dot{x} = v_x \quad (5)$$
$$\frac{d^2x}{dt^2} = \ddot{x} = a_x \quad (6)$$

$$\frac{dy}{dt} = \dot{y} = v_y \quad (7)$$
$$\frac{d^2y}{dt^2} = \ddot{y} = a_y \quad (8)$$

Reaching a point $(x_d,\ y_d)$.

$$a_x = k_1(x_d - x) - k_2 v_x,\ a_y = k_1(y_d - y) - k_2 v_y \quad (9)$$

Acceleration Control

Trajectory $(x(t), y(t))$

$$\ddot{x} + k_2\dot{x} + k_1 x = k_1 x_d,\ \ddot{y} + k_2\dot{y} + k_1 y = k_1 y_d \qquad (10)$$

At steady-state, $\ddot{x} = \dot{x} = 0 \implies x = x_d$ and $\ddot{y} = \dot{y} = 0 \implies y = y_d$.

Velocity tracking through acceleration control

Desired velocity $(v_{xd},\ v_{yd})$

$$\frac{d^2x}{dt^2} = \ddot{x} = \frac{dv_x}{dt} = a_x \qquad (11)$$

$$a_x = k_3(v_{xd} - v_x) \qquad (12)$$

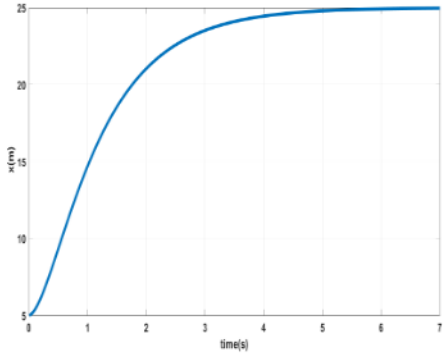
$$\frac{dv_x}{dt} = k_3(v_{xd} - v_x) \qquad (13)$$

$$\frac{dv_x}{dt} + k_3 v_x = k_3 v_{xd} \qquad (14)$$

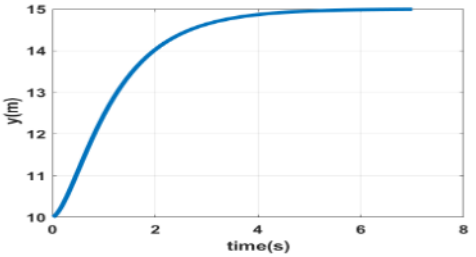
At steady-state $dv_x/dt = 0 \implies v_x = v_{xd}$

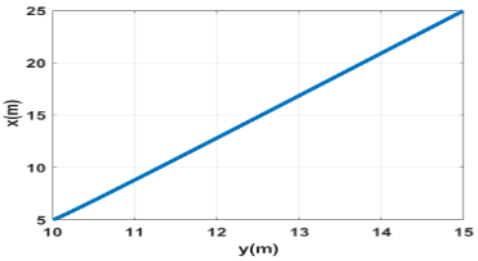
Example

$(x,\ y) = (5,\ 10),\ (v_x,\ v_y) = (1,\ 0.5),$
$(x_d,\ y_d) = (25,\ 15),\ k_1 = 3,\ k_2 = 4.$



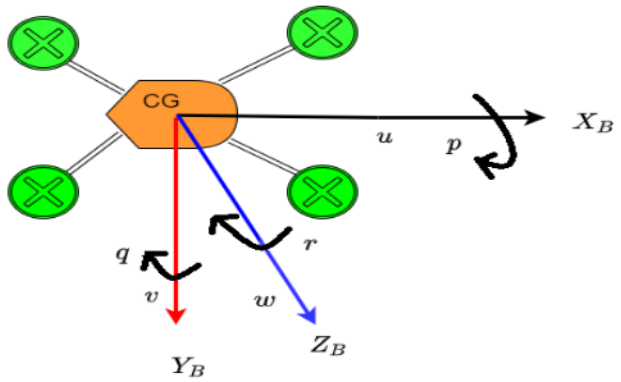
Example





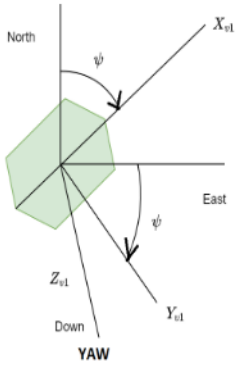
Variables- Body Frame

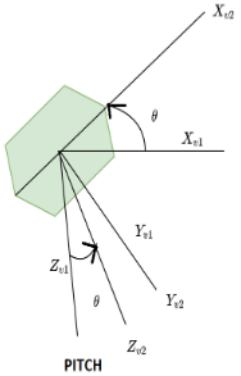
- Velocity: $V_B = [u, v, w]^T$
- Acceleration: $\dot{u} = \frac{du}{dt}$, $\dot{v} = \frac{dv}{dt}$, $\dot{w} = \frac{dw}{dt}$
- Angular velocity: $\Omega = [p, q, r]^T$

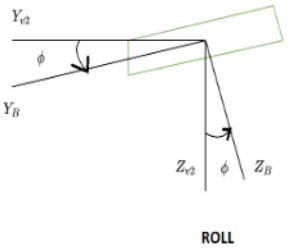


Euler angles - ϕ, θ, ψ

First rotation (i to v_1)- $\dot{\phi} = 0, \dot{\theta} = 0, \dot{\psi} \neq 0$
Second rotation (v_1 to v_2) - $\dot{\phi} = 0, \dot{\theta} \neq 0, \dot{\psi} = 0$
Third rotation (v_2 to B) - $\dot{\phi} \neq 0, \dot{\theta} = 0, \dot{\psi} = 0$







Euler angles and p,q,r

$$\begin{pmatrix} p \\ q \\ r \end{pmatrix} = \begin{pmatrix} \dot{\phi} \\ 0 \\ 0 \end{pmatrix} + R(\phi)^{b}_{v_2} \begin{pmatrix} 0 \\ \dot{\theta} \\ 0 \end{pmatrix} + R(\phi)^{b}_{v_2} R(\theta)^{v_2}_{v_1} \begin{pmatrix} 0 \\ 0 \\ \dot{\psi} \end{pmatrix} \quad (15)$$

$$\begin{pmatrix} p \\ q \\ r \end{pmatrix} = \begin{pmatrix} 1 & 0 & -sin(\theta) \\ 0 & cos(\phi) & sin(\phi)cos(\theta) \\ 0 & -sin(\phi) & cos(\phi)cos(\theta) \end{pmatrix} \begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} \quad (16)$$

Euler angles and p,q,r

Remark: Rotating about only one of the body axis will directly relate $\dot{\phi} = p\,(q = r = 0)$; $\dot{\theta} = q\,(\phi = 0, p = r = 0)$ and $\dot{\psi} = r\,(\phi = \theta = 0,\ p = q = 0)$.

$$\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} 1 & sin(\phi)tan(\theta) & cos(\phi)tan(\theta) \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi)sec(\theta) & cos(\phi)sec(\theta) \end{pmatrix} \begin{pmatrix} p \\ q \\ r \end{pmatrix} \quad (17)$$

why this is important?
Ans. We cannot measure ϕ, θ, ψ directly using a sensor. But we can always measure p, q, r using 3-axis gyroscope mounted on the UAV and integrate the above to get ϕ, θ, ψ.

Connecting inertial and body axis velocities

How do we use the knowledge of ϕ, θ, ψ?

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = R(\phi)^{b}_{v_2} R(\theta)^{v_2}_{v_1} R(\psi)^{v_1}_{i} \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \tag{18}$$

$$\begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} = [R(\phi)^{b}_{v_2} R(\theta)^{v_2}_{v_1} R(\psi)^{v_1}_{i}]^{-1} \begin{pmatrix} u \\ v \\ w \end{pmatrix} \tag{19}$$

How do we get u, v, w ?

Equation of Coriolis

$V_B = [u,\ v,\ w]^T$, $\Omega = [p,\ q,\ r]^T$, $F = [F_{XB},\ F_{YB},\ F_{ZB}]^T$.

- Equation of Coriolis: Rate of change of a vector (that is defined in a rotating frame)in inertial frame = Rate of change of the vector in the rotating frame + change due to relative angular velocity between the inertial frame and the rotating frame.

$$(\frac{dV_B}{dt})_I = (\frac{dV_B}{dt})_B + \Omega \times V_B = \frac{F}{m} \tag{20}$$

Translational Dynamics

$$V_B = [u,\ v,\ w]^T,\ \Omega = [p,\ q,\ r]^T,\ F = [F_{XB},\ F_{YB},\ F_{ZB}]^T.$$

$$(\frac{dV_B}{dt})_I = (\frac{dV_B}{dt})_B + \Omega \times V_B = \frac{F}{m} \tag{21}$$

$$\dot{u} = rv - qw + \frac{F_{XB}}{m} \tag{22}$$

$$\dot{v} = pw - ru + \frac{F_{YB}}{m} \tag{23}$$

$$\dot{w} = qu - pv + \frac{F_{ZB}}{m} \tag{24}$$

Time-Scale Separation

Basic Idea: Faster variable changes and goes to steady value when compared to the slower variable.

- p,q,r
- ϕ, θ, ψ
- $\dot{u}$, $\dot{v}$ and $\dot{w}$ (a_x, a_y, a_z)
- u, v, w and (v_x, v_y, v_z)
- x,y,z

Simplification

$$\begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} = R(\phi, \theta, \psi) \begin{pmatrix} u \\ v \\ w \end{pmatrix} \tag{25}$$

$$\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} = R(\phi, \theta, \psi) \begin{pmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{pmatrix} + \dot{R}(\phi, \theta, \psi) \begin{pmatrix} u \\ v \\ w \end{pmatrix} \tag{26}$$

$\dot{R}(\phi, \theta, \psi) \approx 0$

Implementation of acceleration command

Compute the desired angles θ_d, ϕ_d and the thrust T_d for a given ψ using the below equations. Alternatively, we can also compute desired θ_d, ψ_d, T_d for a given ϕ (or) compute desired ϕ_d, ψ_d, T_d for a given θ.

$$a_x = (-cos\phi sin\theta cos\psi - sin\phi sin\psi)\frac{T}{m} \tag{27}$$

$$a_y = (-cos\phi sin\theta sin\psi - sin\phi cos\psi)\frac{T}{m} \tag{28}$$

$$a_z = g - (cos\phi cos\theta)\frac{T}{m} \tag{29}$$

NB: The angles θ, ϕ, ψ and the thrust T can be controlled precisely.

THANK YOU

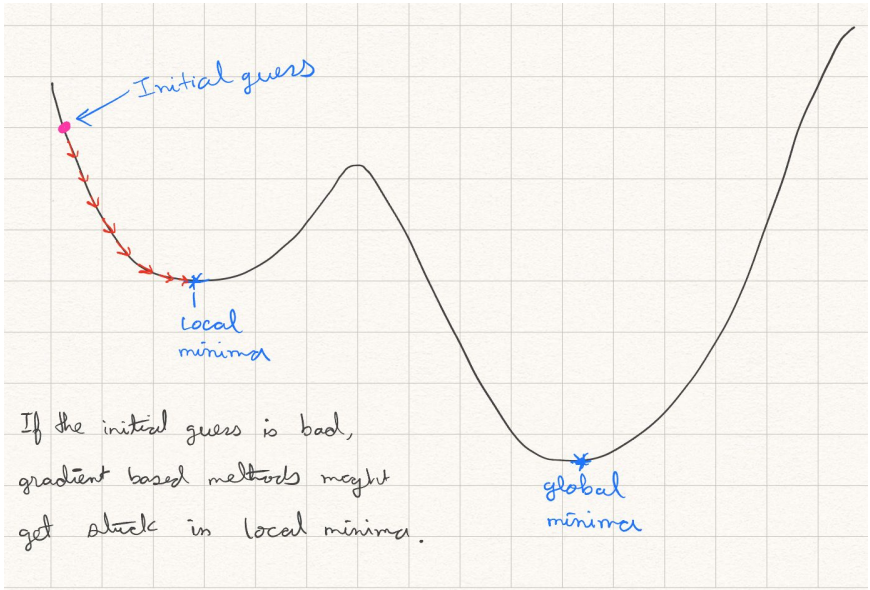
Cross Entropy Method

The motion planning problem in a mobile robot has two main constraints:

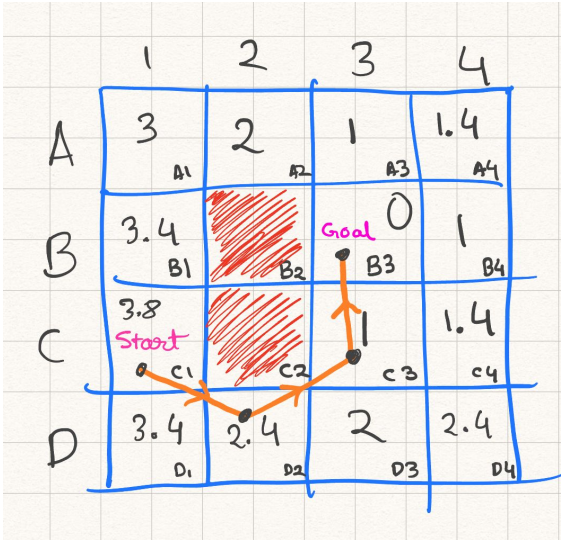
1. Kinematics/Dynamics constraints
2. Obstacle constraints

The goal is to generate a trajectory to reach a goal optimally.

- Gradient based optimization approaches like stochastic gradient descent are not suitable as they need a good starting guess and the obstacle avoidance constraints may be non-smooth. Without a good starting guess the algorithm might get stuck in a local minima.



- An alternative approach is to discretize the vehicle state space and generate a path by transitioning between adjacent cells. However if the state space is very large, this approach becomes computationally expensive. Also we can’t add kinematic constraints in these kinds of approaches.



- To overcome these difficulties, we use sampling based motion planning.
- Here we construct a graph with nodes corresponding to the vehicle states and the edges satisfying the dynamics and constraints.
- Two main approaches here are:
    - **Rapidly Exploring Random Trees:** Quickly explores the state space and gives a solution which may not be optimal.
    - **Probabilistic Road Maps:** Approaches the optimal trajectory but at an exceptionally slow rate.
- **How do we achieve the best of both worlds?**
- For this we use stochastic optimization based methods like **Cross Entropy Method (CEM)** or **Model Predictive Path Integral (MPPI).**

Cross Entropy Method (CEM) based motion planning algorithm

1. Sample controls from a gaussian distribution.

Let's say we are planning 100 (N=100) trajectories for 10 timesteps into the future (H = 10) and our action space is of 2-dimension (a_dim=2) consisting of velocity and angular velocity.

```
controls = torch.randn(H, N, a_dim)
# Here, H = horizon
# N = number of samples
# a_dim = dimension of the action space
```

Cross Entropy Method (CEM) based motion planning algorithm

1. Sample controls from a gaussian distribution.

The sampled controls are from a normal distribution with 0 mean and 1 standard deviation.

```
controls = a_mu + a_std * controls
```

Here, a_mu is a vector of dimension [10, 1, 2] and is the desired mean.

a_std is a vector of dimension [10, 1, 2] and is the desired standard deviation.

Cross Entropy Method (CEM) based motion planning algorithm

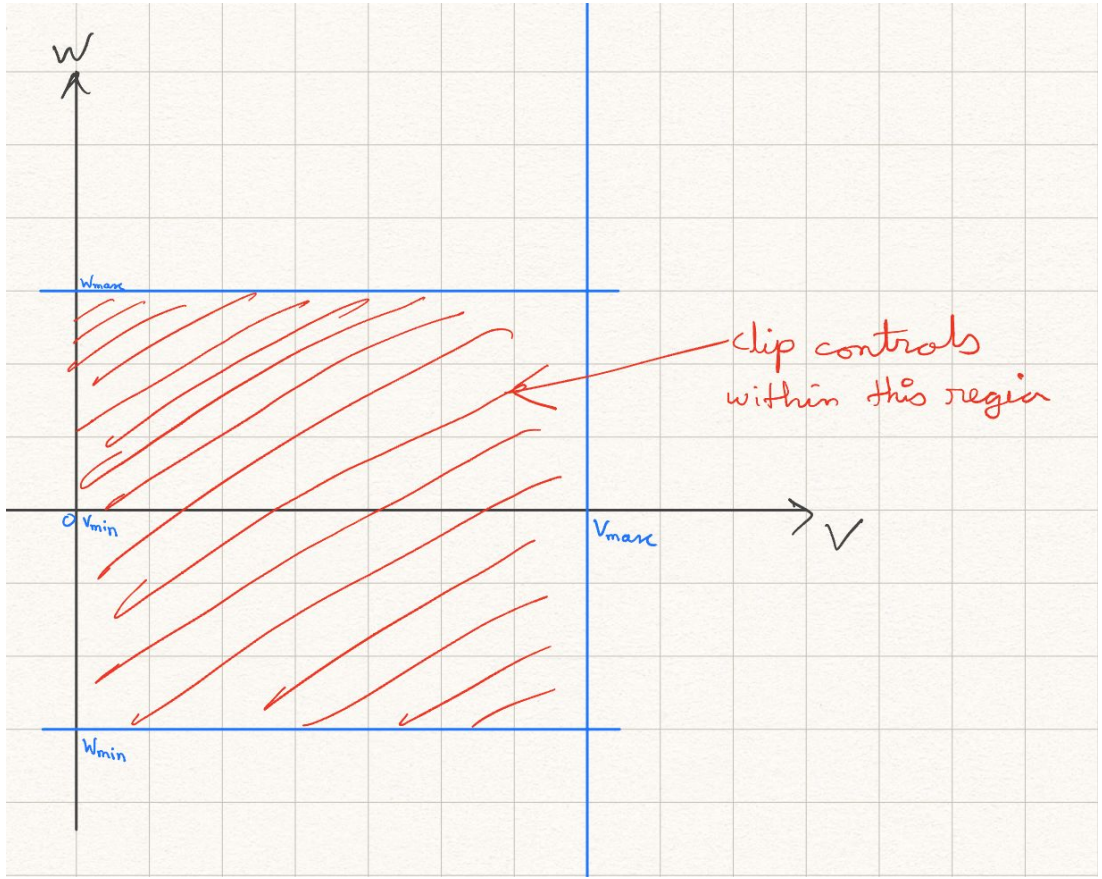
2. **Clip the controls within the control bounds of the agent.**

The sampled controls may have values that are beyond the maximum and minimum velocities and angular velocities of the ego-vehicle.

u_min = torch.tensor([v_min, w_min)

u_max = torch.tensor([v_max, w_max])

clipped_controls = torch.min(torch.max(controls,u_min),u_max)

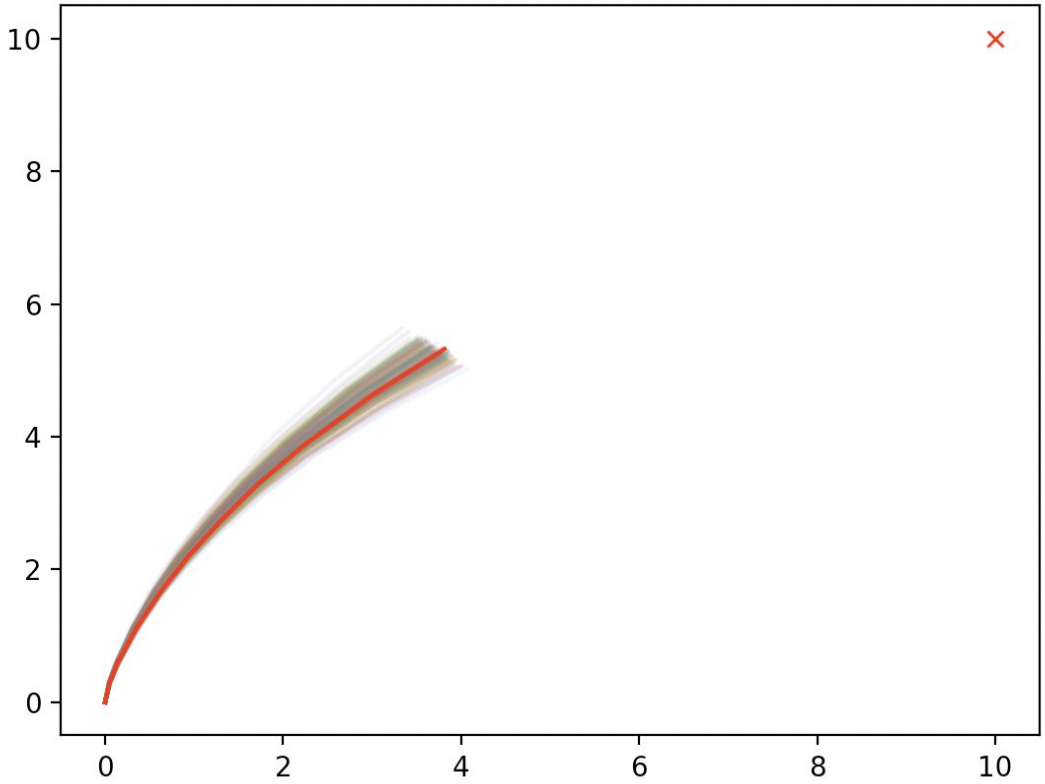

w
v
w_{max}
w_{min}
0
v_{min}
v_{max}
clip controls
within this region

Rollout the trajectories

Using a simplified kinematics model of our vehicle we now pass the sampled controls through our kinematics model to get our trajectories.

- theta_t = theta_{t-1} + w*dt
- x_t = x_{t-1} + v*cos(theta_t)*dt
- y_t = y_{t-1} + v*sin(theta_t)*dt

We will have a tensor named `traj` of dimension [H,N,3] since each trajectory has x,y and theta.



- Planning horizon (H):10.
- Number of samples (N): 200
- Control dimension (a_dim): 2
- The resulting tensor is of shape [10,200,2] # [H, N, a_dim]
- After rolling out the controls through a unicycle kinematics model we get the following trajectories.

Cross Entropy Method (CEM) based motion planning algorithm

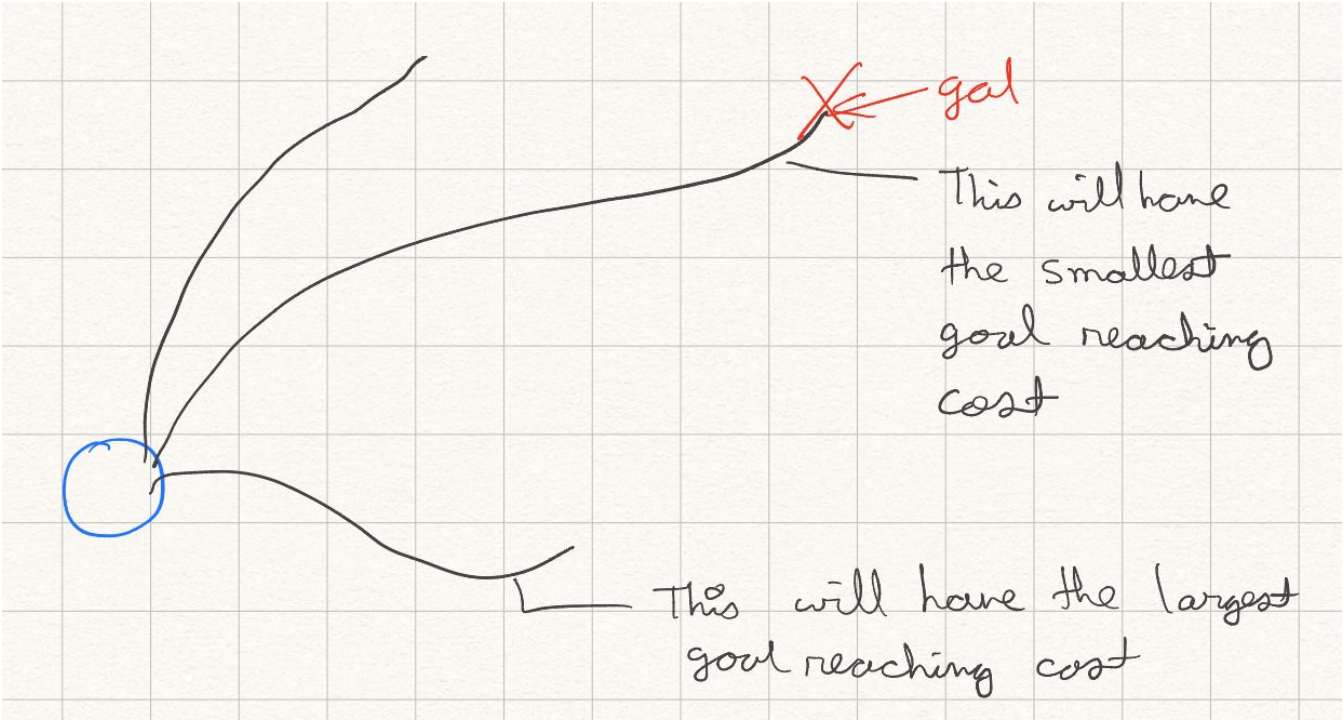
4. **Score the trajectories**

- Score our N trajectories using some cost functions.
- Examples of cost functions:
  1. Goal-reaching cost
  2. Smoothness cost
  3. Obstacle avoidance cost
- For each trajectory the total cost is: goal-reaching cost + smoothness cost + obstacle avoidance cost.
- Select the top k trajectories with the smallest cost. These are the elite trajectories.

4.1 Goal reaching cost

- The trajectory whose endpoint is closest to the goal position gets the smallest cost.

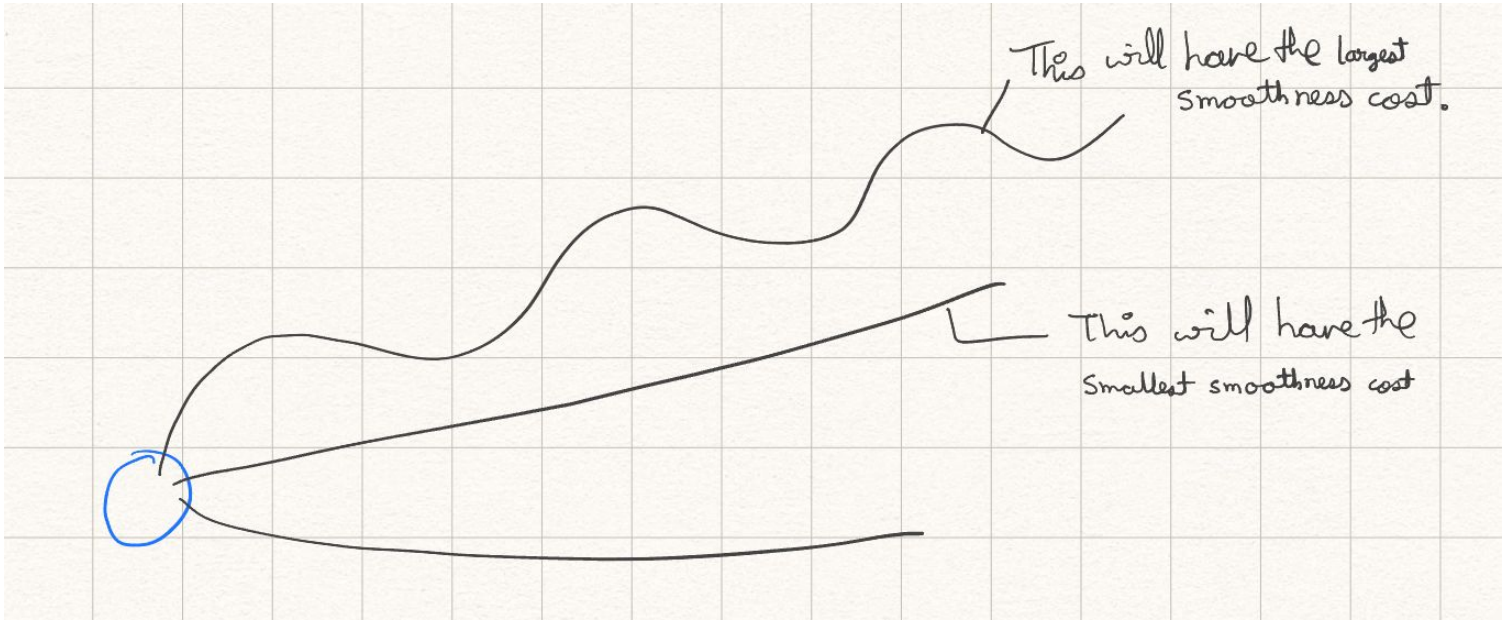
```
goal_state = [goal_x, goal_y, goal_theta]
for i in range(N):
    goal_cost[i] = sqrt(
        (traj[9, i, x]-goal_x)**2 + # Assuming horizon = 10
        (traj[9, i, y]-goal_y)**2 +
        (traj[9, i, theta]-goal_theta)**2
        )
'''
goal_cost is a vector of size N that contains the distance
from the end-point of each trajectory to the goal point
'''
```


goal
This will have the smallest goal reaching cost
This will have the largest goal reaching cost

4.2 Smoothness cost

- The trajectory where the change in the angular velocity and velocity is minimum is considered smoothest.

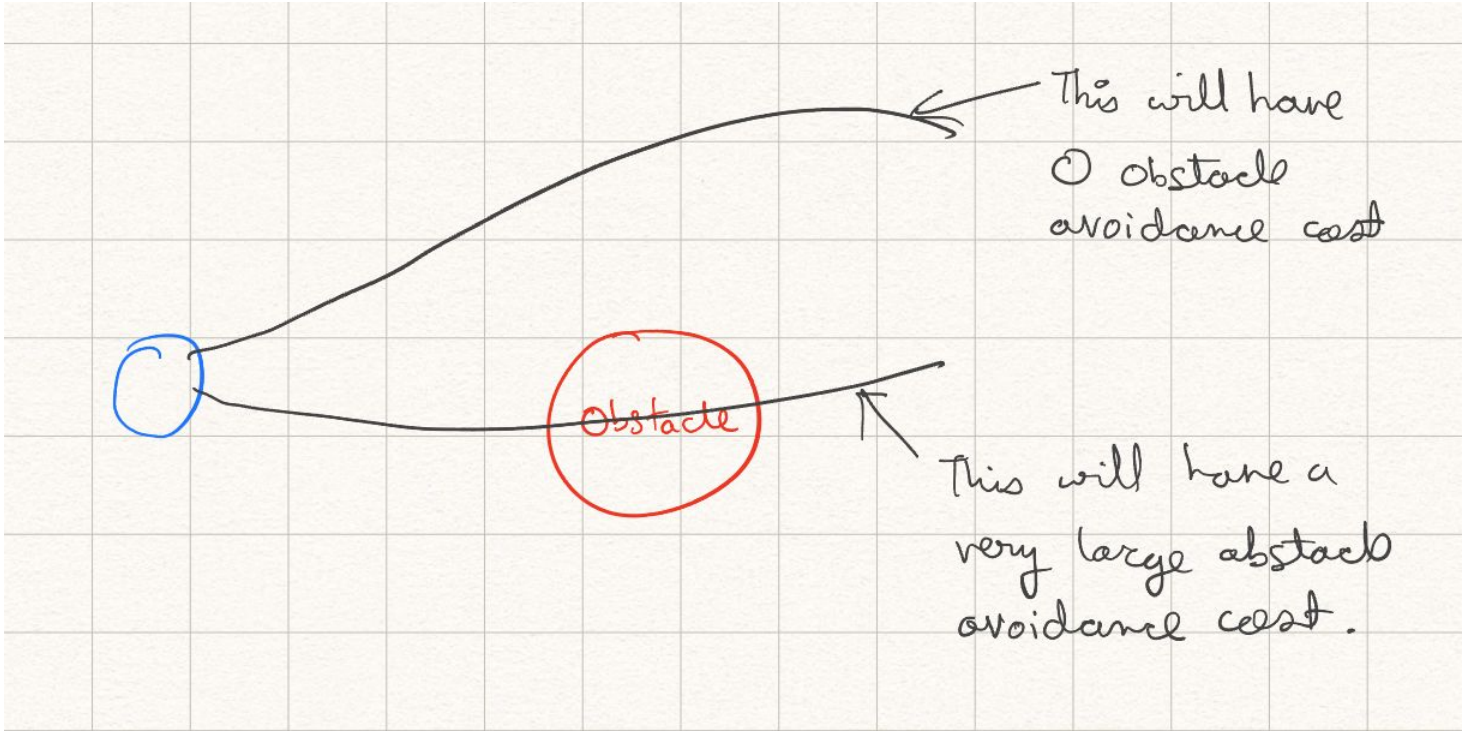
```
for i in range(N): # N = number os samples
    smooth_v[i] = torch.norm(controls[:,i,0])
    smooth_ang_v[i] = torch.norm(controls[:,i,1])
'''
smooth_v and smooth_ang_v are vectors of size N that contains the
smoothness cost for velocity and angular velocity respectively for each of the N trajectoreis.

'''
```


This will have the largest smoothness cost.
This will have the smallest smoothness cost

4.3 Obstacle avoidance cost

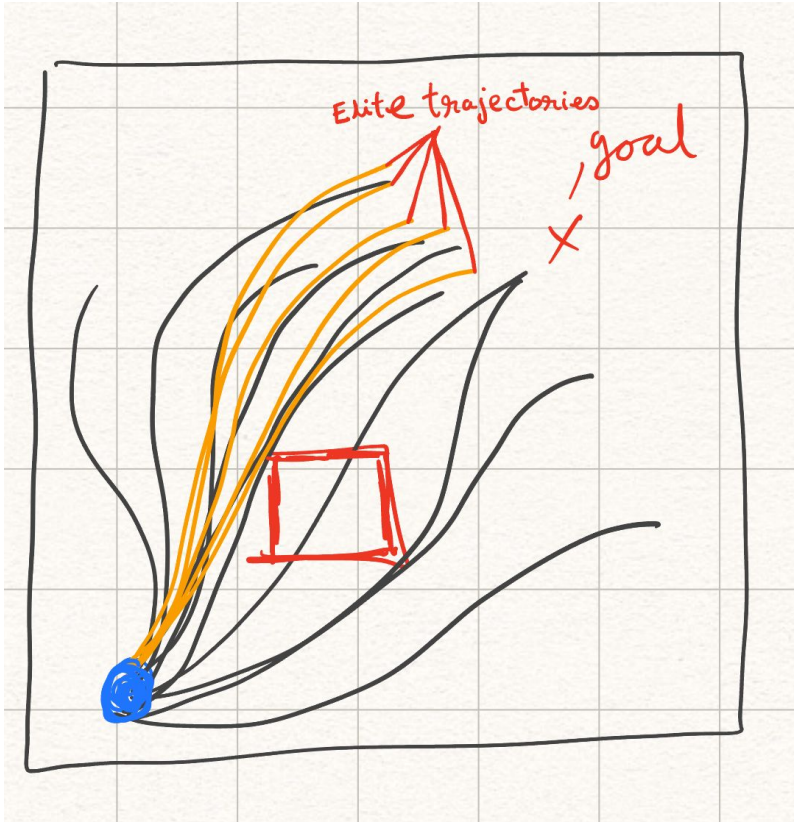
- If the x,y of the trajectory lies inside obstacle, assign a very high cost to the trajectory.

```
obs = [obs_x, obs_y] # vector containing the position of the obstacle
agent_radius = 1
obstacle_radius = 1

for i in range(N): # N = number os samples
    for j in range(H):
        d = sqrt((traj[j,i,0] - obs[0])**2 + (traj[j,i,1] - obs[1])**2)
        if(d<agent_radius+obstacle_radius):
            obs_cost[i] += 5000


'''
obs_cost is a vector of size N that contains the
obstacle  avoidance cost for each of the N trajectoreis.

'''
```


This will have
0 obstacle
avoidance cost
Obstacle
This will have a
very large obstacle
avoidance cost.

5. Select the elite trajectories

- For each trajectory the total cost is: goal-reaching cost + smoothness cost + obstacle avoidance cost.
- Select the top k trajectories with the smallest cost.
  - $k \ll N$. A good rule of thumb is to assign k as 10% of N.
  - This means that out of N trajectories we are only considering the top k trajectories which has the best cost.
  - These k trajectories are called elite trajectories.


Elite trajectories
goal

6. **Set the mean and covariance of the gaussian distribution as the mean and covariance of the elite trajectories**
    a. Find the mean and the std of the elite trajectories using torch.mean and torch.std.
    b. Set the mean and std of the sampling distribution as the mean and std of the elite trajectories.
7. **Goto step 1**
    a. Repeat the above steps until the K-L divergence between the previous distribution and the current distribution is less than a small constant or if the covariances have nearly shrunk to 0.

Initialization of the parameters

In step 1 of the CEM algorithm we sample from a gaussian distribution.

At the very start of the algorithm, we need to have a good initialization for the this gaussian distribution.

For this:

1. We compute the optimal trajectory that reaches the goal ignoring the obstacles and kinematics/dynamics constraints.
2. The initial parameters for the gaussian distribution is centered around this optimal trajectory.

Initialization of the parameters

1. We can use RRT to find the optimal trajectory to the goal.
2. Use the controls for this trajectory to obtain the mean and the covariance.

We can also generate trajectories using bernstein polynomials. The second derivative of the bernstein trajectories gives us the velocities which was can use as the initial guess.

`Note: This step needs to be done only once.`

Once we get a trajectory using the CEM method, we can use that as a guess value for next call to the CEM function for faster computation.

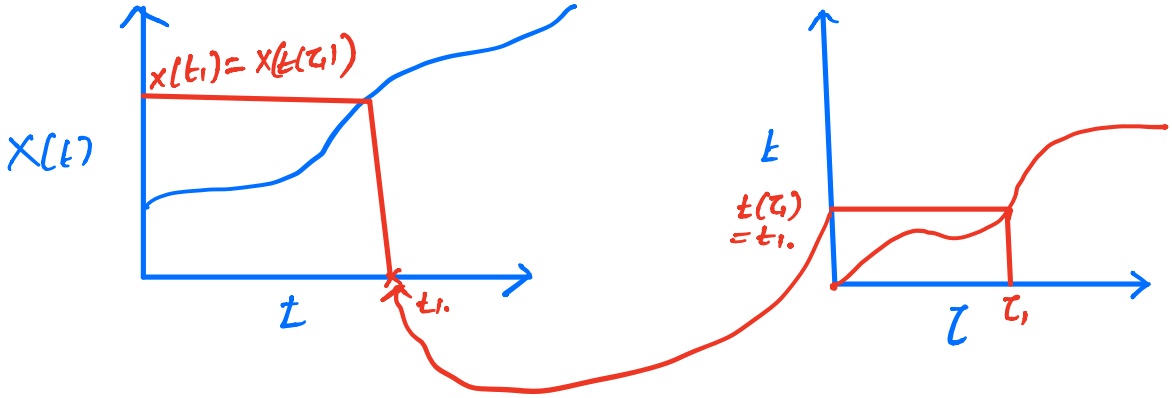
One issue with this approach is that if the environment has very narrow passages, the feasible region in the trajectory space will shrink. As a result a large number of the samples will be rejected and as a result the resultant trajectory may not be close to the optimal trajectory.

Thank You

Time Scaling:

Consider a time parametrized function implicitly defined as $X(t)$

Now what happens if t is some function of a new variable τ such that $t(\tau) = g$.



$$\dot{X}(t(\tau)) \leftarrow \frac{dx}{dt} \cdot \frac{dt}{d\tau} = X'(t)\frac{dt}{d\tau} = X'(t)\,s. \longrightarrow (1)$$

$s = \frac{dt}{d\tau}$ is the scaling function. $\longrightarrow (2)$.

$$X'(t) = \dot{X}(\tau)\,s' \longrightarrow (3).$$

$$\dot{X}(\tau) = \frac{dx(\tau)}{d\tau}, \quad s' = \frac{1}{s} \longrightarrow (4)$$

Write $X'(t)$ for $\frac{dx(t)}{dt}$ and $\dot{X}(\tau)$ for $\frac{dx(\tau)}{d\tau}$.

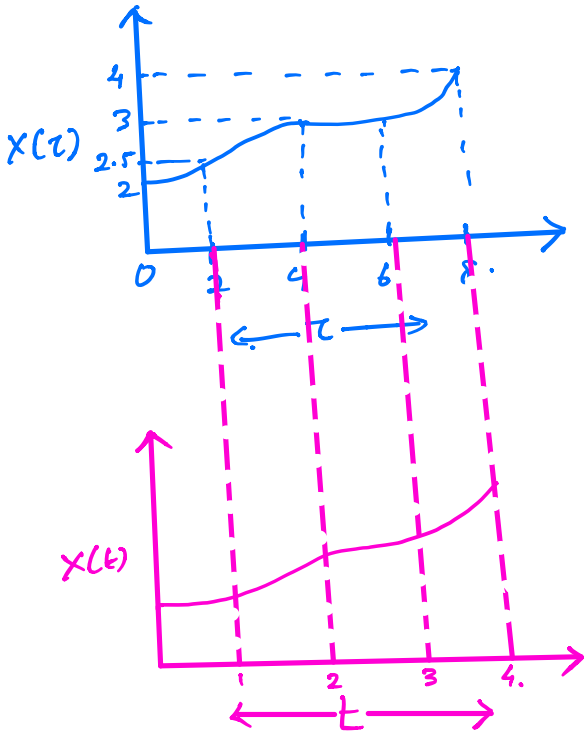
Let $t(\tau) = \frac{\tau}{2}$ $\longrightarrow$ (5).

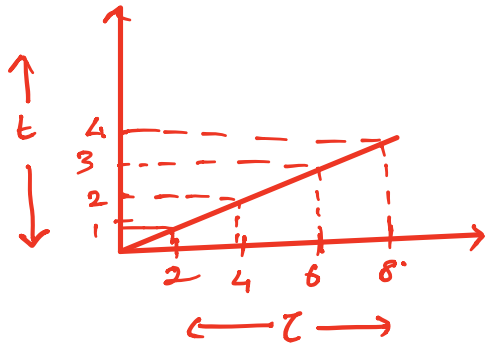
Then $\dot{X}(t(\tau)) = X'(t)(1/2) = \frac{1}{2} x'(t)$ $\longrightarrow$ (6).

$s = \frac{1}{2}$ is a constant scaling function

What does this mean?

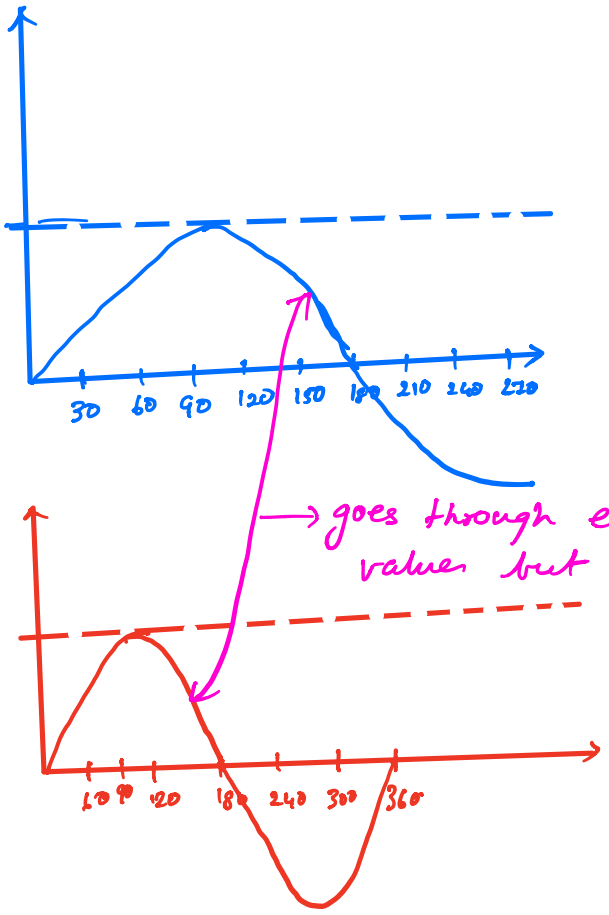
Let $X(\tau)$ be defined for time $\tau \in [0, 8]$





$X(t)$ passes precisely through the same set of points as $X(\tau)$ but at TWO times FASTER.

This has precisely the same connotation to the two functions $\sin(x)$ and $\sin(y)$, $y = 2x$



→ goes through exactly the same values but one travels faster than the other

$$\frac{d}{dx}\left(\sin(y(x))\right) = \cos y \cdot 2 = 2\cos y$$

↳ scaling factor

So how does one traverse the curve?

Let $\hat{x}(t) = 3t^2 + 4t + 5$ ⟶ (1).

and $t = 2\tau$. ⟶ (2)

Then $x(\tau) = 3(2\tau)^2 + 4(2\tau) + 5 = 12\tau^2 + 8\tau + 5$ ⟶ (2)

Let $\hat{x}(t)$ be defined in the interval $[2, 7]$.

Then $\tau \in [1, 7/2]$.

$\hat{x}'(t) = 6t + 4 \longrightarrow (3)$.

$\dot{x}(\tau) = 24\tau + 8 \longrightarrow (4)$.

$dt = 2\,d\tau \longrightarrow (5)$.

Let $dt = 0.1$ then $d\tau = 0.05$

$\hat{x}(t+dt) = \hat{x}(t) + \hat{x}'(t)\,dt \longrightarrow (6)$.

at $t = 2$, $\hat{x}'(t) = 6(2) + 4 = 16$. $\longrightarrow (7)$

$\hat{x}(2) = 3(2^2) + 4(2) + 5 = 25$. $\longrightarrow (8)$.

$\hat{x}(2.1) = \hat{x}(2) + \hat{x}'(2).(0.1)$

$= 25 + 16(0.1) = 25 + 1.6 = 26.6 \longrightarrow (9)$

$X(\tau)$ at $\tau = 1$, $= 12(1)^2 + 8(1) + 5 = 25 \longrightarrow (10)$

$x(\tau + d\tau) = x(\tau) + \dot{x}(\tau)\,d\tau$.

for $dt = 0.1$, $d\tau = 0.05$.

$\dot{x}(1) = 24(1) + 8 = 32$.

$x(1 + 0.05) = 25 + 32(0.05) = 25 + 1.6 = 26.6 \longrightarrow (11)$

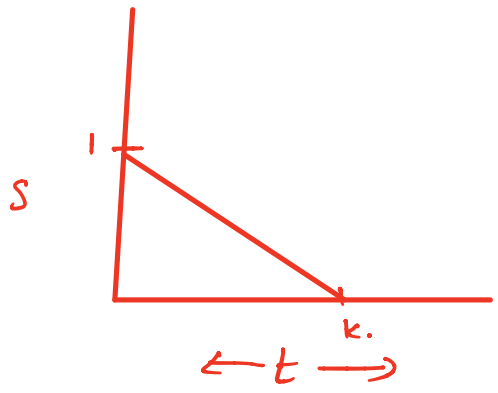
(9) & (11) are same

Linear Time Scaling:

$S(t) = \frac{dt}{d\tau} = \max\left(1 - \frac{\tau}{k}, 0\right)$. $\longrightarrow (12)$.

Scale reduces from 1 to 0 in k time steps.

as t 1's from 0 to k if the step size is 1.



Let $\hat{x}(t) = 3t^2 + 4t + 5$ → (1)

as before.

$$\frac{dt}{d\tau} = 1 - \frac{\tau}{k} \quad \rightarrow (2).$$

$$dt = \left(1 - \frac{\tau}{k}\right) d\tau. \quad \rightarrow (3).$$

$$t = \tau - \frac{\tau^2}{2k} \quad \rightarrow (4)$$

From (1) $X(\tau) = 3\left(t - \frac{\tau^2}{2k}\right)^2 + 4\left(\tau - \frac{\tau^2}{2k}\right) + 5$ → (5).

Let $\tau \in [0, 4]$. then $t \in \left[0, 4 - \frac{4^2}{2(4)}\right]$ $(k=4)$.

$t \in [0, 2]$. → (6).

$$\dot{X}(\tau) = 6\left(\tau - \frac{\tau^2}{2k}\right)\left(1 - \frac{\tau}{k}\right) + 4\left(1 - \frac{\tau}{k}\right)$$

$$= (1 - \tau/k)\left[6\left(\tau - \frac{\tau^2}{2k}\right) + 4\right] \quad \rightarrow (7)$$

Let $\tau_0 = 2$, $X(2) = 3\left(2 - \frac{4}{8}\right)^2 + 4\left(2 - \frac{4}{8}\right) + 5$

$X(2) = 3\left(\frac{3}{2}\right)^2 + 4\left(\frac{3}{2}\right) + 5 = 11 + \frac{27}{4} = 17.75$ → (8)

When $\tau = 2$, $t = 2 - \frac{2^2}{2(4)} = 2 - \frac{1}{2} = \frac{3}{2}$ → (9)

$\hat{X}(3/2) = 3(1.5)^2 + 4(1.5) + 5 = 6.75 + 6 + 5 = 17.75$ → (9)

$\dot{X}(2) = \left(1 - \frac{2}{4}\right)\left(6\left(2 - \frac{4}{8}\right) + 4\right)$ (from (7))

$= \frac{1}{2}\left(6 \cdot \frac{3}{2} + 4\right) = \frac{1}{2}(13) = 6.5$ → (10)

$\hat{X}'(1.5) = 6(1.5) + 4 = 13. \longrightarrow (11).$

Let $d\tau = 0.2$, then $dt = \left(1 - \frac{\tau}{k}\right) d\tau \quad (\tau = 2).$

$dt = \left(1 - \frac{2}{4}\right)(0.2) = \frac{1}{2}(0.2) = 0.1 \longrightarrow (12).$

Traversing the curve:

$X(2+0.2) = X(2) + \dot{x}(2)(0.2) = (11 + 6.75) + 6.5(0.2)$

$= 17.75 + 1.3 = 19.05 \longrightarrow (13)$

$\hat{x}(1.5 + 0.1) = \hat{x}(1.5) + \hat{x}'(1.5)(0.1)$

$= 17.75 + 13(0.1) = 17.75 + 1.3$

$= 19.05 \longrightarrow (14)$

$\therefore (13) = (14)$

17.75
+ 1.3
19.05

$q(t) = a_0 + a_1 t + a_2 t^2 + \dots + a_n t^n$

Conditions:

$q(0) = q_0$
$q(t_1) = q_1$
$q'(t_1) = q_1$
$q^2(t_1) = q_2$
$\vdots$
$q^n(t_n) = q_m$

Then $a_0 = q_0 \longrightarrow (1)$

$a_0 + a_1 t_f + \dots + a_n t_f^n = q_n \longrightarrow (2)$

$0 + a_1 + 2a_2 t_1 + 3a_3 t_1^2 + \dots + n a_n t_1^{n-1} = q_1 \longrightarrow (3)$

$0 + 0 + 2a_2 + 6a_3 t_1 + \dots + n(n-1) a_n t_1^{n-2} = q_2 \longrightarrow$

$\vdots$

$0 + 0 + 0 + 0 + \dots + n! a_n = q_m \longrightarrow$

Then

$$\begin{bmatrix} 1 & 0 & \dots & 0 \\ 1 & t_f & t_f^2 & \dots & t_f^n \\ 0 & 1 & 2t_1 & \dots & n t_1^{n-1} \\ 0 & 0 & 0 & 6t_1 & \dots & n(n-1) t_1^{n-2} \\ & & & & & \\ 0 & 0 & 0 & 0 & \dots & n! \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \\ \\ \\ a_n \end{bmatrix} = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ \\ \\ \end{bmatrix}$$

$$Ma = b$$

$$a = M^{-1} b$$

CSE 483: Mobile Robotics

Lecture by: Prof. K. Madhava Krishna
Scribe:Gourav Kumar, Enna Sachdeva

Module # 13
Date: 25th October, 2016 (Tuesday)

Non-holonomic Trajectory Planning (Bernstein Basis method)

This document discusses the theory of non holonomic trajectory planning using Bernestein polynomial along with the brief description of the associated topics.

$$q(t) = \sum_{i=0}^{n} a_i t_i^n$$

1 What is a Nonholonomic motion

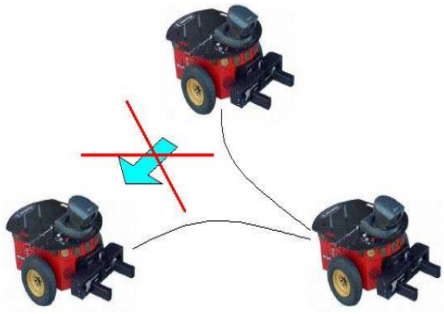


Figure 1:
Non holonomic characteristic of wheeled robots.

Non-holonomic systems are characterized by constraint equations which involves the time derivatives of the system configuration variables. In a configuration space $Q \subset R^n$, the configuration of a mechanical system can be uniquely described by an n-dimensional vector of generalized coordinates.
$q = (q_1, q_2, q_3, ..., q_n)^T$
The generalized velocity at a generic point of a trajectory $q(t) \subset Q$ is the tangent vector given by
$\dot{q} = (\dot{q_1}, \dot{q_2}, \dot{q_3}, ..., \dot{q_n})^T$

For a non holonomic systems, these equations are non integrable as they typically arise when the system has less controls than configuration variables(underactuated systems). As a result, a nonholonomic mechanical system cannot move in arbitrary directions in its configuration space. For instance, a unicycle has two controls (linear(v) and angular(w) velocities), while it moves in a 3-dimensional configuration space(x, y, θ). As a consequence, any path in the configuration space does not necessarily correspond to a feasible path for the system. In other words, for a non-holonomic systems, the line integrals depend not just on the start and end points but also the path taken.
The state transition matrix representation of a holonomic system is of the form

$$\dot{x} = f(x, u) \tag{1}$$

Since, equation 1 is non-integrable, we can approximate the integration using numerical integration methods, say Euler's method, which gives
$x_{new} \approx x + f(x, u)\Delta t$,
This shows that the new state x_{new} is constrained due to the choice of f.
Figure 1 shows one of the feasible paths (represented with lines) of a non-holonomic mobile robot to move between 2 states.

2 Differential Drive Robots

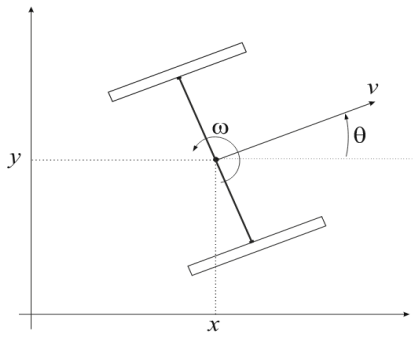


Figure 2:
Differential drive wheeled robot

Consider a differential drive nonholonomic mobile robot in a two-dimensional, free-space environment, as shown in figure 2. It is assumed that the robot cannot slip in lateral direction,
generalized coordinates : $q = (x, y, \theta)^T$
Nonholonomic constraints : $\dot{x}\sin\theta - \dot{y}\cos\theta = 0 \implies \dot{y} = \dot{x}\tan\theta$

$$y = \int \dot{x}\tan\theta dt \tag{2}$$

With 2 control inputs as (v, w), the kinematics model of the system is given by:-

$$\dot{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} \tag{3}$$

3 Motion Planning with Bernstein Polynomials

Since, equation 2 is non integrable, we can approximate the functions $\dot{x}$ and $\tan\theta$ with the bernestein polynomials and solve the integral.

NOTE: The Bernstein polynomials are advantageous over other approximation techniques like taylor series as the former and its derivatives polynomials uniformly approximates f and $\dot{f}$,respectively. It holds true for higher order derivatives as well. Moreover, they are the most numerically stable basis.

3.1 Bernstein basis polynomial

Let $f(x)$ be a real-valued function defined and bounded on the interval [0,1], then $B_n(f)$ is the polynomial on [0,1].

$$\mathbf{B_n(f(x);t) = \sum_{i=0}^{n} f(\frac{i}{n})^n C_i t^i (1-t)^{n-i}} \tag{4}$$

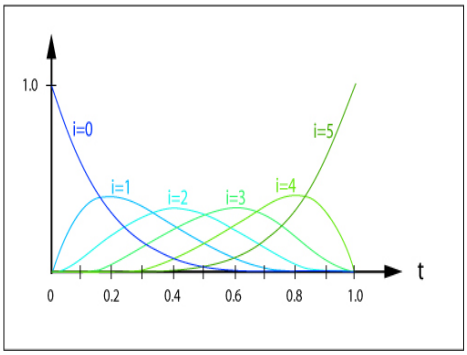


Figure 3:
Bernstein basis polynomials for n=5 .

If function f(x) is continuous on [0,1],then the Bernstein polynomial $B_n(f(x))$ tends uniformly to f as $n \to \infty$. Bernstein basis polynomials with n=5, are shown in figure 3.

For the given initial state $(x_{t_0}, y_{t_0}, \theta_{t_0})$ and final state $(x_{t_f}, y_{t_f}, \theta_{t_f})$,(more state constraints can be added to the system), with the start time t_0 and end time t_f of the trajectory of a nonholonomic system, $\dot{y(t)}$ and $\dot{x(t)}$ can be related as,
$\dot{y(t)} = \dot{x(t)} \tan\theta(t)$,
where, the functions $x(t)$ and $\tan\theta(t)$ can be approximated as a linear combination of Bernstein basis polynomials as following-

$$x(t) \approx B_n(x(t)) = B_x(\mu(t)) = \sum_{i=0}^{5} W_{x_i} B_i(\mu(t)), \tag{5}$$

Similarly,

$$\tan\theta(t) = k(t) \approx B_n(k(t)) = B_k(\mu(t)) = \sum_{i=0}^{5} W_{k_i} B_i(\mu(t)) \tag{6}$$

where,
$B_i(\mu(t)) =^n C_i(1-\mu)^i(\mu)^{n-i}$
$\mu(t) = \frac{t-t_0}{t_f-t_0}$

Differentiating equation 5 w.r.t time gives

$$\dot{x}(t) = \dot{B}_x(\mu(t)) = \sum_{i=0}^{5} W_{x_i} \dot{B}_i(\mu(t) \tag{7}$$

using above 2 equations, equation 2 can be rewritten as-

$$y(t) = y_0 + \int_{t_0}^{t} (\sum_{i=0}^{5} W_{x_i} \dot{B}_i(\mu(t))(\sum_{i=0}^{5} W_{k_i} B_i(\mu(t))dt \tag{8}$$

The bernstein coefficients of the polynomials and their derivatives for n=5, at time $t = t_0$ and $t = t_f$ are shown in the tables below.

Bernstein coefficients	$t = t_0$	$t = t_f$
$B_0(\mu) =^5 C_0(1-\mu)^5\mu^0$	1	0
$B_1(\mu) =^5 C_1(1-\mu)^4\mu$	0	0
$B_2(\mu) =^5 C_2(1-\mu)^3\mu^2$	0	0
$B_3(\mu) =^5 C_3(1-\mu)^2\mu^3$	0	0
$B_4(\mu) =^5 C_4(1-\mu)^1\mu^4$	0	0
$B_5(\mu) =^5 C_5(1-\mu)^0\mu^5$	0	1

Bernstein coefficients derivatives	$t = t_0, \mu = 0$	$t = t_f, \mu = 1$
$\dot{B}_0(\mu) =^5 C_0 \frac{-5(1-\mu)^4}{(t_f-t_0)}$	$\frac{-5}{(t_f-t_0)}$	0
$\dot{B}_1(\mu) =^5 C_1 \frac{-4\mu(1-\mu)^3+(1-\mu)^4}{(t_f-t_0)}$	$\frac{5}{(t_f-t_0)}$	0
$\dot{B}_2(\mu) =^5 C_2 \frac{-3\mu^2(1-\mu)^2+2(1-\mu)^3\mu}{(t_f-t_0)}$	0	0
$\dot{B}_3(\mu) =^5 C_3 \frac{-2\mu^3(1-\mu)+3(1-\mu)^2\mu^2}{(t_f-t_0)}$	0	0
$\dot{B}_4(\mu) =^5 C_4 \frac{-\mu^4+4(1-\mu)\mu^3}{(t_f-t_0)}$	0	$\frac{-5}{(t_f-t_0)}$
$\dot{B}_5(\mu) =^5 C_5 \frac{5\mu^4}{(t_f-t_0)}$	0	$\frac{5}{(t_f-t_0)}$

With the given state constraints(i.e. position, velocity, acceleration, etc.) of the robot at different instants, the unknown, time independent weight parameters $(W_{x_0}, W_{x_1}, W_{x_2}, ..., W_{x_5})$ and $(W_{k_0}, W_{k_1}, W_{k_2}, ..., W_{k_5})$ can be determined.

3.2 Finding $W_{x_0}, W_{x_1}, ..., W_{x_5}$

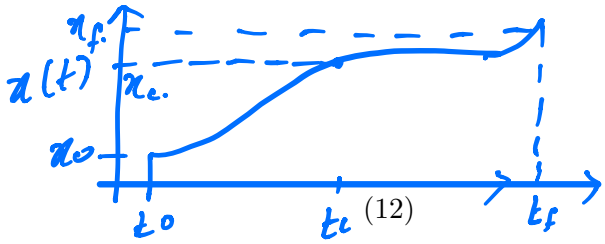
Given Constraints : $(x_{t_0}, y_{t_0}), (x_{t_c}, y_{t_c}), (x_{t_f}, y_{t_f}), (\dot{x_{t_0}}, \dot{y_{t_0}}), (\dot{x_{t_c}}, \dot{y_{t_c}}), (\dot{x_{t_f}}, \dot{y_{t_f}})$
Using equations 5, known parameters can be represented as a linear combination of bernstein basis polynomial as follows-

$$x(t_0) = W_{x_0}B_0(\mu(t_0))+W_{x_1}B_1(\mu(t_0))+W_{x_2}B_2(\mu(t_0))+W_{x_3}B_3(\mu(t_0))+W_{x_4}B_4(\mu(t_0))+W_{x_5}B_5(\mu(t_0)) \tag{9}$$

$$x(t_f) = W_{x_0}B_0(\mu(t_f))+W_{x_1}B_1(\mu(t_f))+W_{x_2}B_2(\mu(t_f))+W_{x_3}B_3(\mu(t_f))+W_{x_4}B_4(\mu(t_f))+W_{x_5}B_5(\mu(t_f)) \tag{10}$$

Putting values of bernstein polynomial coefficients (from tables) in the equations 9 and 10, gives

$$W_{x_0} = x(t_0) = x_{t_0} \tag{11}$$



$$W_{x_5} = x(t_f) = x_{t_f} \tag{12}$$

Using the remaining constraints, all weights $W_{x_1}, W_{x_2}, W_{x_3}, W_{x_4}$ can be evaluated.

$$\begin{bmatrix} x_{t_c} - W_{x_0}B_0(\mu(t_c)) - W_{x_5}B_5(\mu(t_c)) \\ \dot{x}_{t_0} - W_{x_0}\dot{B}_0(\mu(t_0)) - W_{x_5}\dot{B}_5(\mu(t_0)) \\ \dot{x}_{t_f} - W_{x_0}\dot{B}_0(\mu(t_f)) - W_{x_5}\dot{B}_5(\mu(t_f)) \\ \dot{x}_{t_c} - W_{x_0}\dot{B}_0(\mu(t_c)) - W_{x_5}\dot{B}_5(\mu(t_c)) \end{bmatrix} = \begin{bmatrix} B_1(\mu(t_c)) & B_2(\mu(t_c)) & B_3(\mu(t_c)) & B_4(\mu(t_c)) \\ \dot{B}_1(\mu(t_0)) & \dot{B}_2(\mu(t_0)) & \dot{B}_3(\mu(t_0)) & \dot{B}_4(\mu(t_0)) \\ \dot{B}_1(\mu(t_f)) & \dot{B}_2(\mu(t_f)) & \dot{B}_3(\mu(t_f)) & \dot{B}_4(\mu(t_f)) \\ \dot{B}_1(\mu(t_c)) & \dot{B}_2(\mu(t_c)) & \dot{B}_3(\mu(t_c)) & \dot{B}_4(\mu(t_c)) \end{bmatrix} \begin{bmatrix} W_{x_1} \\ W_{x_2} \\ W_{x_3} \\ W_{x_4} \end{bmatrix} \tag{13}$$

$$A_x = B_x W_x \tag{14}$$

$$W_x = pinv(B_x)A_x \tag{15}$$

3.3 Finding $W_{k_0}, W_{k_1}, ..., W_{k_5}$

Expanding equation 8,

$$y(t) = y_0 + \int_{t_0}^{t} (W_{k_0}.f_0(t, t_0, t_f, W_{x_1}, W_{x_2}, ...W_{x_5}) + (W_{k_1}.f_1(t, t_0, t_f, W_{x_1}, W_{x_2}, ...W_{x_5}) + ...+ (W_{k_5}.f_5(t, t_0, t_f, W_{x_1} \tag{16}$$

with the weight parameters $W_{x_1}....W_{x_5}$ calculated above, equation 16 further reduces to,

$$y(t) = y_0 + W_{k_0}F_0(t) + W_{k_1}F_1(t) + W_{k_2}F_2(t) + W_{k_3}F_3(t) + W_{k_4}F_4(t)0 + W_{k_5}F_5(t) \tag{17}$$

$$where F_i(t) = \int_{t_0}^{t} f_i(t, t_0, t_f, W_{x_1}, W_{x_2}, ...W_{x_5})dt \tag{18}$$

Our objective is to get weights, $W_{K_0}, W_{K_1}, W_{K_2}, W_{K_3}, W_{K_4}, W_{K_5}$

$$k(t_0) = k_0 = W_{k_0}B_0(\mu(t_0)) + W_{k_1}B_1(\mu(t_0)) + W_{k_2}B_2(\mu(t_0)) + W_{k_3}B_3(\mu(t_0)) + W_{k_4}B_4(\mu(t_0)) + W_{k_5}B_5(\mu(t_0)) \tag{19}$$

which gives,

$$W_{k_0} = k_0 \tag{20}$$

$$k(t_f) = k_f = W_{k_0}B_0(\mu(t_f)) + W_{k_1}B_1(\mu(t_f)) + W_{k_2}B_2(\mu(t_f)) + W_{k_3}B_3(\mu(t_f)) + W_{k_4}B_4(\mu(t_f)) + W_{k_5}B_5(\mu(t_f)) \tag{21}$$

which gives,

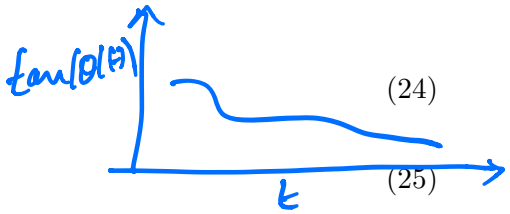
$$W_{k_5} = k_f \tag{22}$$

Using any 4 out of remaining constraints on y and k we can form a full rank matrix for B_k as:

$$\begin{bmatrix} \dot{k}_{t_0} - W_{k_0}\dot{B}_0(\mu(t_0)) - W_{k_5}\dot{B}_5(\mu(t_0)) \\ \dot{k}_{t_f} - W_{k_0}\dot{B}_0(\mu(t_f)) - W_{k_5}\dot{B}_5(\mu(t_f)) \\ y_0 - W_{k_0}F_0 - W_{k_5}F_5 \\ y_f - W_{k_5}F_0 - W_{k_5}F_5 \end{bmatrix} = \begin{bmatrix} \dot{B}_{k_1}(\mu(t_0)) & \dot{B}_2(\mu(t_0)) & \dot{B}_3(\mu(t_0)) & \dot{B}_4(\mu(t_0)) \\ \dot{B}_1(\mu(t_f)) & \dot{B}_2(\mu(t_f)) & \dot{B}_3(\mu(t_f)) & \dot{B}_4(\mu(t_f)) \\ F_1(t_0) & F_2(t_0) & F_3(t_0) & F_4(t_0) \\ F_1(t_f) & F_2(t_f) & F_3(t_f) & F_4(t_f) \end{bmatrix} \begin{bmatrix} W_{k_1} \\ W_{k_2} \\ W_{k_3} \\ W_{k_4} \end{bmatrix} \tag{23}$$

$$A_k = B_k W_k \tag{24}$$

$$W_k = pinv(B_k) A_k \tag{25}$$

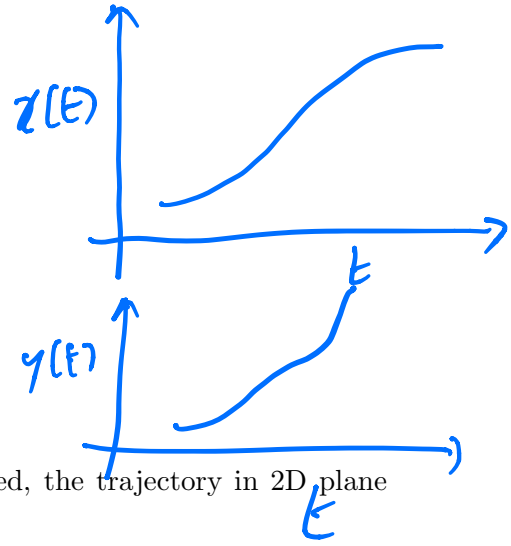


Depending on the number of given constraints i.e., rank of the matrices B_x $(p \times q)$ and B_k $(m \times n)$, the system can be categorized as under constrained, critically constrained and over constrained.

1. **Critically constrained:**
   p=q, rank(B_x)=p $\implies$ $pinv(B_x) = B_x^{-1}$
   m=n, rank(B_k)=m $\implies$ $pinv(B_k) = B_k^{-1}$

2. **Under constrained:**
   $p < q$, rank$(B_x) = p$ $\implies$ $pinv(B_x) = (B_x^T B_x)^{-1} B_x^T$
   $m < n$, rank$(B_k) = m$ $\implies$ $pinv(B_k) = (B_k^T B_k)^{-1} B_k^T$

3. **Over constrained:**
   $p > q$, rank$(B_x) = q$ $\implies$ $pinv(B_x) = B_x^T (B_x B_x^T)^{-1}$
   $m > n$, rank$(B_k) = n$ $\implies$ $pinv(B_k) = B_k^T (B_k B_k^T)^{-1}$



Once, the weight parameters W_k and W_x are determined, the trajectory in 2D plane can be estimated as following,

$$x(t) = W_{x_0} B_0(\mu(t)) + W_{x_1} B_1(\mu(t)) + W_{x_2} B_2(\mu(t)) + W_{x_3} B_3(\mu(t)) + W_{x_4} B_4(\mu(t)) + W_{x_5} B_5(\mu(t)) \tag{26}$$

$$y(t) = y_0 + W_{k_0} F_0(t) + W_{k_1} F_1(t) + W_{k_2} F_2(t) + W_{k_3} F_3(t) + W_{k_4} F_4(t) + W_{k_5} F_5(t) \tag{27}$$

and the orientation of the robot can be defined by-

$$\theta(t) = \arctan((W_{k_0} B_0(\mu(t) + W_{k_1} B_1(\mu(t)) + W_{k_2} B_2(\mu(t)) + W_{k_3} B_3(\mu(t)) + W_{k_4} B_4(\mu(t)) + W_{k_5} B_5(\mu(t)) \tag{28}$$

Kaustab Pal
kaustab21@gmail.com

Quadratic form

We need to write the MPC cost function in the quadratic form $\frac{1}{2}x^T P x + q^T x$ where $x = [v \;\; w]^T$ and $P = \begin{bmatrix} A & B \\ B & C \end{bmatrix}$ and $q = \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}$

$\therefore$ expanding we get $[v \;\; w] \begin{bmatrix} A & B \\ B & C \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} + [q_1 \;\; q_2] \begin{bmatrix} v \\ w \end{bmatrix}$

$$= [v \;\; w] \begin{bmatrix} Av + Bw \\ Bv + Cw \end{bmatrix} + q_1 v + q_2 w$$

$$= Av^2 + 2Bvw + Cw^2 + q_1 v + q_2 w$$

The current MPC cost function is

$$\text{cost} = (x_t - x_g)^2 + (y_t - y_g)^2 + (\theta_t - \theta_g)^2$$

x_t = x-coordinate after timestep t

y_t = y-coordinate after timestep t

θ_t = θ after timestep t

Now we know x_c, y_c and θ_c (current state) and x_g, y_g and θ_g (goal state).

we need to calculate x_{c+t}, y_{c+t} and θ_{c+t}

For sake of simplicity, let's consider $t = 1$ and $c = 0$ (Initial state),

$$\theta_1 = \theta_c + w_1 \, dt \quad \text{— Linear}$$

$$x_1 = x_c + v_1 \cos(\theta_1) \, dt$$

$$y_1 = y_c + v_1 \sin(\theta_1) \, dt$$

Linearizing x_1 we get

$$x_1 = x_c + v_1 \cos(\theta_1)\, dt$$

$$x_1 = x_c + v_{1g} \cos(\theta_0 + w_{1g}\, dt)\, dt + \cos(\theta_0 + w_{1g}\, dt)\, dt\,(v_1 - v_{1g}) - v_{1g} \sin(\theta_0 + w_{1g}\, dt)\, dt^2\,(w_1 - w_{1g})$$

$$= x_c + v_{1g} \cos(\theta_0 + w_{1g}\, dt)\, dt + v_1 \cos(\theta_0 + w_{1g}\, dt)\, dt - v_{1g} \cos(\theta_0 + w_{1g}\, dt)\, dt - w_1 v_{1g} \sin(\theta_0 + w_{1g}\, dt)\, dt^2 + w_{1g} v_{1g} \sin(\theta_0 + w_{1g}\, dt)\, dt^2$$

$$\Rightarrow x_1 = x_c + v_1 \underbrace{\cos(\theta_0 + w_{1g}\, dt)\, dt}_{a} - w_1 \underbrace{v_{1g} \sin(\theta_0 + w_{1g}\, dt)\, dt^2}_{b} + w_{1g} \underbrace{v_{1g} \sin(\theta_0 + w_{1g}\, dt)\, dt^2}_{b}$$

$$= v_1 a - w_1 b + \underbrace{x_c + w_{1g} b}_{c}$$

$$a = \cos(\theta_0 + w_{1g}\, dt)\, dt$$

$$b = v_{1g} \sin(\theta_0 + w_{1g}\, dt)\, dt^2$$

$$c = x_c + w_{1g} b$$

$$\Rightarrow x_1 = v_1 a - w_1 b + c$$

Wow $(x_1 - x_g)^2$

$$= x_1^2 + x_g^2 - 2 x_1 x_g$$

$$= (v_1 a - w_1 b + c)^2 + x_g^2 - 2(v_1 a - w_1 b + c) x_g$$

$$= (v_1 a - w_1 b)^2 + c^2 + 2(v_1 a - w_1 b) c + x_g^2 - 2(v_1 a - w_1 b + c) x_g$$

$$= \underbrace{v_1^2 a^2 + w_1^2 b^2 - 2 v_1 w_1 ab}_{x^T P x} + c^2 + 2 v_1 ac - 2 w_1 bc + x_g^2 - 2 v_1 x_g a + 2 w_1 x_g b - 2 c x_g$$

$$= v_1^2 a^2 + w_1^2 b^2 - 2v_1 w_1 ab + v_1(2ac - 2x_g a) - w_1(2bc - 2x_g b) + \boxed{c^2 + x_g^2 - 2cx_g}$$

K = constant (we can remove this from the cost)

$$= v_1^2 a^2 + w_1^2 b^2 - 2v_1 w_1 ab + v_1(2ac - 2x_g a) - w_1(2bc - 2x_g b) \quad (1)$$

$y_1 = y_c + v_1 \sin(\theta_i)\, dt$

Now, linearizing y_1 we get

$$y_1 = y_c + v_{ig} \sin(\theta_0 + w_{ig} dt) dt + \sin(\theta_0 + w_{ig} dt) dt (v_1 - v_{ig}) + v_{ig} \cos(\theta_0 + w_{ig} dt)\, dt^2 (w_1 - w_{ig})$$

$$= y_c + \cancel{v_{ig} \sin(\theta_0 + w_{ig} dt) dt} + v_1 \sin(\theta_0 + w_{ig} dt) dt - \cancel{v_{ig} \sin(\theta_0 + w_{ig} dt) dt} + w_1 v_{ig} \cos(\theta_0 + w_{ig} dt) dt^2 - w_{ig} v_{ig} \cos(\theta_0 + w_{ig} dt) dt^2$$

$$= y_c + v_1 \boxed{\sin(\theta_0 + w_{ig} dt) dt} + w_1 \boxed{v_{ig} \cos(\theta_0 + w_{ig} dt) dt^2} - w_{ig} \boxed{v_{ig} \cos(\theta_0 + w_{ig} dt) dt^2}$$

d e e

$$= y_c + v_1 d + w_1 e - w_{ig} e$$

$$= v_1 d + w_1 e + \boxed{y_c - w_{ig} e}$$

f

$d = \sin(\theta_0 + w_{ig} dt) dt$

$e = v_{ig} \cos(\theta_0 + w_{ig} dt) dt^2$

$f = y_c - w_{ig} e$

$$\Rightarrow y_1 = v_1 d + w_1 e + f$$

Now $(y_1 - y_g)^2$

$= y_1^2 + y_g^2 - 2 y_1 y_g$

$= (v_1 d + w_1 e + f)^2 + y_g^2 - 2(v_1 d + w_1 e + f) y_g$

$= (v_1 d + w_1 e)^2 + f^2 + 2(v_1 d + w_1 e) f + y_g^2 - 2 v_1 d y_g - 2 w_1 e y_g - 2 f y_g$

$= v_1^2 d^2 + w_1^2 e^2 + 2 v_1 w_1 de + f^2 + 2 v_1 d f + 2 w_1 e f + y_g^2 - 2 v_1 d y_g - 2 w_1 e y_g - 2 f y_g$

$= v_1^2 d^2 + w_1^2 e^2 + 2 v_1 w_1 de + v_1 (2df - 2 d y_g) + w_1 (2ef - 2 e y_g) + f^2 + y_g^2 - 2 f y_g$

Constant so we can remove this part

$= v_1^2 d^2 + w_1^2 e^2 + 2 v_1 w_1 de + v_1 (2df - 2d y_g) + w_1 (2ef - 2 e y_g)$ — (2)

$\theta_1 = \theta_c + w_1 dt$

Now $(\theta_1 - \theta_g)^2$

$= \theta_1^2 + \theta_g^2 - 2 \theta_1 \theta_g$

$= (\theta_c + w_1 dt)^2 + \theta_g^2 - 2(\theta_c + w_1 dt) \theta_g$

$= \theta_c^2 + w_1^2 dt^2 + 2 \theta_c w_1 dt + \theta_g^2 - 2 \theta_c \theta_g - 2 w_1 dt \theta_g$

$= w_1^2 dt^2 + w_1 (2 \theta_c dt - 2 \theta_g dt) + \theta_c^2 + \theta_g^2 - 2 \theta_c \theta_g$

Constant so can be removed

$= w_1^2 dt^2 + w_1 (2 \theta_0 dt - 2 \theta_g dt)$ — (3)

$$\text{cost} = ① + ② + ③$$

$$= V_1^2 a^2 + W_1^2 b^2 - 2V_1 W_1 ab + V_1(2ac - 2x_g a) - W_1(2bc - 2x_g b)$$
$$+ V_1^2 d^2 + W_1^2 e^2 + 2V_1 W_1 de + V_1(2df - 2dy_g) + W_1(2ef - 2ey_g)$$
$$+ W_1^2 dt^2 + W_1(2\theta_0 dt - 2\theta_g dt)$$

$$= V_1^2 \underbrace{(a^2 + d^2)}_{A} + W_1^2 \underbrace{(b^2 + e^2 + dt^2)}_{C} + 2V_1 W_1 \underbrace{(-ab + de)}_{B}$$
$$+ V_1 \underbrace{(2ac - 2x_g a + 2df - 2dy_g)}_{q_1} + W_1 \underbrace{(-2bc + 2x_g b + 2ef - 2ey_g + 2\theta_0 dt - 2\theta_g dt)}_{q_2}$$

$$= \frac{1}{2} \begin{bmatrix} V_1 & W_1 \end{bmatrix} \underbrace{2\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{P} \begin{bmatrix} V_1 \\ W_1 \end{bmatrix} + \underbrace{\begin{bmatrix} q_1 & q_2 \end{bmatrix}}_{q^T} \begin{bmatrix} V_1 \\ W_1 \end{bmatrix}$$

$$P = 2\begin{bmatrix} A & B \\ B & C \end{bmatrix} \qquad q = \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}$$

Now

$$A = a^2 + d^2$$

$$B = -ab + de$$

$$C = b^2 + e^2 + dt^2$$

$$q_1 = 2ac - 2x_g a + 2df - 2dy_g$$

$$q_2 = -2bc + 2x_g b + 2ef - 2ey_g + 2\theta_0 dt - 2\theta_g dt$$

$$a = \cos(\theta_0 + w_{ig} dt)\, dt$$

$$b = v_{ig} \sin(\theta_0 + w_{ig} dt)\, dt^2$$

$$c = x_0 + w_{ig} b$$

$$d = \sin(\theta_0 + w_{ig} dt)\, dt$$

$$e = v_{ig} \cos(\theta_0 + w_{ig} dt)\, dt^2$$

$$f = y_0 - w_{ig} e$$

Constraints

$$v_{lb} \leq v \leq v_{ub}$$

$$w_{lb} \leq w \leq w_{ub}$$

$$v_g - t_b \leq v \leq v_g + t_b$$

$$w_g - t_b \leq w \leq w_g + t_b$$

] trust region constraints

The constraints needs to be written in the form $Gx \leq h$, $Ax = b$

$$\therefore \quad G = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \\ 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix}$$

$$b = \begin{bmatrix} v_{ub} \\ v_{lb} \\ w_{ub} \\ w_{lb} \\ v_g + \text{trust\_boundary} \\ -(v_g - \text{trust\_boundary}) \\ w_g + \text{trust\_boundary} \\ -(w_g - \text{trust\_boundary}) \end{bmatrix}$$

For $t = 3$, the cost becomes

$$\text{cost} = (x_3 - x_g)^2 + (y_3 - y_g)^2 + (\theta_3 - \theta_g)^2$$

Solving for x

$$x_3 = x_0 + x_1 + x_2 + v_3 \cos(\theta_3)\, dt$$

$$= x_0 + v_1 \cos(\theta_1) dt + v_2 \cos(\theta_2) dt + v_3 \cos(\theta_3) dt$$

$$\Rightarrow x_3 = x_0 + v_1 \cos(\theta_0 + w_1 dt) dt + v_2 \cos(\theta_0 + (w_1 + w_2) dt) dt + v_3 \cos(\theta_0 + (w_1 + w_2 + w_3) dt) dt$$

Linearizing x_3 we get

$$x_3 = \underbrace{x_0 + v_{1g} \cos(\theta_0 + w_{1g} dt) dt + v_{2g} \cos(\theta_0 + (w_{1g} + w_{2g}) dt) dt + v_{3g} \cos(\theta_0 + (w_{1g} + w_{2g} + w_{3g}) dt) dt}_{K_1}$$

$$+ \cos(\theta_0 + w_{1g} dt) dt\, (v_1 - v_{1g})$$

$$+ \cos(\theta_0 + (w_{1g} + w_{2g}) dt) dt\, (v_2 - v_{2g})$$

$$+ \cos(\theta_0 + (w_{1g} + w_{2g} + w_{3g}) dt) dt\, (v_3 - v_{3g})$$

$$- \left(v_{1g} \sin(\theta_0 + w_{1g} dt) dt^2 + v_{2g} \sin(\theta_0 + (w_{1g} + w_{2g}) dt) dt^2 + v_{3g} \sin(\theta_0 + (w_{1g} + w_{2g} + w_{3g}) dt) dt^2\right)(w_1 - w_{1g})$$

$$- \left(v_{2g} \sin(\theta_0 + (w_{1g} + w_{2g}) dt) dt^2 + v_{3g} \sin(\theta_0 + (w_{1g} + w_{2g} + w_{3g}) dt) dt^2\right)(w_2 - w_{2g})$$

$$- \left(v_{3g} \sin(\theta_0 + (w_{1g} + w_{2g} + w_{3g}) dt) dt^2\right)(w_3 - w_{3g})$$

$$= K_1 + v_1 \cos(\theta_0 + w_{1g} dt) dt - v_{1g} \cos(\theta_0 + w_{1g} dt) dt$$

$$+ v_2 \cos(\theta_0 + (w_{1g} + w_{2g}) dt) dt - v_{2g} \cos(\theta_0 + (w_{1g} + w_{2g}) dt) dt$$

$$+ v_3 \cos(\theta_0 + (w_{1g} + w_{2g} + w_{3g}) dt) dt - v_{3g} \cos(\theta_0 + (w_{1g} + w_{2g} + w_{3g}) dt) dt$$

$$- V_{1g}\sin(\theta_0 + w_{1g}\,dt)\,dt^2\,(w_1 - w_{1g}) - V_{2g}\sin(\theta_0 + (w_{1g}+w_{2g})\,dt)\,dt^2(w_1 - w_{1g})$$
$$- V_{3g}\sin(\theta_0 + (w_{1g}+w_{2g}+w_{3g})\,dt)\,dt^2\,(w_1 - w_{1g})$$
$$- V_{2g}\sin(\theta_0 + (w_{1g}+w_{2g})\,dt)\,dt^2\,(w_2 - w_{2g})$$
$$- V_{3g}\sin(\theta_0 + (w_{1g}+w_{2g}+w_{3g})\,dt)\,dt^2\,(w_2 - w_{2g})$$
$$- V_{3g}\sin(\theta_0 + (w_{1g}+w_{2g}+w_{3g})\,dt)\,dt^2\,(w_3 - w_{3g})$$

$$= K_1 + V_1 \underbrace{\cos(\theta_0 + w_{1g}\,dt)\,dt}_{a} - V_{1g}\cos(\theta_0 + w_{1g}\,dt)\,dt$$
$$+ V_2 \underbrace{\cos(\theta_0 + (w_{1g}+w_{2g})\,dt)\,dt}_{b} - V_{2g}\cos(\theta_0 + (w_{1g}+w_{2g})\,dt)\,dt$$
$$+ V_3 \underbrace{\cos(\theta_0 + (w_{1g}+w_{2g}+w_{3g})\,dt)\,dt}_{c} - V_{3g}\cos(\theta_0 + (w_{1g}+w_{2g}+w_{3g})\,dt)\,dt$$
$$- w_1 V_{1g}\sin(\theta_0 + w_{1g}\,dt)\,dt^2 + w_{1g} V_{1g}\sin(\theta_0 + w_{1g}\,dt)\,dt^2$$
$$- w_1 V_{2g}\sin(\theta_0 + (w_{1g}+w_{2g})\,dt)\,dt^2 + w_{1g} V_{2g}\sin(\theta_0 + (w_{1g}+w_{2g})\,dt)\,dt^2$$
$$- w_1 V_{3g}\sin(\theta_0 + (w_{1g}+w_{2g}+w_{3g})\,dt)\,dt^2 + w_{1g} V_{3g}\sin(\theta_0 + (w_{1g}+w_{2g}+w_{3g})\,dt)\,dt^2$$

$$- w_2 V_{2g}\sin(\theta_0 + (w_{1g}+w_{2g})\,dt)\,dt^2 + w_{2g} V_{2g}\sin(\theta_0 + (w_{1g}+w_{2g})\,dt)\,dt^2$$
$$- w_2 V_{3g}\sin(\theta_0 + (w_{1g}+w_{2g}+w_{3g})\,dt)\,dt^2 + w_{2g} V_{3g}\sin(\theta_0 + (w_{1g}+w_{2g}+w_{3g})\,dt)\,dt^2$$
$$- w_3 V_{3g}\sin(\theta_0 + (w_{1g}+w_{2g}+w_{3g})\,dt)\,dt^2 + w_{3g} V_{3g}\sin(\theta_0 + (w_{1g}+w_{2g}+w_{3g})\,dt)\,dt^2$$

$$= \underbrace{K_1 - V_{1g}\cos(\theta_0 + w_{1g}dt)dt - V_{2g}\cos(\theta_0 + (w_{1g}+w_{2g})\,dt)\,dt - V_{3g}\cos(\theta_0 + (w_{1g}+w_{2g}+w_{3g})dt)dt}_{K}$$
$$+ w_{1g} V_{1g}\sin(\theta_0 + w_{1g}\,dt)\,dt^2 + w_{1g} V_{2g}\sin(\theta_0 + (w_{1g}+w_{2g})\,dt)\,dt^2$$
$$+ w_{1g} V_{3g}\sin(\theta_0 + (w_{1g}+w_{2g}+w_{3g})\,dt)\,dt^2 + w_{2g} V_{2g}\sin(\theta_0 + (w_{1g}+w_{2g})\,dt)\,dt^2$$
$$+ w_{2g} V_{3g}\sin(\theta_0 + (w_{1g}+w_{2g}+w_{3g})\,dt)\,dt^2 + w_{3g} V_{3g}\sin(\theta_0 + (w_{1g}+w_{2g}+w_{3g})\,dt)\,dt^2$$
$$+ V_1 a + V_2 b + V_3 c$$
$$+ w_1 \Big(\underbrace{-V_{1g}\sin(\theta_0 + w_{1g}\,dt)\,dt^2}_{d} + \underbrace{-V_{2g}\sin(\theta_0 + (w_{1g}+w_{2g})\,dt)\,dt^2}_{e}$$
$$+ \underbrace{-V_{3g}\sin(\theta_0 + (w_{1g}+w_{2g}+w_{3g})\,dt)\,dt^2}_{f} \Big)$$

$$+ w_2 \left(\underbrace{-v_{2g} \sin\left(\theta_0 + (w_{1g} + w_{2g})\,dt\right) dt^2}_{e} \underbrace{- v_{3g} \sin(\theta_0 + (w_{1g} + w_{2g} + w_{3g})\,dt)\,dt^2}_{\phi} \right)$$

$$+ w_3 \left(\underbrace{-v_{3g} \sin\left(\theta_0 + (w_{1g} + w_{2g} + w_{3g})\,dt\right) dt^2}_{\phi} \right)$$

$$= K + v_1 a + v_2 b + v_3 c + w_1(dte + f) + w_2(e + f) + w_3 f$$

Now

$$\text{cost} = (x_3 - x_g)^2$$

$$= \left(K - x_g + v_1 a + v_2 b + v_3 c + w_1(dte + f) + w_2(e + f) + w_3 f \right)^2$$

$$= \underbrace{(K - x_g)^2}_{z} + \left(v_1 a + v_2 b + v_3 c + w_1(dte + f) + w_2(e + f) + w_3 f \right)^2 + 2\underbrace{(K - x_g)}_{z}\left(v_1 a + v_2 b + v_3 c + w_1(dte + f) + w_2(e + f) + w_3 f \right)$$

$$= (v_1 a + v_2 b + v_3 c)^2 + (w_1(dte + f) + w_2(e + f) + w_3 f)^2 + 2(v_1 a + v_2 b + v_3 c)(w_1(dte + f) + w_2(e + f) + w_3 f) + 2z(v_1 a + v_2 b + v_3 c + w_1(dte + f) + w_2(e + f) + w_3 f)$$

$$= (v_1 a + v_2 b)^2 + v_3^2 c^2 + 2(v_1 a + v_2 b) v_3 c + (w_1(dte + f) + w_2(e + f))^2 + w_3^2 f^2 + 2(w_1(dte + f) + w_2(e + f)) w_3 f + 2v_1 w_1 a(dte + f) + 2v_1 w_2 a(e + f) + 2v_1 w_3 af + 2v_2 w_1 b(dte + f) + 2v_2 w_2 b(e + f) + 2v_2 w_3 bf + 2v_3 w_1 c(dte + f) + 2v_3 w_2 c(e + f) + 2v_3 w_3 cf$$

This part will be the $x^T P x$ part

$$+ 2v_1 az + 2v_2 bz + 2v_3 cz + 2w_1(dte + f)z + 2w_2(e + f)z + 2w_3 fz$$

This part will be the $q^T x$ part

Simplifying the $x^T P x$ part

$$v_1^2 a^2 + v_2^2 b^2 + v_3^2 c^2 + 2 v_1 v_2 ab + 2 v_1 v_3 ac + 2 v_2 v_3 bc$$
$$+ w_1^2 (d+e+f)^2 + w_2^2 (e+f)^2 + w_3^2 f^2 + 2 w_1 w_2 (d+e+f)(e+f)$$
$$+ 2 w_1 w_3 (d+e+f) f + 2 w_2 w_3 (e+f) f + 2 v_1 w_1 a (d+e+f)$$
$$+ 2 v_1 w_2 a (e+f) + 2 v_1 w_3 af + 2 v_2 w_1 b (d+e+f) + 2 v_2 w_2 b (e+f)$$
$$+ 2 v_2 w_3 bf + 2 v_3 w_1 c (d+e+f) + 2 v_3 w_3 c (e+f)$$
$$+ 2 v_3 w_3 cf.$$

$$= a^2 v_1^2 + b^2 v_2^2 + c^2 v_3^2 + (d+e+f)^2 w_1^2 + (e+f)^2 w_2^2 + f^2 w_3^2$$
$$+ 2 v_1 v_2 ab + 2 v_1 v_3 ac + 2 v_1 w_1 a (d+e+f) + 2 v_1 w_2 a (e+f)$$
$$+ 2 v_1 w_3 af + 2 v_2 v_3 bc + 2 v_2 w_1 b (d+e+f) + 2 v_2 w_2 b (e+f)$$
$$+ 2 v_2 w_3 bf + 2 v_3 w_1 c (d+e+f) + 2 v_3 w_2 c (e+f)$$
$$+ 2 v_3 w_3 cf + 2 w_1 w_2 (d+e+f)(e+f) + 2 w_1 w_3 (d+e+f) f$$
$$+ 2 w_2 w_3 (e+f) f$$

$$= \begin{bmatrix} v_1 & v_2 & v_3 & w_1 & w_2 & w_3 \end{bmatrix}$$

$$P_x = 2 * \begin{bmatrix}
a^2 & ab & ac & a(d+e+f) & a(e+f) & af \\
ab & b^2 & bc & b(d+e+f) & b(e+f) & bf \\
ac & bc & c^2 & c(d+e+f) & c(e+f) & cf \\
a(d+e+f) & b(d+e+f) & c(d+e+f) & (d+e+f)^2 & (d+e+f)(e+f) & (d+e+f) f \\
a(e+f) & b(e+f) & c(e+f) & (d+e+f)(e+f) & (e+f)^2 & (e+f) f \\
af & bf & cf & (d+e+f) f & (e+f) f & f^2
\end{bmatrix}
\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

Simplifying $q^T K$ part

$$2V_1 aZ + 2V_2 bZ + 2V_3 CZ + 2W_1(d+e+f)Z + 2W_2(e+f)Z + 2W_3 fZ$$

$$= \begin{bmatrix} 2aZ & 2bZ & 2CZ & 2(d+e+f)Z & 2(e+f)Z & 2fZ \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ W_1 \\ W_2 \\ W_3 \end{bmatrix}$$

$$a = \cos(\theta_0 + W_{1g}dt)\,dt$$

$$b = \cos(\theta_0 + (W_{1g}+W_{2g})dt)\,dt$$

$$c = \cos(\theta_0 + (W_{1g}+W_{2g}+W_{3g})dt)\,dt$$

$$d = -V_{1g}\sin(\theta_0 + W_{1g}dt)\,dt^2$$

$$e = -V_{2g}\sin(\theta_0 + (W_{1g}+W_{2g})dt)\,dt^2$$

$$f = -V_{3g}\sin(\theta_0 + (W_{1g}+W_{2g}+W_{3g})dt)\,dt^2$$

$$Z = K - K_g$$

$$K = K_1 - V_{1g}a - V_{2g}b - V_{3g}C - W_{1g}(d+e+f) - W_{2g}(e+f) - W_{3g}f$$

$$K_1 = K_0 + V_{1g}a + V_{2g}b + V_{3g}C$$

$$P_4 = \begin{bmatrix} a \\ b \\ c \\ d+e+f \\ e+f \\ f \end{bmatrix} \begin{bmatrix} a & b & c & d+e+f & e+f & f \end{bmatrix}$$

Solving for y_3

$$y_3 = y_0 + y_1 + y_2 + V_3 \sin(\theta_3)\,dt$$

$$= y_0 + V_1 \sin(\theta_0 + w_1 dt)\,dt + V_2 \sin(\theta_0 + (w_1 + w_2)\,dt)\,dt + V_3 \sin(\theta_0 + (w_1 + w_2 + w_3)\,dt)\,dt$$

Linearizing y_3, we get

$$y_0 + V_{1g} \sin(\theta_0 + w_{1g}\,dt)\,dt + V_{2g} \sin(\theta_0 + (w_{1g} + w_{2g})\,dt)\,dt + V_{3g} \sin(\theta_0 + (w_{1g} + w_{2g} + w_{3g})\,dt)\,dt \quad K_1$$

$$+ \sin(\theta_0 + w_{1g}\,dt)\,dt\,(V_1 - V_{1g}) + \sin(\theta_0 + (w_{1g} + w_{2g})\,dt)\,dt\,(V_2 - V_{2g})$$

$$+ \sin(\theta_0 + (w_{1g} + w_{2g} + w_{3g})\,dt)\,dt\,(V_3 - V_{3g})$$

$$+ V_{1g} \cos(\theta_0 + w_{1g}\,dt)\,dt^2\,(w_1 - w_{1g})$$

$$+ V_{2g} \cos(\theta_0 + (w_{1g} + w_{2g})\,dt)\,dt^2\,(w_1 - w_{1g})$$

$$+ V_{3g} \cos(\theta_0 + (w_{1g} + w_{2g} + w_{3g})\,dt)\,dt^2\,(w_1 - w_{1g})$$

$$+ V_{2g} \cos(\theta_0 + (w_{1g} + w_{2g})\,dt)\,dt^2\,(w_2 - w_{2g})$$

$$+ V_{3g} \cos(\theta_0 + (w_{1g} + w_{2g} + w_{3g})\,dt)\,dt^2\,(w_2 - w_{2g})$$

$$+ V_{3g} \cos(\theta_0 + (w_{1g} + w_{2g} + w_{3g})\,dt)\,dt^2\,(w_3 - w_{3g})$$

$$= K_1 + V_1 \sin(\theta_0 + w_{1g}\,dt)\,dt - V_{1g} \sin(\theta_0 + w_{1g}\,dt)\,dt$$

$$+ V_2 \sin(\theta_0 + (w_{1g} + w_{2g})\,dt)\,dt - V_{2g} \sin(\theta_0 + (w_{1g} + w_{2g})\,dt)\,dt$$

$$+ V_3 \sin(\theta_0 + (w_{1g} + w_{2g} + w_{3g})\,dt)\,dt - V_{3g} \sin(\theta_0 + (w_{1g} + w_{2g} + w_{3g})\,dt)\,dt$$

$$+ w_1 V_{1g} \cos(\theta_0 + w_{1g}\,dt)\,dt^2 - w_{1g} V_{1g} \cos(\theta_0 + w_{1g}\,dt)\,dt^2$$

$$+ w_1 V_{2g} \cos(\theta_0 + (w_{1g} + w_{2g})\,dt)\,dt^2 - w_{1g} V_{2g} \cos(\theta_0 + (w_{1g} + w_{2g})\,dt)\,dt^2$$

$$+ w_1 V_{3g} \cos\left(\theta_0 + (w_{1g} + w_{2g} + w_{3g})\,dt\right) dt^2 - w_{1g} V_{3g} \cos\left(\theta_0 + (w_{1g} + w_{2g} + w_{3g})\,dt\right) dt^2$$

$$+ w_2 V_{2g} \cos\left(\theta_0 + (w_{1g} + w_{2g})\,dt\right) dt^2 - w_{2g} V_{2g} \cos\left(\theta_0 + (w_{1g} + w_{2g})\,dt\right) dt^2$$

$$+ w_2 V_{3g} \cos\left(\theta_0 + (w_{1g} + w_{2g} + w_{3g})\,dt\right) dt^2 - w_{2g} V_{3g} \cos\left(\theta_0 + (w_{1g} + w_{2g} + w_{3g})\,dt\right) dt^2$$

$$+ w_3 V_{3g} \cos\left(\theta_0 + (w_{1g} + w_{2g} + w_{3g})\,dt\right) dt^2 - w_{3g} V_{3g} \cos\left(\theta_0 + (w_{1g} + w_{2g} + w_{3g})\,dt\right) dt^2$$

$$= K_1 - V_{1g} \sin(\theta_0 + w_{1g}\,dt)\,dt - V_{2g} \sin\left(\theta_0 + (w_{1g} + w_{2g})\,dt\right) dt$$

$$- V_{3g} \sin\left(\theta_0 + (w_{1g} + w_{2g} + w_{3g})\,dt\right) dt - w_{1g} V_{1g} \cos(\theta_0 + w_{1g}\,dt)\,dt^2$$

$$- w_{1g} V_{2g} \cos\left(\theta_0 + (w_{1g} + w_{2g})\,dt\right) dt^2 - w_{1g} V_{3g} \cos\left(\theta_0 + (w_{1g} + w_{2g} + w_{3g})\,dt\right) dt^2$$

$$- w_{2g} V_{2g} \cos\left(\theta_0 + (w_{1g} + w_{2g})\,dt\right) dt^2 - w_{2g} V_{3g} \cos\left(\theta_0 + (w_{1g} + w_{2g} + w_{3g})\,dt\right) dt^2$$

$$- w_{3g} V_{3g} \cos\left(\theta_0 + (w_{1g} + w_{2g} + w_{3g})\,dt\right) dt^2$$

$$+ V_1 \sin(\theta_0 + w_{1g}\,dt)\,dt + V_2 \sin\left(\theta_0 + (w_{1g} + w_{2g})\,dt\right) dt$$

$$+ V_3 \sin\left(\theta_0 + (w_{1g} + w_{2g} + w_{3g})\,dt\right) dt + w_1 V_{1g} \cos(\theta_0 + w_{1g}\,dt)\,dt^2$$

$$+ V_{2g} \cos\left(\theta_0 + (w_{1g} + w_{2g})\,dt\right) dt^2 + V_{3g} \cos\left(\theta_0 + (w_{1g} + w_{2g} + w_{3g})\,dt\right) dt^2$$

$$+ w_2 \left(V_{2g} \cos\left(\theta_0 + (w_{1g} + w_{2g})\,dt\right) dt^2 + V_{3g} \cos\left(\theta_0 + (w_{1g} + w_{2g} + w_{3g})\,dt\right) dt^2 \right)$$

$$+ w_3 V_{3g} \cos\left(\theta_0 + (w_{1g} + w_{2g} + w_{3g})\,dt\right) dt^2$$

$$= K + V_1 a + V_2 b + V_3 c + w_1 (d + e + f) + w_2 (e + f) + w_3 f$$

Now the cost is

$$(y_3 - y_g)^2$$

$$= \left(K - y_g + V_1 a + V_2 b + V_3 c + w_1 (d + e + f) + w_2 (e + f) + w_3 f \right)^2$$

$$= \underbrace{z^2}_{\text{constant}} + (v_1a + v_2b + v_3c + w_1(d+e+f) + w_2(e+f) + w_3 f)^2$$
$$+ 2z(v_1a + v_2b + v_3c + w_1(d+e+f) + w_2(e+f) + w_3 f)$$

$$= \underbrace{(v_1a + v_2b + v_3c)^2 + (w_1(d+e+f) + w_2(e+f) + w_3 f)^2 + 2(v_1a + v_2b + v_3c)(w_1(d+e+f) + w_2(e+f) + w_3 f)}_{\text{Quadratic part}}$$
$$+ \underbrace{2z(v_1a + v_2b + v_3c + w_1(d+e+f) + w_2(e+f) + w_3 f)}_{\text{Linear part}}$$

Simplifying the quadratic part, we get

$$\begin{bmatrix} v_1 & v_2 & v_3 & w_1 & w_2 & w_3 \end{bmatrix} P_y \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

$P_y = 2 *$

a^2	ab	ac	$a(d+e+f)$	$a(e+f)$	af
ab	b^2	bc	$b(d+e+f)$	$b(e+f)$	bf
ac	bc	c^2	$c(d+e+f)$	$c(e+f)$	cf
$a(d+e+f)$	$b(d+e+f)$	$c(d+e+f)$	$(d+e+f)^2$	$(d+e+f)(e+f)$	$(d+e+f)f$
$a(e+f)$	$b(e+f)$	$c(e+f)$	$(d+e+f)(e+f)$	$(e+f)^2$	$(e+f)f$
af	bf	cf	$(d+e+f)f$	$(e+f)f$	f^2

Simplifying the linear part

$$\underbrace{\begin{bmatrix} 2za & 2zb & 2zc & 2z(d+e+f) & 2z(e+f) & 2zf \end{bmatrix}}_{q_y^T} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

Here

$$a = \sin(\theta_0 + w_{1g}\,dt)\,dt$$

$$b = \sin(\theta_0 + (w_{1g}+w_{2g})\,dt)\,dt$$

$$c = \sin(\theta_0 + (w_{1g}+w_{2g}+w_{3g})\,dt)\,dt$$

$$d = v_{1g}\cos(\theta_0 + w_{1g}\,dt)\,dt^2$$

$$e = v_{2g}\cos(\theta_0 + (w_{1g}+w_{2g})\,dt)\,dt^2$$

$$f = v_{3g}\cos(\theta_0 + (w_{1g}+w_{2g}+w_{3g})\,dt)\,dt^2$$

$$Z = K - y_g$$

$$K = K_1 - v_{1g}a - v_{2g}b - v_{3g}c - w_{1g}(d+e+f) - w_{2g}(e+f) - w_{3g}f$$

$$K_1 = y_0 + v_{1g}a + v_{2g}b + v_{3g}c$$

Solving for θ_3

Now $\theta_3 = \theta_0 + \theta_1 + \theta_2 + w_3\,dt$

$$\Rightarrow \theta_3 = \theta_0 + (w_1 + w_2 + w_3)\,dt$$

Cost is

$$(\theta_3 - \theta_g)^2$$

$$\therefore (\theta_0 + (w_1+w_2+w_3)\,dt - \theta_g)^2$$

$$= (\theta_0 - \theta_g)^2 + (w_1\,dt + w_2\,dt + w_3\,dt)^2 + 2Z(w_1\,dt + w_2\,dt + w_3\,dt)$$

$$= (w_1\,dt + w_2\,dt)^2 + w_3^2\,dt^2 + 2(w_1\,dt + w_2\,dt)\,w_3\,dt + 2Z(w_1\,dt + w_2\,dt + w_3\,dt)$$

$$= w_1^2\,dt^2 + w_2^2\,dt^2 + 2w_1w_2\,dt^2 + w_3^2\,dt^2 + 2w_1w_3\,dt^2 + 2w_2w_3\,dt^2 + 2Zw_1\,dt + 2Zw_2\,dt + 2Zw_3\,dt$$

$$= w_1^2\,dt^2 + w_2^2\,dt^2 + w_3^2\,dt^2 + 2w_1w_2\,dt^2 + 2w_1w_3\,dt^2 + 2w_2w_3\,dt^2 + 2Zw_1\,dt + 2Zw_2\,dt + 2Zw_3\,dt$$

$$\begin{bmatrix} v_1 & v_2 & v_3 & w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & dt^2 & dt^2 & dt^2 \\ 0 & 0 & 0 & dt^2 & dt^2 & dt^2 \\ 0 & 0 & 0 & dt^2 & dt^2 & dt^2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

$P_\theta = 2*$

$$q_\theta = \begin{bmatrix} 0 & 0 & 0 & 2_z\,dt & 2_z\,dt & 2_z\,dt \end{bmatrix}$$

$$\therefore \quad P = 2*(P_x + P_y + P_\theta)$$

$$q = (q_x + q_y + q_\theta)$$

$$
\begin{bmatrix}
0 & -1 \\
-1 & 0 \\
1 & 0 \\
0 & 1 \\
1 & 0 \\
-1 & 0 \\
0 & 1 \\
0 & -1
\end{bmatrix}
\begin{bmatrix} v \\ w \end{bmatrix}
$$

$$
\begin{bmatrix}
w - w_{lb} \\
0 \quad v_{lb} \\
v - v_{lb} \\
w - w_{ub} \\
v_g + s \\
-(v_g - s) \\
w_g + c_1 \\
-(w_g - c_1)
\end{bmatrix}
\begin{matrix}
w_{lb} < w \\
-w \leq -w_{lb} \\
\\
v_{lb} \leq v \\
-v \leq -v_{lb}
\end{matrix}
$$

$$
\begin{bmatrix}
1 & 0 \\
0 & 1 \\
-1 & 0 \\
0 & -1 \\
1 & 0 \\
0 & 1
\end{bmatrix}
\begin{bmatrix} v \\ w \end{bmatrix}
$$

$$
\begin{matrix}
v_{ub} & v_{ub} \\
w_{ub} & v_{lb} \\
v_{lb} & w_{ub} \\
w_{lb} & w_{lb} \\
v_g + s & v_g + s \\
v_{g+c_1} & -(v_g - s) \\
 & w_g + c_1 \\
 & -(w_g - c_1)
\end{matrix}
$$

$$a_{min} \leq \frac{V_2 - V_1}{dt} \leq a_{max}$$

$$a_{min} \leq \frac{V_3 - V_2}{dt} \leq a_{max}$$

$$a_{min} \leq \frac{w_2 - w_1}{dt} \leq a_{max}$$

$$a_{min} \leq \frac{w_3 - w_2}{dt} \leq a_{max}$$

$$a_{min} \leq V_2 - V_1$$

$$-(V_2 - V_1) \leq -a_{min}$$

$$-V_2 + V_1$$

$$V_1 - V_2$$

$$V_2 - V_3$$

$$\begin{matrix} -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 \end{matrix}$$

V_1

V_2

V_3

w_1

w_2

w_3

$v_1 \quad v_2 \quad v_3$

$(v_2 - v_1)^2 + (v_3 - v_2)^2$

$= v_2^2 + v_1^2 - 2v_1v_2 + v_3^2 + v_2^2 - 2v_2v_3$

$= v_1^2 + v_2^2 + v_3^2 - 2v_1v_2 - 2v_2v_3$

$$\begin{pmatrix} v_1 & v_2 & v_3 \end{pmatrix} \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{matrix} v_1 \\ v_2 \\ v_3 \end{matrix}$$

$v_1^2 + v_1v_2 + v_1v_3$

$+ v_2^2$

$$v_1, v_2 v_3 \begin{pmatrix} v_1 + v_2 \\ v_1 + v_2 + v_3 \\ v_2 + v_3 \end{pmatrix}$$

$$\begin{matrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}$$

$v_1^2 + v_1v_2 + v_1v_2 + v_2^2 + v_2v_3 + v_2v_3 + v_3^2$

$v_1^2 + v_2^2 + v_3^2 + 2v_1v_2 + 2v_2v_3$

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix} \quad v_1 + v_2$$

$$\begin{pmatrix} v_1 & v_2 & v_3 & v_4 \end{pmatrix} \begin{bmatrix} v_1 + v_2 \\ v_1 + v_2 + v_3 \\ v_2 + v_3 + v_4 \\ v_3 + v_4 \end{bmatrix}$$

Static obstacle avoidance

For $t=1$

$$d(a,o) \leq (r_a + r_o)^2$$

$$d(a,o) = (x_1 - x_o)^2 + (y_1 - y_o)^2$$

Now x_1 and y_1 are functions of v_1 and w_1

$$(x_1 - x_o)^2 + (y_1 - y_o)^2 \leq (r_a + r_o)^2$$

$$= x_1^2 + x_o^2 - 2x_1x_o + y_1^2 + y_o^2 - 2y_1y_o$$

$$= x_1^2 - 2x_1x_o + y_1^2 - 2y_1y_o \leq (r_a + r_o)^2 - x_o^2 - y_o^2$$

Now $x_1 = x_o + v_1 \cos(\theta_o + w_1 dt)\, dt$

$$y_1 = y_o + v_1 \sin(\theta_o + w_1 dt)\, dt$$

$$\therefore (x_o + v_1 \cos(\theta_o + w_1 dt)\, dt)^2 + (y_o + v_1 \sin(\theta_o + w_1 dt)\, dt)^2$$
$$- 2(x_o + v_1 \cos(\theta_o + w_1 dt)\, dt)\, x_o - 2(y_o + v_1 \sin(\theta_o + w_1 dt)\, dt)\, y_o$$

$$= x_o^2 + v_1^2 \cos^2(\theta_o + w_1 dt)\, dt^2 + \cancel{2x_o v_1 \cos(\theta_o + w_1 dt)\, dt}$$
$$+ y_o^2 + v_1^2 \sin^2(\theta_o + w_1 dt)\, dt^2 + \cancel{2y_o v_1 \sin(\theta_o + w_1 dt)\, dt}$$
$$- 2x_o^2 - \cancel{2x_o v_1 \cos(\theta_o + w_1 dt)\, dt} - 2y_o^2 - \cancel{2y_o v_1 \sin(\theta_o + w_1 dt)\, dt}$$

$$= v_1^2 \cos^2(\theta_o + w_1 dt)\, dt^2 + v_1^2 \sin^2(\theta_o + w_1 dt)\, dt^2 - x_o^2 - y_o^2$$

$$= v_1^2 \cos^2(\theta_o + w_1 dt)\, dt^2 + v_1^2 \sin^2(\theta_o + w_1 dt)\, dt^2 \leq (r_a + r_o)^2$$

$$= v_1^2\, dt^2 \leq (r_a + r_o)^2$$

Linearizing LHS

$$v_{ig}^2\,dt^2 + 2v_{ig}\,dt^2\,(v_i - v_{ig}) = 2v_i v_{ig}\,dt^2 - v_{ig}^2\,dt^2$$

$$\therefore \boxed{2v_i v_{ig}\,dt^2 \le (r_a + r_o)^2 + v_{ig}^2\,dt^2} \quad \text{for 1 step}$$

For $t = 2$

$$d(a,o) = (x_2 - x_o)^2 + (y_2 - y_o)^2$$

$$x_2 = x_o + x_1 + v_2 \cos(\theta_o + (w_1 + w_2)\,dt)\,dt$$

$$y_2 = y_o + y_1 + v_2 \sin(\theta_o + (w_1 + w_2)\,dt)\,dt$$

Now

$$(x_2 - x_o)^2 + (y_2 - y_o)^2 \le (r_a + r_o)^2$$

$$\Rightarrow x_2^2 + x_o^2 - 2x_2x_o + y_2^2 + y_o^2 - 2y_2y_o \le (r_a + r_o)^2$$

$$\Rightarrow x_2^2 - 2x_2x_o + y_2^2 - 2y_2y_o \le (r_a + r_o)^2 - x_o^2 - y_o^2$$

Expanding LHS

$$\left(x_0 + v_1 \cos(\theta_0 + w_1 dt)dt + v_2 \cos(\theta_0 + (w_1+w_2)dt)\,dt\right)^2$$
$$- 2\left(x_0 + v_1 \cos(\theta_0 + w_1 dt)\,dt + v_2 \cos(\theta_0 + (w_1+w_2)dt)\,dt\right)x_0$$
$$+ \left(y_0 + v_1 \sin(\theta_0 + w_1 dt)\,dt + v_2 \sin(\theta_0 + (w_1+w_2)\,dt)\,dt\right)^2$$
$$- 2\left(y_0 + v_1 \sin(\theta_0 + w_1 dt)\,dt + v_2 \sin(\theta_0 + (w_1+w_2)\,dt)\,dt\right)y_0$$

$$= x_0^2 + \left(v_1 \cos(\theta_0 + w_1 dt)\,dt + v_2 \cos(\theta_0 + (w_1+w_2)\,dt)\,dt\right)^2$$
$$+ \cancel{2x_0\left(v_1 \cos(\theta_0 + w_1 dt)\,dt + v_2 \cos(\theta_0 + (w_1+w_2)dt)\,dt\right)}$$
$$- 2x_0^2 - \cancel{2x_0\left(v_1 \cos(\theta_0 + w_1 dt)\,dt + v_2 \cos(\theta_0 + (w_1+w_2)dt)\,dt\right)}$$
$$+ y_0^2 + \left(v_1 \sin(\theta_0 + w_1 dt)\,dt + v_2 \sin(\theta_0 + (w_1+w_2)\,dt)\,dt\right)^2$$
$$+ \cancel{2y_0\left(v_1 \sin(\theta_0 + w_1 dt)\,dt + v_2 \sin(\theta_0 + (w_1+w_2)\,dt)\,dt\right)}$$
$$- 2y_0^2 - \cancel{2y_0\left(v_1 \sin(\theta_0 + w_1 dt)\,dt + v_2 \sin(\theta_0 + (w_1+w_2)\,dt)\,dt\right)}$$

gets cancelled out with RHS

$$= \cancel{-x_0^2} + \left(v_1 \cos(\theta_0 + w_1 dt)\,dt + v_2 \cos(\theta_0 + (w_1+w_2)\,dt)\,dt\right)^2$$
$$\cancel{-y_0^2} + \left(v_1 \sin(\theta_0 + w_1 dt)\,dt + v_2 \sin(\theta_0 + (w_1+w_2)\,dt)\,dt\right)^2$$

$$= \left(v_1 \cos(\theta_0 + w_1 dt)\,dt + v_2 \cos(\theta_0 + (w_1+w_2)\,dt)\,dt\right)^2$$
$$+ \left(v_1 \sin(\theta_0 + w_1 dt)\,dt + v_2 \sin(\theta_0 + (w_1+w_2)\,dt)\,dt\right)^2$$

$$= v_1^2 \cos^2(\theta_0 + w_1 dt)\,dt^2 + v_2^2 \cos^2(\theta_0 + (w_1+w_2)\,dt)\,dt^2$$
$$+ 2v_1v_2 \cos(\theta_0 + w_1 dt)\cos(\theta_0 + (w_1+w_2)\,dt)\,dt^2$$
$$+ v_1^2 \sin^2(\theta_0 + w_1 dt)\,dt^2 + v_2^2 \sin^2(\theta_0 + (w_1+w_2)\,dt)\,dt^2$$
$$+ 2v_1v_2 \sin(\theta_0 + w_1 dt)\sin(\theta_0 + (w_1+w_2)\,dt)\,dt^2$$

$$= V_1^2 dt^2 \left(\cos^2(\theta_0 + w_1 dt) + \sin^2(\theta_0 + w_1 dt) \right)$$
$$+ V_2^2 dt^2 \left(\cos^2(\theta_0 + (w_1+w_2) dt) + \sin^2(\theta_0 + (w_1+w_2) dt) \right)$$
$$+ 2 V_1 V_2 dt^2 \left(\cos(\theta_0 + w_1 dt) \cos(\theta_0 + (w_1+w_2) dt) + \sin(\theta_0 + w_1 dt) \sin(\theta_0 + (w_1+w_2) dt) \right)$$

$$= V_1^2 dt^2 + V_2^2 dt^2 + 2 V_1 V_2 dt^2 \left(\cos(\theta_0 + w_1 dt) \cos(\theta_0 + (w_1+w_2) dt) + \sin(\theta_0 + w_1 dt) \sin(\theta_0 + (w_1+w_2) dt) \right)$$

$$= V_1^2 dt^2 + V_2^2 dt^2 + 2 V_1 V_2 dt^2 \cos\left(\theta_0 + w_1 dt - (\theta_0 + (w_1+w_2) dt) \right)$$

$$= V_1^2 dt^2 + V_2^2 dt^2 + 2 V_1 V_2 dt^2 \cos\left(\theta_0 + w_1 dt - \theta_0 - w_1 dt - w_2 dt \right)$$

$$= V_1^2 dt^2 + V_2^2 dt^2 + 2 V_1 V_2 dt^2 \cos(w_2 dt)$$

Linearizing, we get

$$V_{1g}^2 dt^2 + V_{2g}^2 dt^2 + 2 V_{1g} V_{2g} dt^2 \cos(w_{2g} dt)$$
$$+ \left(2 V_{1g} dt^2 + 2 V_{2g} dt^2 \cos(w_{2g} dt) \right) (V_1 - V_{1g})$$
$$+ \left(2 V_{2g} dt^2 + 2 V_{1g} dt^2 \cos(w_{2g} dt) \right) (V_2 - V_{2g})$$
$$+ \left(- 2 V_{1g} V_{2g} dt^3 \sin(w_{2g} dt) \right) (w_2 - w_{2g})$$

$$= V_{1g}^2 dt^2 + V_{2g}^2 dt^2 + 2 V_{1g} V_{2g} dt^2 \cos(w_{2g} dt)$$
$$+ 2 V_1 V_{1g} dt^2 + 2 V_1 V_{2g} dt^2 \cos(w_{2g} dt) - 2 V_{1g}^2 dt^2$$
$$- 2 V_{1g} V_{2g} dt^2 \cos(w_{2g} dt)$$

$$+ 2V_2V_{2g}\,dt^2 + 2V_2V_{1g}\,dt^2\cos(w_{2g}\,dt) - 2V_{2g}^2\,dt^2 - 2V_{1g}V_{2g}\,dt^2\cos(w_{2g}\,dt) - 2V_{1g}V_{2g}w_2\,dt^3\sin(w_{2g}\,dt) + 2V_{1g}V_{2g}w_{2g}\,dt^3\sin(w_{2g}\,dt)$$

$$= -V_{1g}^2\,dt^2 - V_{2g}^2\,dt^2 + 2V_1V_{1g}\,dt^2 + 2V_1V_{2g}\,dt^2\cos(w_{2g}\,dt) + 2V_2V_{2g}\,dt^2 + 2V_2V_{1g}\,dt^2\cos(w_{2g}\,dt) - 2V_{1g}V_{2g}\,dt^2\cos(w_{2g}\,dt) - 2V_{1g}V_{2g}w_2\,dt^3\sin(w_{2g}\,dt) + 2V_{1g}V_{2g}w_{2g}\,dt^3\sin(w_{2g}\,dt)$$

constant $= K$

$$= \boxed{-V_{1g}^2\,dt^2 - V_{2g}^2\,dt^2 - 2V_{1g}V_{2g}\,dt^2\cos(w_{2g}\,dt) + 2V_{1g}V_{2g}w_{2g}\,dt^3\sin(w_{2g}\,dt)} + V_1\left(2V_{1g}\,dt^2 + 2V_{2g}\,dt^2\cos(w_{2g}\,dt)\right) + V_2\left(2V_{2g}\,dt^2 + 2V_{1g}\,dt^2\cos(w_{2g}\,dt)\right) + w_2(-2)V_{1g}V_{2g}\,dt^3\sin(w_{2g}\,dt)$$

For $t=3$

$$(x_3 - x_0)^2 + (y_3 - y_0)^2 \leq (r_a + r_0)^2$$

$$\Rightarrow x_3^2 + x_0^2 - 2x_3x_0 + y_3^2 + y_0^2 - 2y_3y_0 \leq (r_a + r_0)^2$$

$$\Rightarrow x_3^2 - 2x_3x_0 + y_3^2 - 2y_3y_0 \leq (r_a + r_0)^2 - x_0^2 - y_0^2$$

Now,

$$x_3 = x_0 + v_1 \cos(\theta_0 + w_1 dt)\, dt + v_2 \cos(\theta_0 + (w_1 + w_2)\, dt)\, dt + v_3 \cos(\theta_0 + (w_1 + w_2 + w_3)\, dt)\, dt$$

$$y_3 = y_0 + v_1 \sin(\theta_0 + w_1 dt)\, dt + v_2 \sin(\theta_0 + (w_1 + w_2)\, dt)\, dt + v_3 \sin(\theta_0 + (w_1 + w_2 + w_3)\, dt)\, dt$$

Substituting x_3 and y_3, we get

$$\Big(x_0 + v_1 \cos(\theta_0 + w_1 dt)\, dt + v_2 \cos(\theta_0 + (w_1 + w_2)\, dt)\, dt + v_3 \cos(\theta_0 + (w_1 + w_2 + w_3)\, dt)\, dt\Big)^2$$

$$- 2x_0^2 - 2x_0\Big(v_1 \cos(\theta_0 + w_1 dt)\, dt + v_2 \cos(\theta_0 + (w_1 + w_2)\, dt)\, dt + v_3 \cos(\theta_0 + (w_1 + w_2 + w_3)\, dt)\, dt\Big)$$

$$+ \Big(y_0 + v_1 \sin(\theta_0 + w_1 dt)\, dt + v_2 \sin(\theta_0 + (w_1 + w_2)\, dt)\, dt + v_3 \sin(\theta_0 + (w_1 + w_2 + w_3)\, dt)\, dt\Big)^2$$

$$- 2y_0^2 - 2y_0\Big(v_1 \sin(\theta_0 + w_1 dt)\, dt + v_2 \sin(\theta_0 + (w_1 + w_2)\, dt)\, dt + v_3 \sin(\theta_0 + (w_1 + w_2 + w_3)\, dt)\, dt\Big)$$

gets cancelled out with RHS

$$= -x_0^2 + \left(V_1 \cos(\theta_0 + w_1 dt)\, dt + V_2 \cos(\theta_0 + (w_1 + w_2)\, dt)\, dt + V_3 \cos(\theta_0 + (w_1 + w_2 + w_3)\, dt)\, dt \right)^2$$

$$- y_0^2 + \left(V_1 \sin(\theta_0 + w_1 dt)\, dt + V_2 \sin(\theta_0 + (w_1 + w_2)\, dt)\, dt + V_3 \sin(\theta_0 + (w_1 + w_2 + w_3)\, dt)\, dt \right)^2$$

$$= V_1^2 \cos^2(\theta_0 + w_1 dt)\, dt^2 + \left(V_2 \cos(\theta_0 + (w_1 + w_2)\, dt)\, dt + V_3 \cos(\theta_0 + (w_1 + w_2 + w_3)\, dt)\, dt \right)^2$$

$$+ 2 V_1 V_2 \cos(\theta_0 + w_1 dt) \cos(\theta_0 + (w_1 + w_2)\, dt)\, dt^2 + 2 V_1 V_3 \cos(\theta_0 + w_1 dt) \cos(\theta_0 + (w_1 + w_2 + w_3)\, dt)\, dt^2$$

$$+ V_1^2 \sin^2(\theta_0 + w_1 dt)\, dt^2 + \left(V_2 \sin(\theta_0 + (w_1 + w_2)\, dt)\, dt + V_3 \sin(\theta_0 + (w_1 + w_2 + w_3)\, dt)\, dt \right)^2$$

$$+ 2 V_1 V_2 \sin(\theta_0 + w_1 dt) \sin(\theta_0 + (w_1 + w_2)\, dt)\, dt^2 + 2 V_1 V_3 \sin(\theta_0 + w_1 dt) \sin(\theta_0 + (w_1 + w_2 + w_3)\, dt)\, dt^2$$

$$= V_1^2 dt^2 + 2 V_1 V_2 dt^2 \cos(\theta_0 + w_1 dt - \theta_0 - w_1 dt - w_2 dt) + 2 V_1 V_3 dt^2 \cos(\theta_0 + w_1 dt - \theta_0 - w_1 dt - w_2 dt - w_3 dt)$$

$$+ \left(V_2 \cos(\theta_0 + (w_1 + w_2)\, dt)\, dt + V_3 \cos(\theta_0 + (w_1 + w_2 + w_3)\, dt)\, dt \right)^2$$

$$+ \left(V_2 \sin(\theta_0 + (w_1 + w_2)\, dt)\, dt + V_3 \sin(\theta_0 + (w_1 + w_2 + w_3)\, dt)\, dt \right)^2$$

$$= V_1^2\,dt^2 + 2V_1V_2\,dt^2\cos(w_2\,dt) + 2V_1V_3\,dt^2\cos((w_2+w_3)\,dt)$$
$$+ V_2^2\cos^2(\theta_0+(w_1+w_2)\,dt)\,dt^2 + V_3^2\,dt^2\cos^2(\theta_0+(w_1+w_2+w_3)dt)$$
$$+ 2V_2V_3\,dt^2\cos(\theta_0+(w_1+w_2)dt)\cos(\theta_0+(w_1+w_2+w_3)\,dt)$$
$$+ V_2^2\sin^2(\theta_0+(w_1+w_2)\,dt)\,dt^2 + V_3^2\sin^2(\theta_0+(w_1+w_2+w_3)\,dt)\,dt^2$$
$$+ 2V_2V_3\,dt^2\sin(\theta_0+(w_1+w_2)\,dt)\sin(\theta_0+(w_1+w_2+w_3)dt)$$

$$= V_1^2\,dt^2 + V_2^2\,dt^2 + V_3^2\,dt^2 + 2V_1V_2\,dt^2\cos(w_2\,dt)$$
$$+ 2V_1V_3\,dt^2\cos((w_2+w_3)\,dt)$$
$$+ 2V_2V_3\,dt^2\cos(\cancel{\theta_0}+w_1\,dt+w_2\,dt-\cancel{\theta_0}-\cancel{w_1\,dt}-\cancel{w_2\,dt}-w_3\,dt)$$

$$= V_1^2\,dt^2 + V_2^2\,dt^2 + V_3^2\,dt^2 + 2V_1V_2\,dt^2\cos(w_2\,dt)$$
$$+ 2V_1V_3\,dt^2\cos((w_2+w_3)\,dt) + 2V_2V_3\,dt^2\cos(w_3\,dt)$$

Linearizing, we get

$$V_{1g}^2\,dt^2 + V_{2g}^2\,dt^2 + V_{3g}^2\,dt^2 + 2V_{1g}V_{2g}\,dt^2\cos(w_{2g}\,dt)$$
$$+ 2V_{1g}V_{3g}\,dt^2\cos((w_{2g}+w_{3g})\,dt) + 2V_{2g}V_{3g}\,dt^2\cos(w_{3g}\,dt)$$
$$+\left(2V_{1g}\,dt^2 + 2V_{2g}\,dt^2\cos(w_{2g}\,dt) + 2V_{3g}\,dt^2\cos((w_{2g}+w_{3g})dt)\right)(V_1-V_{1g})$$
$$+\left(2V_{2g}\,dt^2 + 2V_{1g}\,dt^2\cos(w_{2g}\,dt) + 2V_{3g}\,dt^2\cos(w_{3g}\,dt)\right)(V_2-V_{2g})$$
$$+\left(2V_{3g}\,dt^2 + 2V_{1g}\,dt^2\cos((w_{2g}+w_{3g})dt) + 2V_{2g}\,dt^2\cos(w_{3g}\,dt)\right)(V_3-V_{3g})$$
$$+\left(-2V_{1g}V_{2g}\,dt^3\sin(w_{2g}\,dt) - 2V_{1g}V_{3g}\,dt^3\sin((w_{2g}+w_{3g})\,dt)\right)(w_2-w_{2g})$$
$$+\left(-2V_{1g}V_{3g}\,dt^3\sin((w_{2g}+w_{3g})\,dt) - 2V_{2g}V_{3g}\,dt^3\sin(w_{3g}dt)\right)(w_3-w_{3g})$$

$$= -V_{1g}^2\,dt^2 - V_{2g}^2\,dt^2 - V_{3g}^2\,dt^2 + 2V_{1g}V_{2g}\,dt^2\cos(w_{2g}\,dt)$$
$$+ 2V_{1g}V_{3g}\,dt^2\cos((w_{2g}+w_{3g})\,dt) + 2V_{2g}V_{3g}\,dt^2\cos(w_{3g}\,dt)$$
$$+ 2V_1V_{1g}\,dt^2 + 2V_1V_{2g}\,dt^2\cos(w_{2g}\,dt) + 2V_1V_{3g}\,dt^2\cos((w_{2g}+w_{3g})\,dt)$$
$$- 2V_{1g}V_{2g}\,dt^2\cos(w_{2g}\,dt) - 2V_{1g}V_{3g}\,dt^2\cos((w_{2g}+w_{3g})\,dt)$$
$$+ 2V_2V_{2g}\,dt^2 + 2V_2V_{1g}\,dt^2\cos(w_{2g}\,dt) + 2V_2V_{3g}\,dt^2\cos(w_{3g}\,dt)$$
$$- 2V_{1g}V_{2g}\,dt^2\cos(w_{2g}\,dt) - 2V_{2g}V_{3g}\,dt^2\cos(w_{3g}\,dt)$$
$$+ 2V_3V_{3g}\,dt^2 + 2V_3V_{1g}\,dt^2\cos((w_{2g}+w_{3g})\,dt) + 2V_3V_{2g}\,dt^2\cos(w_{3g}\,dt)$$
$$- 2V_{1g}V_{3g}\,dt^2\cos((w_{2g}+w_{3g})\,dt) - 2V_{2g}V_{3g}\,dt^2\cos(w_{3g}\,dt)$$
$$- 2w_2V_{1g}V_{2g}\,dt^3\sin(w_{2g}\,dt) - 2w_2V_{1g}V_{3g}\,dt^3\sin((w_{2g}+w_{3g})\,dt)$$
$$+ 2w_{2g}V_{1g}V_{2g}\,dt^3\sin(w_{2g}\,dt) + 2w_{2g}V_{1g}V_{3g}\,dt^3\sin((w_{2g}+w_{3g})\,dt)$$
$$- 2w_3V_{1g}V_{3g}\,dt^3\sin((w_{2g}+w_{3g})\,dt) - 2w_3V_{2g}V_{3g}\,dt^3\sin(w_{3g}\,dt)$$
$$+ 2w_{3g}V_{1g}V_{3g}\,dt^3\sin((w_{2g}+w_{3g})\,dt) + 2w_{3g}V_{2g}V_{3g}\,dt^3\sin(w_{3g}\,dt)$$

$$= -V_{1g}^2\,dt^2 - V_{2g}^2\,dt^2 - V_{3g}^2\,dt^2 - 2V_{1g}V_{2g}\,dt^2\cos(w_{2g}\,dt)$$
$$- 2V_{1g}V_{3g}\,dt^2\cos((w_{2g}+w_{3g})\,dt) - 2V_{2g}V_{3g}\,dt^2\cos(w_{3g}\,dt)$$
$$+ 2w_{2g}V_{1g}V_{2g}\,dt^3\sin(w_{2g}\,dt) + 2w_{2g}V_{1g}V_{3g}\,dt^3\sin((w_{2g}+w_{3g})\,dt)$$
$$+ 2w_{3g}V_{1g}V_{3g}\,dt^3\sin((w_{2g}+w_{3g})\,dt) + 2w_{3g}V_{2g}V_{3g}\,dt^3\sin(w_{3g}\,dt)$$
$$+ V_1\left(2V_{1g}\,dt^2 + 2V_{2g}\,dt^2\cos(w_{2g}\,dt) + 2V_{3g}\,dt^2\cos((w_{2g}+w_{3g})\,dt)\right)$$
$$+ V_2\left(2V_{2g}\,dt^2 + 2V_{1g}\,dt^2\cos(w_{2g}\,dt) + 2V_{3g}\,dt^2\cos(w_{3g}\,dt)\right)$$
$$+ V_3\left(2V_{3g}\,dt^2 + 2V_{1g}\,dt^2\cos((w_{2g}+w_{3g})\,dt) + 2V_{2g}\,dt^2\cos(w_{3g}\,dt)\right)$$
$$+ W_2\left(-2V_{1g}V_{2g}\,dt^3\sin(w_{2g}\,dt) - 2V_{1g}V_{3g}\,dt^3\sin((w_{2g}+w_{3g})\,dt)\right)$$

$$+ w_3 \left(-2 v_{1g} v_{3g} dt^3 \sin\left((w_{2g} + w_{3y}) dt \right) - 2 v_{2g} v_{3y} dt^3 \sin\left(w_{3g} dt \right) \right)$$

$$(x_1 - x_g)^2 + (x_2 - x_g)^2 + (x_3 - x_g)^2$$

$$= x_1^2 + x_g^2 - 2x_1x_g + x_2^2 + x_g^2 - 2x_2x_g + x_3^2 + x_g^2 - 2x_3x_g$$

$$= x_1^2 + x_2^2 + x_3^2 + 3x_g^2 - 2(x_1 + x_2 + x_3)x_g$$

CSE471: Advanced Robotics & Control **Spring 2018**

Lecture 7: March 29

Lecturer: *Scribes: Avijit Ashe, Roll:20172067*

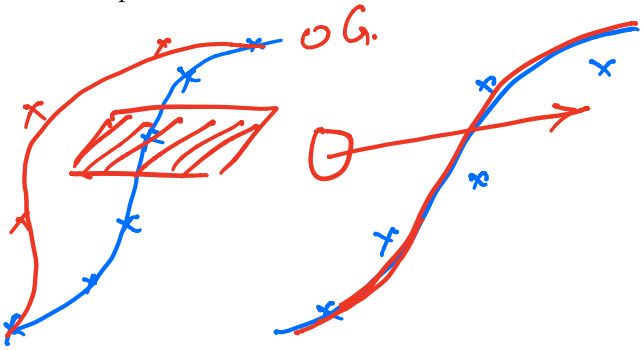
Note: *LaTeX template courtesy of UC Berkeley EECS dept.*
Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.*

7.1 Derivations for the trajectory optimization problem

We consider a **holonomic system**, and it's motion model. That is, a system whose constraints can be expressed in integrable form that equates to zero across all the positional constraints.

7.1.1 Objectives

- 1. Goal reaching
- 2. Obstacle avoidance
- 3. Velocity and acceleration bounds
- 4. Smoothness



We now build on the objectives above and formulate the entire trajectory optimization problem

7.2 Objective: Goal reaching

We need to minimize this :

$$(x_{robo}(t) - x_g)^2 + (y_{robo}(t) - y_g)^2 \tag{7.1}$$

where, the

$$(x_{robo}, y_{robo})$$

is a function of 't' or discrete time steps, and

$$(x_g, y_g)$$

is the goal in world coordinates

Now, if a holonomic motion model is considered i.e. f(n1, n2...; t) = 0, we get the following form

$$x_{robo}(t) = x_{robo}(t-1) + x_{robo}\dot{(}t-1)x\delta t$$

or

$$x_{robo}(n) = x_{robo}(n-1) + x_{robo}\dot{(}n-1)x\delta t$$

where, the robo is at *nth* position at time t given by the physical coordinates *(x(n), y(n))*. The starting point is assumed to be at zero or origin.

The steps are: *(n-1)th* position at time *t-1*, *nth* position at time t, it takes a small time-step to reach from (n-1) to n, and $(\dot{x_n}, \dot{y_n})$ are the velocities at *nth* position

So, the objective is to find **the set** of

$$\dot{x}$$

the velocity component along x and

$$\dot{y}$$

the velocity component along y, that would push the robo from any position in the world towards the goal,

$$(x_g, y_g)$$

.

Only that set minimizes the above equation 7.1

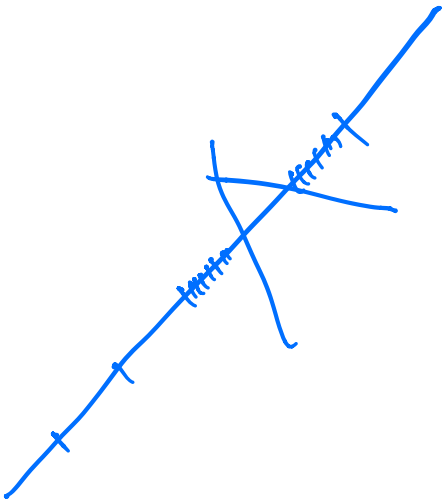
More specifically, we try to solve for a set of $\dot{x_0}, \dot{x_1}, \dot{x_2}, \dot{x_3}, ...\dot{x_{n-1}}$ and $\dot{y_0}, \dot{y_1}, \dot{y_2}, \dot{y_3}, ...\dot{y_{n-1}}$ that would push the robo from any position in the world towards the goal, (x_g, y_g).

Now onwards, (x_{robo}, y_{robo}) is written as (x_r, y_r)

Now, the equation for holonomic model is used below

$$\begin{aligned} x_n &= x_{n-1} + \dot{x_{n-1}}\delta t \\ x_{n-1} &= x_{n-2} + \dot{x_{n-2}}\delta t \\ x_{n-2} &= x_{n-3} + \dot{x_{n-3}}\delta t \\ &... \\ &... \\ &... \end{aligned}$$

$$x_1 = x_0 + \dot{x_0}\delta t$$



That is,

$$\begin{aligned} x_n &= x_0 + (\dot{x_0}, \dot{x_1}, \dot{x_2}, \dot{x_3}, ...\dot{x_{n-1}})\delta t \\ &= x_0 + \sum_{i=0}^{n-1} \dot{x_i}\delta t \end{aligned}$$

and similarly for the y-component like

$$\begin{aligned} y_n &= y_0 + (\dot{y_0}, \dot{y_1}, \dot{y_2}, \dot{y_3}, ...\dot{y_{n-1}})\delta t \\ &= y_0 + \sum_{i=0}^{n-1} \dot{y_i}\delta t \end{aligned}$$

7.2.1 Deriving goal reaching as a quadriatic equation

Let us assume that we have just 3 control points (or way points through which the robo must pass through as close as possible) for simplicity

So, we have $\dot{x_0}, \dot{x_1}, \dot{x_2}$ and $\dot{y_0}, \dot{y_1}, \dot{y_2}$

Here is a small illustration of the same

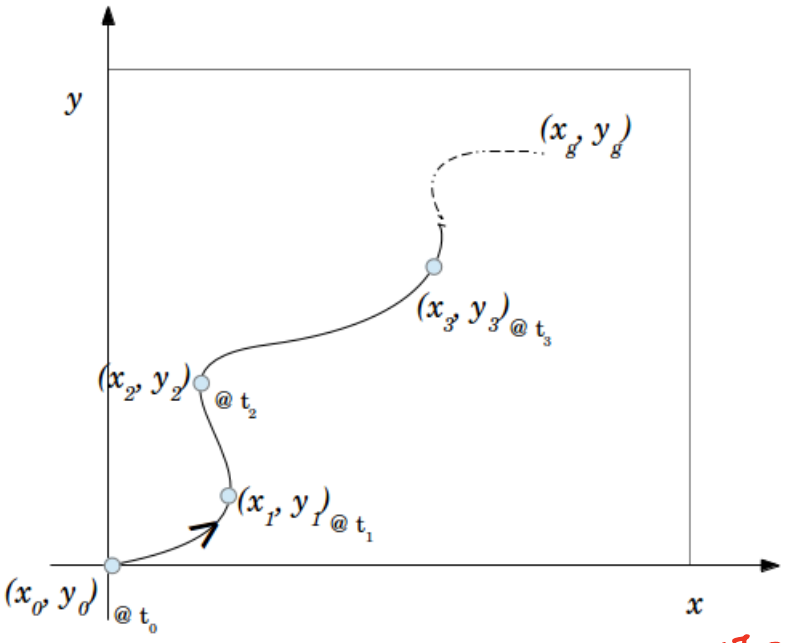


Figure 7.1: Positional constraints on robo trajectory

The cost function from equation 7.1 can be rewritten as

$$(x_3 - x_g)^2 + (y_3 - y_g)^2$$

Upon expanding it using equation 7.2, we get

$$(x_0 + (\dot{x_0} + \dot{x_1} + \dot{x_2})\delta t - x_g)^2 + (y_0 + (\dot{y_0} + \dot{y_1} + \dot{y_2})\delta t - y_g)^2$$

Writing the x-term in quadriatic form, we have

$$(\quad (x_0 - x_g) \quad + \quad (\dot{x_0} + \dot{x_1} + \dot{x_2})\delta t) \quad)^2 \tag{7.2}$$

following $(a+b)^2 = a^2 + b^2 + 2ab$, where $a = (x_0 - x_g)$ and $b = (\dot{x_0} + \dot{x_1} + \dot{x_2})\delta t$. So, we have

$$(x_o - x_g)^2 \quad + \quad (\dot{x_0} + \dot{x_1} + \dot{x_2})^2 \delta t^2 \quad + \quad 2(x_0 - x_g)(\dot{x_0} + \dot{x_1} + \dot{x_2})\delta t$$

7.2.2 Rewriting 1st term from the quadriatic form

$(x_0 - x_g)^2$ is just a constant C1 as they're both known $(x_0, y_0) = (0, 0)$ or the origin and (x_g, y_g) can be some arbitrary values e.g. (10, 10) in units

7.2.3 Rewriting 2nd term from the quadriatic form

Let

$$X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

That is, $\dot{x_0} + \dot{x_1} + \dot{x_2}$ can be written in matrix-vector form like

$$\begin{bmatrix} \dot{x_0} & \dot{x_1} & \dot{x_2} \end{bmatrix} \quad \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

and let's call it K^T

So, $(\dot{x_0} + \dot{x_1} + \dot{x_2})^2 \delta t^2 = K^T K$ from linear algebra, like so

$$K^T K = \begin{bmatrix} \dot{x_0} & \dot{x_1} & \dot{x_2} \end{bmatrix} \quad \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} \dot{x_0} \\ \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} \dot{x_0} & \dot{x_1} & \dot{x_2} \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} \dot{x_0} \\ \dot{x_1} \\ \dot{x_2} \end{bmatrix} \tag{7.3}$$

7.2.4 Rewriting last term from the quadriatic form

$$2(x_0 - x_g)(\dot{x_0} + \dot{x_1} + \dot{x_2})\delta t = 2(x_0 - x_g)\delta t \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} \dot{x_0} \\ \dot{x_1} \\ \dot{x_2} \end{bmatrix} = q_1^T X$$

The final matrix vector form can be rewritten as

$$\boxed{min_{X,Y} \quad X^T A X + q_1^T X + C1 \quad + \quad Y^T B Y + q_2^T Y + C2} \tag{7.4}$$

where,

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, B = A$$

$$q_1 = 2(x_0 - x_g)\delta t \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \quad q_2 = 2(y_0 - y_g)\delta t \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

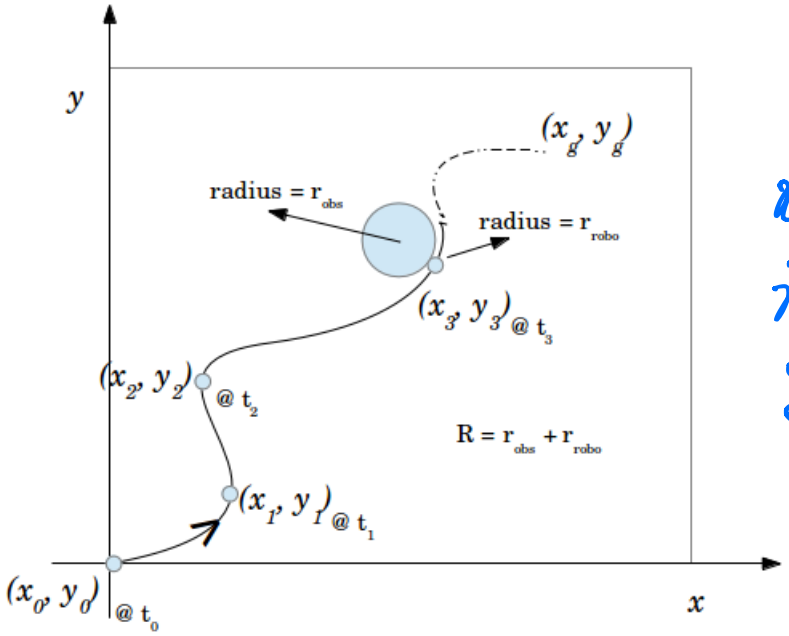
Then $q^T_{1\times 2n} X_{2n\times 1} = [q_{1\,1\times n} \quad q_{2\,1\times n}] \begin{bmatrix} \dot{x}_0 \\ \vdots \\ \dot{x}_n \\ \dot{y}_0 \\ \vdots \\ \dot{y}_n \end{bmatrix}_{2n\times 1}$

$$X = \begin{bmatrix} \dot{x_0} \\ \dot{x_1} \\ \dot{x_2} \\ \vdots \\ \dot{x_{n-1}} \end{bmatrix} \quad , Y = \begin{bmatrix} \dot{y_0} \\ \dot{y_1} \\ \dot{y_2} \\ \vdots \\ \dot{y_{n-1}} \end{bmatrix}$$

7.3 Objective: Obstacle avoidance

If we have an obstacle at (x_ob, y_ob) then an appropriate way to avoid it could be written as

$$(x_n - x_{ob})^2 + (y_n - y_{ob})^2 \geq R^2$$
$$-\left[(x_n - x_{ob})^2 + (y_n - y_{ob})^2 - R^2\right] < 0$$



$\dot{x}_0 \leq \dot{x}_{max}$
$\dot{x}_1 \leq \dot{x}_{max}$
$\dot{y}_1 \leq \dot{y}_{max}$

To get it in the matrix-vector form $AX = B$ we linearize the objective function 7.2 using Taylor's expansion up to 1st order.

We use only 3 control/way points for ease thus framing the equation as

$$-\left[(x_3 - x_{ob})^2 + (y_3 - y_{ob})^2 - R^2\right] < 0$$

expanding the terms using expansion of 7.2 from above quadriatic form

$$-\left[(x_0 + (\dot{x_0} + \dot{x_1} + \dot{x_2})\delta t - x_{ob})^2 + (y_0 + (\dot{y_0} + \dot{y_1} + \dot{y_2})\delta t - y_{ob})^2 - R^2\right] < 0$$
$$-\left[(x_0 - x_{ob} + (\dot{x_0} + \dot{x_1} + \dot{x_2})\delta t)^2 + (y_0 - y_{ob} + (\dot{y_0} + \dot{y_1} + \dot{y_2})\delta t)^2 - R^2\right] < 0$$

Thus, ignoring the -ve sign, R^2 and using just x-component at a time we apply MTS (multivariate taylor series) $(x_0 - x_{ob} + (\dot{x_0} + \dot{x_1} + \dot{x_2})\delta t)^2$

7.3.1 Multivariate Taylor Series

We linearize 7.2 for x-term now as

$$f(\dot{x_0} + \dot{x_1} + \dot{x_2}) = f(\dot{x_0^*} + \dot{x_1^*} + \dot{x_2^*}) + \frac{\delta f}{\delta \dot{x_0}}|_{\dot{x_0},\dot{x_1},\dot{x_2}},(\dot{x_0} - \dot{x_0^*}) + \frac{\delta f}{\delta \dot{x_1}}|_{\dot{x_0},\dot{x_1},\dot{x_2}},(\dot{x_1} - \dot{x_1^*}) + \frac{\delta f}{\delta \dot{x_2}}|_{\dot{x_0},\dot{x_1},\dot{x_2}},(\dot{x_2} - \dot{x_2^*})$$

$$f(\dot{x_0} + \dot{x_1} + \dot{x_2}) = f(\dot{x_0^*} + \dot{x_1^*} + \dot{x_2^*}) + \begin{bmatrix} \frac{\delta f}{\delta \dot{x_0}} & \frac{\delta f}{\delta \dot{x_1}} & \frac{\delta f}{\delta \dot{x_2}} \end{bmatrix}_{\dot{x_0},\dot{x_1},\dot{x_2},} \begin{bmatrix} (\dot{x_0} - \dot{x_0^*}) \\ (\dot{x_1} - \dot{x_1^*}) \\ (\dot{x_2} - \dot{x_2^*}) \end{bmatrix}$$

Now, $f = (x_0 - x_{ob} + (\dot{x_0} + \dot{x_1} + \dot{x_2})\delta t)^2$, So, $\frac{\delta f}{\delta \dot{x_0}} = 2\left((x_0 - x_{ob}) + (\dot{x_0} + \dot{x_1} + \dot{x_2})\delta t\right)\delta t$

$$\begin{aligned} \frac{\delta f}{\delta \dot{x_0}} &= 2\Delta t((x_0 - x_{ob}) + (\dot{x_0} + \dot{x_1} + \dot{x_2})\Delta t) \\ &= 2\Delta t((x_0 - x_{ob}) + 2\Delta t^2 \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \dot{x_0^*} \\ \dot{x_1^*} \\ \dot{x_2^*} \end{bmatrix} \end{aligned}$$

and similarly for

$$\frac{\delta f}{\delta \dot{x_1}} \quad , \frac{\delta f}{\delta \dot{x_2}}$$

Now, considering the quadriatic form of 7.2 $(x_0 - x_{ob} + (\dot{x_0} + \dot{x_1} + \dot{x_2})\delta t)^2 = X^T AX + q_1^T X + C1$, we get

7.3.2 1st term of Taylor

$X_*^T AX_* + q_1^T X_* + C1$ where $X_* = \begin{bmatrix} \dot{x_0^*} \\ \dot{x_1^*} \\ \dot{x_2^*} \end{bmatrix}$

7.3.3 2nd term of Taylor

$$\frac{\delta X_*^T AX_* + q_1^T X_* + C1}{\delta \dot{x_0}} = (2AX_* + 2q_1)^T \begin{bmatrix} (\dot{x_0} - \dot{x_0^*}) \\ (\dot{x_1} - \dot{x_1^*}) \\ (\dot{x_2} - \dot{x_2^*}) \end{bmatrix}$$

Finally, we have

$$-\left[(x_0 - x_{ob} + (\dot{x_0} + \dot{x_1} + \dot{x_2})\delta t)^2 + (y_0 - y_{ob} + (\dot{y_0} + \dot{y_1} + \dot{y_2})\delta t)^2 - R^2\right] < 0$$

in quadriatic form is

$$\boxed{-\left[X^T AX + q_1^T X + C1 \quad + \quad X^T BY + q_2^T Y + C2\right]} \tag{7.5}$$

where,

$$A = \Delta t^2 \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, B = A$$

$$q_1 = 2(x_0 - x_{ob})\delta t \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \quad q_2 = 2(y_0 - y_{ob})\delta t \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

$$X = \begin{bmatrix} \dot{x_0} \\ \dot{x_1} \\ \dot{x_2} \end{bmatrix} \quad , Y = \begin{bmatrix} \dot{y_0} \\ \dot{y_1} \\ \dot{y_2} \end{bmatrix}$$

such that after linearization it becomes

$-\left[(X^T AX + q_1^T X + C1 + Y^T BY + q_2^T Y + C2)|_{X^*,Y^*} + (2AX + 2q_1)^T (X - X^*)|_{X^*} + (2BY + 2q_2)^T (Y - Y^*)|_{Y^*}\right] - R^2] < 0$

7.4 Objective with constraints

Finally, we get the objectives for goal reaching and obstacle avoidance together into a convex optimization equation

$$-\left[X^T AX + q_1^T X + C1 \quad + \quad X^T BY + q_2^T Y + C2\right] \tag{7.6}$$

such that

$$-\left[Constant + A_{matrix}(X - X^*) + B_{matrix}(Y - Y^*) - R^2\right] < 0$$

where,

$$A_{matrix} = (2AX + 2q_1)^T|_{X^*}, \quad A = \Delta t^2 \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}. \quad q_1 = 2(x_0 - x_{ob})\delta t \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

References

[ML13] Figures 7.1 to 7.4 generated in Matlab

Min $(x_n - x_g)^2 + (y_n - y_g)^2$

sub to $x_n = x_{n-1} + \dot{x}_{n-1}\delta t$; $y_n = y_{n-1} + \dot{y}_{n-1}\delta t$

$\vdots$ $\vdots$

$x_1 = x_0 + \dot{x}_0 \delta t$; $y_1 = y_0 + \dot{y}_0 \delta t$

$(x_n - x_g)^2$ can be written as

$\left((x_0 - x_g) + (\dot{x}_0 + \dot{x}_1 + \cdots + \dot{x}_{n-1})\delta t\right)^2 +$

$\left((y_0 - y_f) + (\dot{y}_0 + \dot{y}_1 + \cdots + \dot{y}_{n-1})\delta t\right)^2$.

This will have terms like:

$(\dot{x}_0 + \dot{x}_1 + \cdots + \dot{x}_{n-1})^2 \delta t^2$ & $(\dot{y}_0 + \dot{y}_1 + \cdots + \dot{y}_{n-1})^2 \delta t^2$.

$$\underset{1\times n}{\begin{bmatrix} \dot{x}_0 & \cdots & \dot{x}_{n-1} \end{bmatrix}} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \vdots & & & \\ 1 & \cdots & & 1 \end{bmatrix}_{n\times n} \begin{bmatrix} \dot{x}_0 \\ \vdots \\ \dot{x}_{n-1} \end{bmatrix}_{n\times 1}$$

$= X^T A X$. & similarly for the $\dot{y}$'s as $Y^T B Y$

Then $X^T AX + Y^T BY$ is nothing but.

$$\underset{1\times 2n}{\begin{bmatrix} X^T & Y^T \end{bmatrix}} \begin{bmatrix} A_{n\times n} & O_{n\times n} \\ O_{n\times n} & B_{n\times n} \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix}_{2n\times 1}$$

$$= \underset{(1\times 2n)}{X^T} \; P_{2n\times 2n} \; X_{2n\times 1}$$

and from before $q^T X$ is of the form

$$\begin{bmatrix} q_1^T{}_{1\times n} & q_2^T{}_{1\times n} \end{bmatrix} X_{2n\times 1}$$

The above is the optimal control formulation for goal reaching cost with holonomic constraints where control variable are velocities.

HW: Derive for control as acceleration or double integrator system. with velocity bounds

$$\dot{x}_m \le \dot{x}_1 \le \dot{x}_M$$
$$\dot{y}_m \le \dot{y}_1 \le \dot{y}_M$$
$$\vdots$$
$$\dot{x}_m \le \dot{x}_n \le \dot{x}_M$$
$$\dot{y}_m \le \dot{y}_n \le \dot{y}_M$$

The formulation $X^T P X + q^T X$ s.t $GX \leq h$.
is an optimization problem or optimal control problem.

→ solve for X the control variable.
→ forward evolve by this control for dt
→ resolve the optimization problem from the current state obtained after forward evolution.
→ Repeat till goal state is reached.

↓

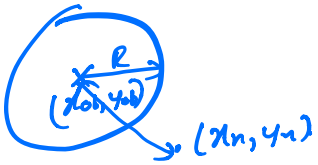
MPC.

Obstacle avoidance constraint:

for time instant n:

$(x_n - x_{ob})^2 + (y_n - y_{ob})^2 > R^2 \longrightarrow (1)$.

R: robot + obstacle size approximated as a circle



$\Rightarrow -[(x_n - x_{ob})^2 + (y_n - y_{ob})^2 - R^2] < 0 \longrightarrow (2)$.

Consider $n = 3$

$x_3 = x_0 + (\dot{x}_0 + \dot{x}_1 + \dot{x}_2)\delta t$, Similarly for $y_3 \longrightarrow (3)$.

To get it of the form $Ax \le h$, which is convex, we need to linearize (3) about a certain $\hat{x}_0, \hat{\dot{x}}_1, \hat{\dot{x}}_2$

$$(x_3 - x_{obt})^2 = [x_0 + (\dot{x}_1 + \dot{x}_2 + \dot{x}_3)\delta t - x_{obt}]^2.$$

Linearize about $\hat{\dot{x}}_1, \hat{\dot{x}}_2, \hat{\dot{x}}_3$ to get.

$$(x_3 - x_{obt})^2 = (x_0 + (\hat{\dot{x}}_1 + \hat{\dot{x}}_2 + \hat{\dot{x}}_3)\delta t - x_{obt})^2.$$

$$+ 2[(x_0 - x_{obt}) + (\hat{\dot{x}}_1 + \hat{\dot{x}}_2 + \hat{\dot{x}}_3)\delta t]\delta t$$

$$[(\dot{x}_0 - \hat{\dot{x}}_0) + (\dot{x}_1 - \hat{\dot{x}}_1) + (\dot{y}_1 - \hat{\dot{y}}_1)] \longrightarrow (4).$$

$$= C_{1x} - 2[(x_0 - x_{obt})\delta t + (\hat{\dot{x}}_1 + \hat{\dot{x}}_2 + \hat{\dot{x}}_3)\delta t^2].[\hat{\dot{x}}_0 + \hat{\dot{x}}_1 + \hat{\dot{x}}_2]$$

$$+ \cdot 2[(x_0 - x_{obt})\delta t + (\hat{\dot{x}}_1 + \hat{\dot{x}}_2 + \hat{\dot{x}}_3)\delta t^2][\dot{x}_0 + \dot{x}_1 + \dot{x}_2]$$

$$= C_{1x} + C_{2x} + [q_{1x}\ q_{1x}\ q_{1x}] \begin{bmatrix} \dot{x}_0 \\ \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} \longrightarrow (5).$$

$$C_{1x} = (x_0 + (\dot{x}_1 + \dot{x}_2 + \dot{x}_3)\delta t - x_{obt})^2 \longrightarrow (6)$$

$$C_{2x} = -2[(x_0 - x_{obt})\delta t + (\hat{\dot{x}}_1 + \hat{\dot{x}}_2 + \hat{\dot{x}}_2)\delta t^2].$$
$$[\hat{\dot{x}}_0 + \hat{\dot{x}}_1 + \hat{\dot{x}}_2] \longrightarrow (7).$$

$$q_{1x} = 2[(x_0 - x_{ob})\delta t + (\hat{\dot{x}}_1 + \hat{\dot{x}}_2 + \hat{\dot{x}}_3)\delta t^2]$$
$$\longrightarrow (8)$$

Similarly get C_{1y}, C_{2y}, q_{1y}

We need to do this for all $x_1, \ldots, x_n$ where $x_i = [x_i\ y_i]^T$

We need to stack together terms to get it of the form $Ax \leq b$.

Generalizing the Obstacle Avoidance Constraint over n time steps into the future:

From (6) we get for $n=3$

$$C_{1x} = C_{31x} \text{ (new notation)} = (x_0 - x_{0t} + (\dot{x}_1 + \dot{x}_2 + \dot{x}_3)\delta t)^2 \longrightarrow (9)$$

$$C_{31y} = (y_0 - y_{0t} + (\dot{y}_1 + \dot{y}_2 + \dot{y}_3)\delta t)^2 \longrightarrow (10)$$

$$C_{31} = C_{31x} + C_{31y} \longrightarrow (11)$$

C_{2x} or C_{32x} (new notations)

$$= -2\left[(\hat{\dot{x}}_1 + \hat{\dot{x}}_2 + \hat{\dot{x}}_3)\left[(x_0 - x_{0t})\delta t + (\hat{\dot{x}}_1 + \hat{\dot{x}}_2 + \hat{\dot{x}}_3)\delta t^2\right]\right. \longrightarrow (12)$$

and similarly

$$C_{32y} = -2\left[(\hat{\dot{y}}_1 + \hat{\dot{y}}_2 + \hat{\dot{y}}_3)\left[(y_0 - y_{0t})\delta t + (\hat{\dot{y}}_1 + \hat{\dot{y}}_2 + \hat{\dot{y}}_3)\delta t^2\right]\right. \longrightarrow (13)$$

$$C_{32} = C_{32x} + C_{32y} \longrightarrow (14)$$

$$q_{3x} = 2((x_0 - x_{0t})\delta t + (\hat{\dot{x}}_1 + \hat{\dot{x}}_2 + \hat{\dot{x}}_3)\delta t^2) \longrightarrow (15)$$

$$q_{3y} = 2[(y_0 - y_{0t})\delta t + (\hat{\dot{y}}_1 + \hat{\dot{y}}_2 + \hat{\dot{y}}_3)\delta t^2] \longrightarrow (16).$$

This easily generalizes over a time horizon.

For example $t = n$ we have

$$C_{n1} = [(x_0 - x_{0t}) + (\hat{\dot{x}}_1 + \hat{\dot{x}}_2 + \cdots + \hat{\dot{x}}_n)\delta t)^2 + [(y_0 - y_{0t}) + (\hat{\dot{y}}_1 + \hat{\dot{y}}_2 + \cdots + \hat{\dot{y}}_n)\delta t]^2 \longrightarrow (17)$$

$$C_{n2} = -2\left[(\hat{x}_1+\hat{x}_2+\cdots+\hat{x}_n)\right]\left[(x_0-x_{ot})\delta t + (\hat{x}_1+\hat{x}_2+\cdots+\hat{x}_n)\delta t^2\right] + \left[(\hat{y}_1+\hat{y}_2+\cdots+\hat{y}_n)\left[(y_0-y_{ot})\delta t + (\hat{y}_1+\hat{y}_2+\cdots+\hat{y}_n)\delta t^2\right]\right. \longrightarrow (18)$$

$$q_{nx} = 2\left([x_0-x_{ot}]\delta t + (\hat{x}_1+\hat{x}_2+\cdots+\hat{x}_n)\delta t^2\right) \longrightarrow (19)$$

$$q_{ny} = 2\left[(y_0-y_{ot})\delta t + (\hat{y}_1+\hat{y}_2+\cdots+\hat{y}_n)\delta t^2\right) \longrightarrow (20)$$

Then the obstacle avoidance constraint at nth time step is

$$-\left[C_{n1}+C_{n2}+\left[Q_{nx_{(1,n)}}\ Q_{ny_{(1,n)}}\right]\begin{bmatrix}\dot{x}_1\\ \vdots\\ \dot{x}_n\\ \dot{y}_1\\ \vdots\\ \dot{y}_n\end{bmatrix}\right] \angle R^2 \longrightarrow (21)$$

where

$$Q_{nx} = [q_{nx}\ \cdots\ q_{nx}]_{(1,n)} \longrightarrow (22)$$

$$Q_{ny} = [q_{ny}\ \cdots\ q_{ny}]_{(1,n)} \longrightarrow (23)$$

Then the overall set of inequality constraints for a single obstacle is

$$-\begin{bmatrix} q_{1x} & \overbrace{0_2\cdots 0_n}^{n-1\text{ times}} & q_{1y} & \overbrace{0\cdots 0}^{n-1\text{ times}} \\ q_{2x} & q_{2x}\ \overbrace{0\cdots 0}^{n-2} & q_{2y} & q_{2y}\ \overbrace{0\cdots 0}^{n-2} \\ q_{3x} & q_{3x}\ q_{3x}\cdots 0 & q_{3y} & q_{3y}\ q_{3y}\cdots 0 \\ \vdots & & & \\ q_{nx} & q_{nx}\cdots q_{nx} & q_{ny} & \cdots\ q_{ny} \end{bmatrix} \begin{bmatrix}\dot{x}_1\\ \vdots\\ \dot{x}_n\\ \dot{y}_1\\ \vdots\\ \dot{y}_n\end{bmatrix} \angle \begin{bmatrix} C_{n1}+C_{n2}\delta t^2 \\ C_{21}+C_{22}\delta t^2 \\ \vdots \\ C_{n1}+C_{n2}\delta t^2 \end{bmatrix} \longrightarrow (24)$$

$$A \qquad x \angle b$$

The matrix G is $n \times 2n$ for a time horizon of n as there are n obstacle avoidance constraint at each of these n time steps.

If there are m obstacles then we get a matrix G of size $nm \times 2n$

Incorporating trust region constraint

$\dot{x}_1 \leq \hat{\dot{x}}_1 + \delta x \rightarrow (1)$. $\quad \dot{x}_1 \leq \hat{\dot{x}}_1 + \delta x \rightarrow (3)$.

$\dot{x}_1 \geq \hat{\dot{x}}_1 - \delta x \rightarrow (2)$ $\quad -\dot{x}_1 \leq \delta x - \hat{\dot{x}}_1 \rightarrow (4)$.

Hence we get a set of constraints of the form

$\dot{x}_1 \leq \hat{\dot{x}}_1 + \delta x \rightarrow (1)$.

$-\dot{x}_1 \leq \delta x - \hat{\dot{x}}_1 \rightarrow (2)$

$\dot{x}_2 \leq \hat{\dot{x}}_2 + \delta x \rightarrow (3)$.

$-\dot{x}_2 \leq \delta x - \hat{\dot{x}}_2 \rightarrow (4)$.

$\vdots$

$\dot{x}_n \leq \hat{\dot{x}}_n + \delta x \rightarrow (5)$

$-\dot{x}_n \leq \delta x - \hat{\dot{x}}_n \rightarrow (6)$

and similarly for $\dot{y}_1 \ldots \dot{y}_n$

Then we get the new trust region matrix G_{trust}, where $G_{trust}\, x \leq h_{trust} \rightarrow (7)$

where G_{trust} =

$$\begin{bmatrix} \overbrace{1\ 0\ 0\ \cdots\ 0}^{n} & \overbrace{0\ 0\ \cdots\ 0}^{n} \\ -1\ 0\ 0\ \cdots\ 0 & 0\ 0\ \cdots\ 0 \\ 0\ 1\ 0\ \cdots\ 0 & 0\ 0\ 0\ \cdots\ 0 \\ 0\ -1\ 0\ \cdots\ 0 & 0\ 0\ 0\ \cdots\ 0 \\ \vdots & \\ 0\ 0\ \cdots\ 0 & 1\ 0\ \cdots\ 0 \\ 0\ 0\ \cdots\ 0 & -1\ 0\ \cdots\ 0 \\ 0\ 0\ \cdots\ 0 & 0\ 1\ \cdots\ 0 \\ 0\ 0\ \cdots\ 0 & 0\ -1\ \cdots\ 0 \\ \vdots & \\ 0\ \cdots\ 0 & 0\ 0\ \cdots\ 1 \\ 0\ \cdots\ 0 & 0\ 0\ \cdots\ -1 \end{bmatrix}_{(4n,2n)} \begin{bmatrix} \dot{x}_1 \\ \vdots \\ \dot{x}_n \\ \dot{y}_1 \\ \vdots \\ \dot{y}_n \end{bmatrix} \leq \begin{bmatrix} \dot{x}_1 + \delta x \\ \delta x - \dot{x}_1 \\ \dot{x}_2 + \delta x \\ \delta x - \dot{x}_2 \\ \vdots \\ \dot{x}_n + \delta x \\ \delta x - \dot{x}_n \\ \dot{y}_1 + \delta y \\ \delta y - \dot{y}_1 \\ \vdots \\ \dot{y}_n + \delta y \\ \delta y - \dot{y}_n \end{bmatrix}$$

We can then stack the trust region matrix or constraint with the G matrix above to get one large matrix $G_L x \leq h_L$.

GradCEM

Source: Model-Predictive Control via Cross-Entropy and Gradient-Based Optimization

Cross Entropy Method (CEM) based motion planning algorithm

1. Sample controls from a gaussian distribution.
2. Clip the controls within the control bounds of the agent.
3. Rollout the trajectories.
4. Score the trajectories.
5. Select the elite trajectories.
6. Set the mean and covariance of the gaussian distribution as the mean and covariance of the elite trajectories.
7. Goto step 1.

CEM: Advantages and Disadvantages

Advantages:

1. Exploratory in nature.
2. Works with non-smooth cost functions.

Disadvantages:

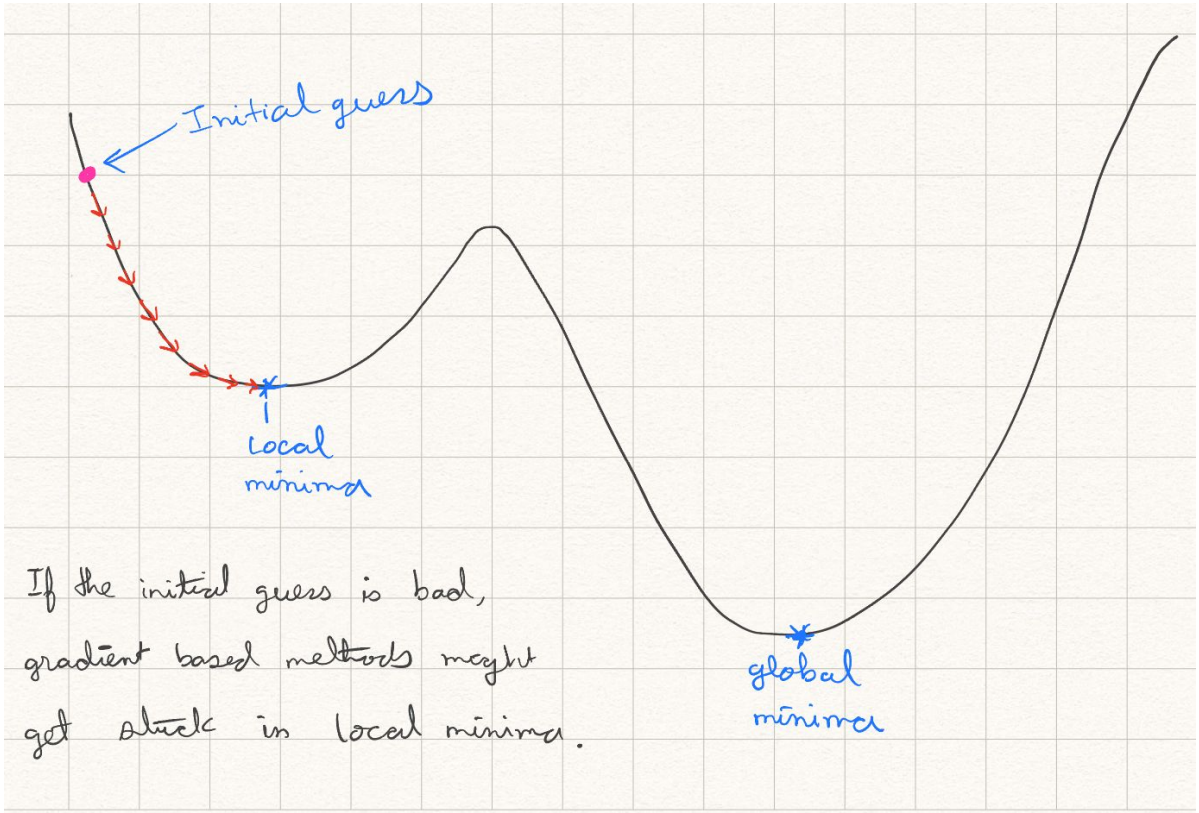
1. Computational time increases as the dimensionality of the action space increase.

Gradient based motion planning algorithm

1. Involves backpropagating derivatives of the scoring function with respect to actions for updating the sequence of actions iteratively through gradient descent.

$$\bar{a}_{0:H}^{(k)} := \bar{a}_{0:H}^{(k-1)} + \alpha \nabla_{\bar{a}} \bar{R}(\bar{a}_{0:H}^{(k-1)}, \bar{s}_{0:H}^{(k-1)}), \qquad k = 1, 2, ..., K$$

2. **Advantages:** They tend to be very fast even when the action dimension increases.
3. **Disadvantages:**
   a. Gradient based approaches like sgd need a good starting guess.
   b. The scoring functions needs to be differentiable.
   c. They tend to converge to a local-minima.



If the initial guess is bad, gradient based methods might get stuck in local minima.

How do we combine the advantages of both the approaches?


GradCEM
imgflip.com

GradCEM based motion planning algorithm

1. Sample controls from a gaussian distribution.
2. Clip the controls within the control bounds of the agent.
3. Rollout the trajectories.
4. Score the trajectories.
5. Compute gradients of the scoring function wrt the controls
6. Perform gradient descent
7. Select the elite trajectories.
8. Set the mean and covariance of the gaussian distribution as the mean and covariance of the elite trajectories.
9. Goto step 1.


ENOUGH TALK
LET'S CODE


CEM/MPPI
GradCEM
WHAT TO CHOOSE?
JAKE-CLARK.TUMBLR
imgflip.com


Thank you :)

VISUAL SERVO CONTROL

Robotics Planning and Navigation

What is Visual Servo?

- Visual servo (VS) control refers to the use of **computer vision data** to **control the motion** of a robot.
- Vision data can we acquired from:
  - Camera mounted on a manipulator.
  - Camera fixed in a scene.

In this lecture we will focus primarily on the former, so-called eye-in-hand case.

Basic Components

The aim of all visual servo algorithms is to minimize error *e(t)* defined by

$$\boldsymbol{e(t) = s(m(t), a) - s^{*}}$$

m(t) - image measurements

a - additional camera information.

s(m(t); a) – visual features

*s** - the desired values of the features

Classification of Visual Servo

1. IBVS (Image-based visual servoing)
   - Error is computed directly on the values of the features extracted on the 2D image plane, without going through a 3D reconstruction.
   - ***s*** consists of a set of features that are immediately available in the image
2. PBVS (Position-based visual servoing)
   - Information extracted from images (features) is used to reconstruct the current 3D pose (pose/orientation) of an object.
   - ***s*** consists of a pose, which must be estimated from image measurements.

Velocity Controller

$$\boldsymbol{s' = L_s v_c}$$

$\boldsymbol{v}_c = (v_c, w_c)$ - the spatial velocity of the camera.

s' - time variation of the features.

L_s - Interaction Matrix

Interaction Matrix calculation (IBVS)

Projection of 3D world-point to image plane with normalized coordinates:

$$\begin{cases} x &= X/Z = (u - c_u)/f\alpha \\ y &= Y/Z = (v - c_v)/f \end{cases} \quad \text{eq.1}$$

***X** = (X; Y; Z)* – 3D world point,

***x** = (x; y)* – normalized image coordinates

(u, v) - coordinates of the image point expressed in pixel units

(c_u, c_v) - coordinates of the principal point,

f - focal length

α - ratio of the pixel dimensions.

Camera velocity: $\boldsymbol{v_c} = (v_x; v_y; v_z)$ and $\boldsymbol{w_c} = (w_x, w_y, w_z)$.

Taking derivative of the projection equation:

$$\begin{cases} \dot{x} &= \dot{X}/Z - X\dot{Z}/Z^2 = (\dot{X} - x\dot{Z})/Z \\ \dot{y} &= \dot{Y}/Z - Y\dot{Z}/Z^2 = (\dot{Y} - y\dot{Z})/Z \end{cases} \quad \text{eq.2}$$

We can relate the velocity of the 3-D point to the camera spatial velocity using the well-known equation:

$$\dot{\boldsymbol{X}} = -\boldsymbol{v}_c - \omega_c \times \boldsymbol{X} \Leftrightarrow \begin{cases} \dot{X} = -v_x - \omega_y Z + \omega_z Y \\ \dot{Y} = -v_y - \omega_z X + \omega_x Z \\ \dot{Z} = -v_z - \omega_x Y + \omega_y X \end{cases} \quad \text{eq.3}$$

Inserting eq.3 into eq.2, and grouping terms.

$$\begin{cases} \dot{x} = -v_x/Z + xv_z/Z + xy\omega_x - (1 + x^2)\omega_y + y\omega_z \\ \dot{y} = -v_y/Z + yv_z/Z + (1 + y^2)\omega_x - xy\omega_y - x\omega_z \end{cases}$$

Which can be written as: $\dot{\boldsymbol{x}} = \mathbf{L}_{\boldsymbol{x}} \boldsymbol{v}_c$

$$\mathbf{L}_{\boldsymbol{x}} = \begin{pmatrix} -1/Z & 0 & x/Z & xy & -(1 + x^2) & y \\ 0 & -1/Z & y/Z & 1 + y^2 & -xy & -x \end{pmatrix}$$

Interaction Matrix calculation (IBVS)

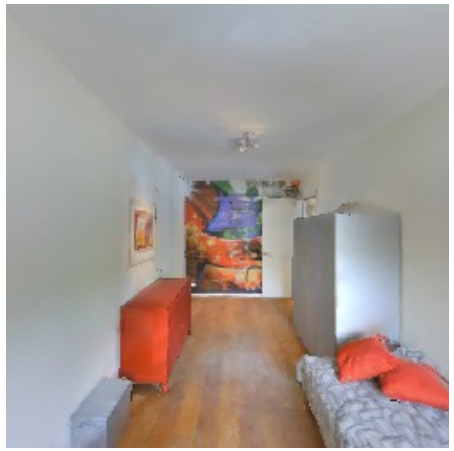
In the matrix L_x, the value Z is the depth of the point relative to the camera frame. Therefore, any control scheme that uses this form of the interaction matrix must estimate or approximate the value of **Z.**

Similarly, the camera intrinsic parameters are involved in the computation of x and y.

$$\mathbf{L}_{\boldsymbol{x}} = \begin{pmatrix} -1/Z & 0 & x/Z & xy & -(1+x^2) & y \\ 0 & -1/Z & y/Z & 1+y^2 & -xy & -x \end{pmatrix}$$

Example of IBVS

Initial Image



Target Image