# Assignment 1- Report

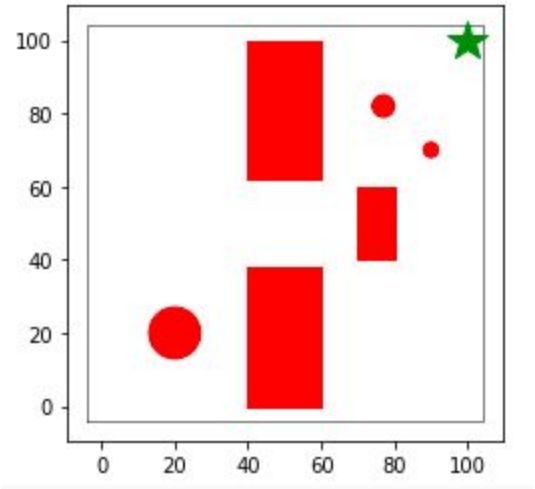
Avinash Prabhu- 2018102027
Dipanwita Guhathakurta- 2018112004

**Link to Videos**-
https://drive.google.com/drive/folders/1_n2JNXdeTra6pv10C2WkjxNGkfpLvmM3?usp=sharing

**Github Repo-**

https://github.com/susiejojo/planning_RRT

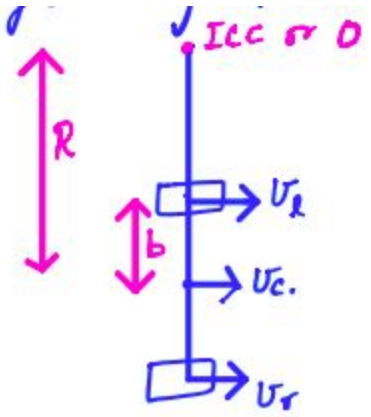
## Obstacle Map used




For the obstacle map, we used a 100x100 map with rectangles and circles as obstacles. For our robot, we used a circular robot of radius 2. Thus, to make our obstacle space a configuration space, we increased the side of all obstacles by 2*2, which is clearly visible in the final graph once RRT has finished running.

## Non- Holonomic Robot

For the non-holonomic case, we have used a differential drive robot with 2 wheels.




**Kinematics of the Robot-**

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

$$y(t) = \int v(t)cos(\theta(t))\,dt$$

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

**Algorithm Used for Non-holonomic RRT**

1. Randomly sample $x$,$y$,$\theta$ within a given limit, let us denote this as $x_{rand}, y_{rand}, \theta_{rand}$ .
2. Find the nearest node to this randomly sampled node which already exists in the graph, let us denote this as $x_{near}, y_{near}, \theta_{near}$
3. From this nearest node, we carry out a set of predefined controls. It is carried out as follows-

For $v, w$ in controls:

$$x_{new} = x_{near} + v * dt * cos(\theta_{near} + w * dt)$$

$$y_{new} = y_{near} + v * dt * sin(\theta_{near} + w * dt)$$

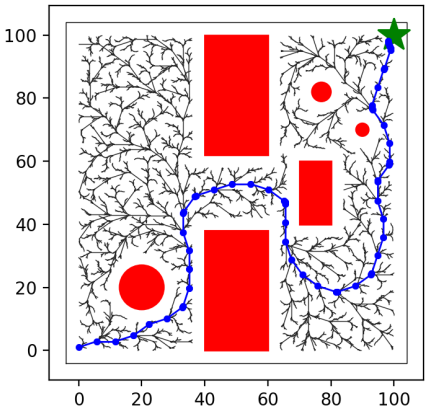
$$\theta_{new} = theta_{near} + w * dt$$

4. Out of all the new points that we generate, pick the one which is closest to the randomly generated point we generated at step 1.
    a. We then check if the point lies within an obstacle.
    b. Next, we check if the edge joining the new point and the previous nearest node is lesser than a certain threshold. If not, we choose another point along the same direction but within a certain threshold.
    c. We then check if the edge joining the new point and the previous point lies within any obstacle. If not, we continue.
5. If all the above points are satisfied, we add the node to the graph and repeat steps 1-4 till we reach a node which is within the vicinity of the goal.

**Finding the optimal Path to the goal**

1. We start from the node in the graph which is closest to the goal node and push it into the final path's list.
2. We iteratively take the parent of the current node we are on and push it into the final path's list.
3. We carry this out until we reach the starting node.

## Sample Output



**Modelling the wheel trajectories**

1. We now have the platform velocities and the  w and $\theta$ at every point on the final path.
2. We can now model the center platform positions as follows:

$$\theta_{current} = \theta_{previous} + w_{current} * dt$$

$$centerx_{current} = centerx_{previous} + v_{current} * dt * cos(\theta_{current})$$

$$centery_{current} = centery_{previous} + v_{current} * dt * sin(\theta_{current})$$

3. Given the center platform positions, we can go ahead calculate the wheel positions as follows

$V_c, \omega \rightarrow$ centre of platform velocities.

$V_l = \omega(R-b)$

$V_R = \omega(R+b)$

$V_c = R\omega$

$\therefore V_l = V_c - \omega b \qquad V_R = V_c + \omega b$

Initial values

$\theta_0 = 0$

$\therefore \theta(t) = \omega t$

$y_l = b$

$y_r = -b$.

$$x_l = \int_0^t V_l \cos(\theta(t))\,dt = \int_0^t (V_c - \omega b)\cos(\omega t)\,dt$$

$$= \int_0^t V_c \cos(\omega t)\,dt - \int_0^t \omega b \cos(\omega t)\,dt$$

$$= x_c - b \cancel{\omega} \cdot \frac{\sin(\omega t)}{\cancel{\omega}}\Big|_0^t$$

$$= x_c - b\sin(\omega t)$$

$$= x_c - b\sin(\theta(t))$$

$$y_l = \int_0^t V_l \sin(\theta(t))\,dt = \int_0^t (V_c - \omega b)\sin(\omega t)\,dt + b$$

$$= \int_0^t V_c \sin(\omega t)\,dt - \int_0^t \omega b \sin(\omega t)\,dt + b$$

$$= y_c + b \cdot \cancel{\omega} \cdot \frac{\cos\omega t}{\cancel{\omega}}\Big|_0^t + b$$

$$= y_c + b\cos(\omega t) - \cancel{b} + \cancel{b}$$

$$= y_c + b\cos(\theta(t))$$

$$x_r = \int_0^t V_r \cos(\theta(t))\,dt = \int_0^t (V_c + \omega b)\cos(\omega t)\,dt$$

$$= \int_0^t V_c \cos(\omega t)\,dt + b\int_0^t \omega \cos(\omega t)\,dt$$

$$= x_c + b \cdot \cancel{\omega} \cdot \frac{\sin(\omega t)}{\cancel{\omega}}\Big|_0^t$$

$$= x_c + b\sin(\omega t) = x_c + b\sin(\theta(t))$$

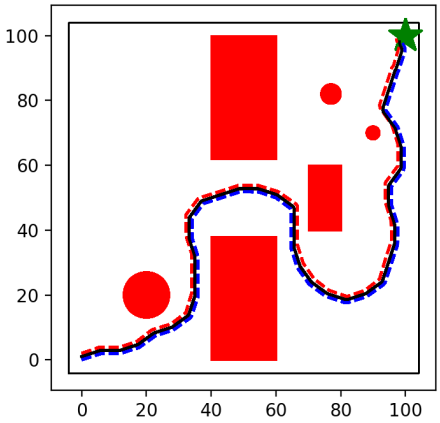
$$y_r = \int_0^t V_r \sin(\theta(t))\,dt - b$$

$$= \int_0^t (V_c + \omega b)\sin(\omega t)\,dt - b$$

$$= y_c + b \cdot \cancel{\omega} \cdot \frac{\cos(\omega t)}{\cancel{\omega}}\Big|_0^t - b$$

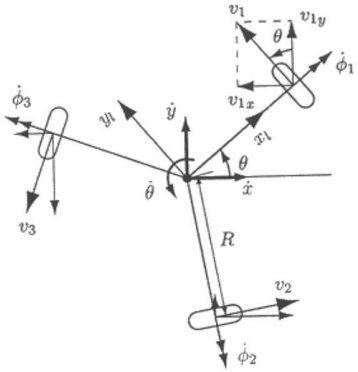
$$= y_c - b\cos(\theta(t)) + \cancel{b} - \cancel{b} = y_c - b\cos\theta(t)$$

## Sample Output




# Holonomic Robot

For the holonomic robot case, we used an omnidirectional robot with 3 degrees of freedom along $x$,$y$,$\theta$.




$$
\begin{aligned}
\dot{\phi}_1 &= (-\sin(\theta)\cos(\theta)\dot{x}_L + \cos^2(\theta)\dot{y}_L + R\dot{\theta})/r \\
\dot{\phi}_2 &= (-\sin(\theta+\alpha_2)\cos(\theta)\dot{x}_L + \cos(\theta+\alpha_2)\cos(\theta)\dot{y}_L + R\dot{\theta})/r \\
\dot{\phi}_3 &= (-\sin(\theta+\alpha_3)\cos(\theta)\dot{x}_L + \cos(\theta+\alpha_3)\cos(\theta)\dot{y}_L + R\dot{\theta})/r
\end{aligned}
$$

## Algorithm Used for Holonomic RRT

1. Randomly sample $x,y,\theta$ within a given limit, let us denote this as $x_{rand}, y_{rand}, \theta_{rand}$.

2. We then check if this new random point lies within some obstacle, if not, we continue with the algorithm.

3. Find the nearest node to this randomly sampled node which already exists in the graph, let us denote this as $x_{near}, y_{near}, \theta_{near}$

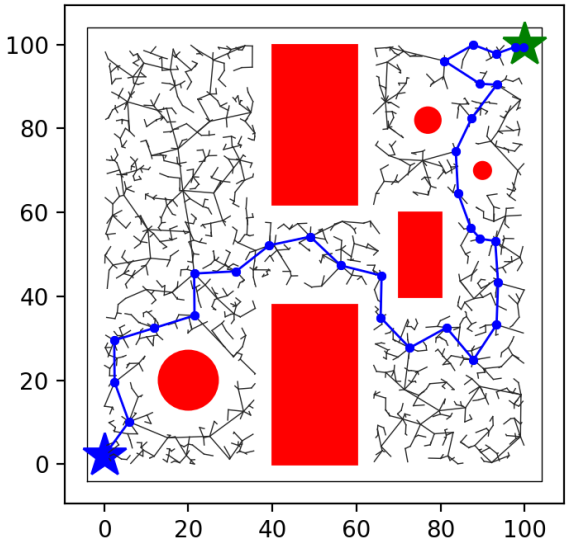
4. Next, we check if the edge joining the new random point and the previous nearest node is lesser than a certain threshold. If not, we choose another point along the same direction but within a certain threshold.

5. We then check if the edge joining the new point and the previous point lies within any obstacle. If not, we continue.
6. If all the above points are satisfied, we add the node to the graph and repeat steps 1-5 till we reach a node which is within the vicinity of the goal.

## Finding the optimal Path to the goal

1. We start from the node in the graph which is closest to the goal node and push it into the final path's list.
2. We iteratively take the parent of the current node we are on and push it into the final path's list.
3. We carry this out until we reach the starting node.

## Sample Output




## Modelling the wheel trajectories

1. Once we have the platform $x$,$y$,$\theta$, we can model the corresponding wheel positions as follows:

$$wheel1x = centerx + radiusofbot * cos(\theta_{current})$$

$$wheel1y = centery + radiusofbot * sin(\theta_{current})$$

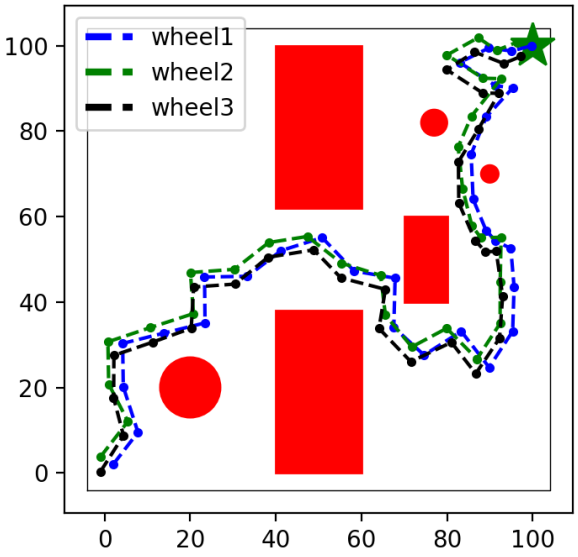
$$wheel2x = centerx + radiusofbot * cos(\theta_{current} + 2 * \pi/3)$$

$$wheel2y = centery + radiusofbot * sin(\theta_{current} + 2 * \pi/3)$$

$$wheel3x = centerx + radiusofbot * cos(\theta_{current} + 4 * \pi/3)$$

$$wheel3y = centery + radiusofbot * sin(\theta_{current} + 4 * \pi/3)$$

## Sample Output




## Generating Videos:

We saved the plots showing the evolution of the RRT graph and the trajectories of the wheels and platform centre in separate directories so that during video generation they can just be stitched together.

1. holonomic_data : contains the RRT tree and platform centre velocity plots for holonomic robot
2. holo_wheel_data: contains the wheel trajectories for holonomic robot
3. nonhn_data: contains the RRT tree and platform centre velocity plots for nonholonomic robot
4. nh_wheel_data: contains the wheel trajectories for non-holonomic robot

## Video Generation

We wrote a bash script mkmovie.sh using ffmpeg which stitches all images from a given directory into a .mp4 file.

- **For wheel trajectory:** We ran ./mkmovie.sh on the wheel_data directories for holonomic and non holonomic robots to obtain the simulation videos at 2 fps.
- **For platform centre trajectory:** We saved all the tree generation/ platform path plotting images in the data directories for holonomic and non holonomic robots. We wrote the script combine.py to take images from a given directory and stitch them together into a single .mp4 file. Python and openCV gave us the freedom here to choose custom fps.
    - For visualising the tree: We ran combine.py on the plots of the RRT tree at 20-25 fps
    - For visualising the path: We ran combine.py on the trajectory plots of the platform centre at 2 fps
    - We wrote a Python script using the moviepy Python library to stitch the above 2 videos into a single final video.

## Work Distribution

1. We both worked equally on report writing
2. Dipanwita worked on modelling positions of wheels for both the holonomic and nonholonomic cases.
3. Avinash worked on modelling the platform positions for both holonomic and nonholonomic cases.
4. Dipanwita worked on creating the videos