# RSS Practical:
# Autonomous Navigation and Inverse Kinematics for Turtlebot 3

s1983630

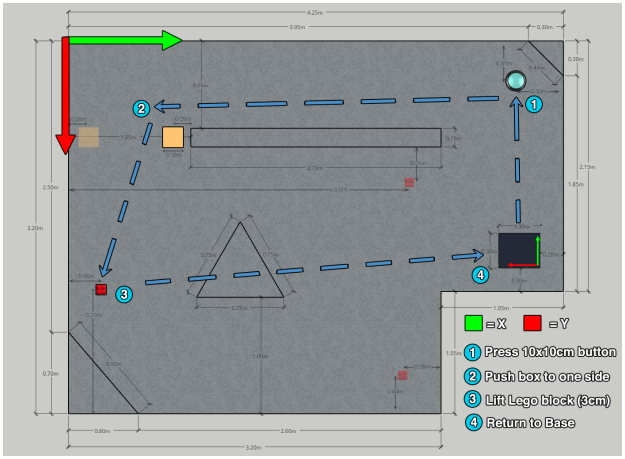


*Figure 1.* Map with measurements and task descriptions for the RSS Practical as attempted by our robot (alternate locations for relevant tasks are shown as faded


## Abstract

This paper details the design and implementation of path-planning, localization, inverse-kinematics, perception and control algorithms on a Turtlebot 3, equipped with Lidar and a 6df robotic arm, so that it may autonomously navigate a known map and perform actions at specified locations. The key design considerations and choices are discussed and justified and an empirical analysis of each components performance is presented. Significant improvements were achieved in the robots performance due to optimization of the localization and path-finding algorithms which resulted in a 396% improvement in run-time efficiency that allowed for higher accuracy and performance overall.


## 1. Introduction

This paper discusses the design and implementation of various algorithms and models to complete the RSS practical challenge (figure 1) using a Turtlebot 3 Waffle-Pi equipped with a PincherX-100 robotic arm.

The components of this challenge highlight several key difficulties in Robotics such as control, perception, navigation and inverse kinematics. In order to design a solution that could work autonomously, cohesively and achieve sufficient performance to complete all tasks, the following system components were developed:

- Path Planning
- Localisation
- Motion controller
- Inverse Kinematics

Section 2 details the design decisions and implementation of the above components and how they were modified to overcome issues related to performance during testing.

Quantitative results from these tests are then presented in section 3. These demonstrate key performance metrics of each subsystem, both in terms of real-world task performance and run-time efficiencies.

These results are then critically analysed with any issues encountered and their potential causes/solutions detailed.

Finally, relevant conclusions drawn from these as well as opportunities for further work are identified section 4.

## 2. Methods

Figure 2 describes the design of the main control loop and the interactions between various subsystems as implemented.

An overall system management script creates instances of the controller and localization objects. The path-planning script is then called for each predetermined task Cartesian co-ordinate and rotation (yaw) to provide a list of points for the controller to navigate to based on an occupancy grid map. This list of intermediate co-ordinates and the final pose are passed to the controller which begins to navigate to each point after it has been transformed into the robot's local co-ordinate frame. The velocity at a given time-step (10Hz) is determined by the Proportional Integral Derivative (PID) controller relative to the error measured between the current state, estimated by a particle filter, and the goal pose. The values for the Proportional, Integral and Derivative error weightings were obtained through experimentation detailed in section 3. Once the absolute error has reduced to an acceptable level the robot stops and the command that matches the task to be performed at this location is given to the robotic arm and executed using Inverse Kinematics (IK).

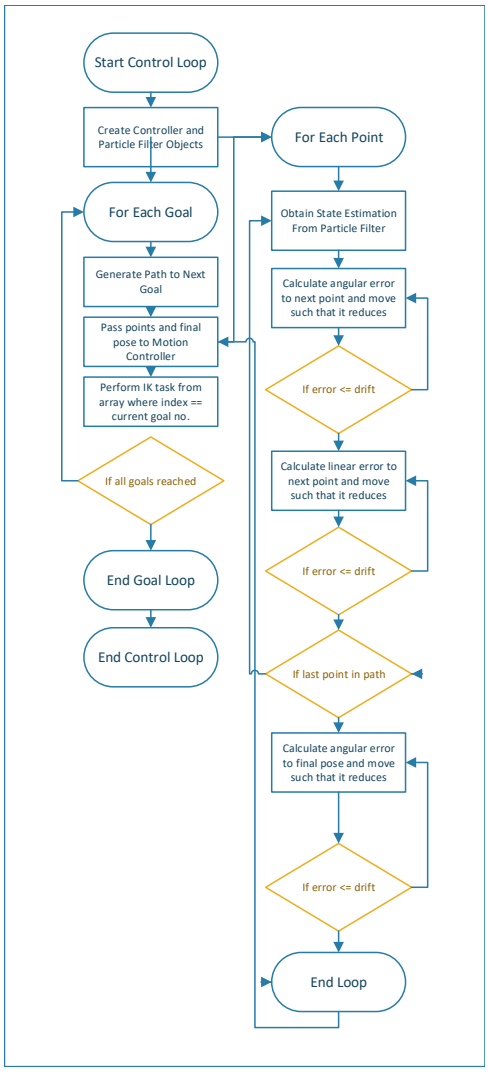


*Figure 2.* Main Control Loop. Component functions of path-finding, localization and perception models are omitted.

The key part of this system is the localization module as, without an accurate state estimation, the robot will not only fail to negotiate obstacles in the arena successfully but would also not arrive at the correct location to a degree of accuracy whereby the arm could execute its task and achieve the goal. For this reason Monte-Carlo Localization (MCL) using a Particle Filter (PF) was preferred to Grid Localization using a Bayes Filter. The trade off between time and accuracy was deemed to be too high a risk since the degree of accuracy required would be instrumental in completing all challenges successfully. Instead an optimized Particle Filter could provide much higher accuracy with few additional computational costs.

Given that the practical challenge involved a known map and constant initial position, global localization was not necessary. Instead, Monte-Carlo Localization was implemented using a particle filter as described by Thrun et al. (2005), with all particles initialised to the the robot's original pose. The particles then diverge from this point to the approximate error in the control information of the

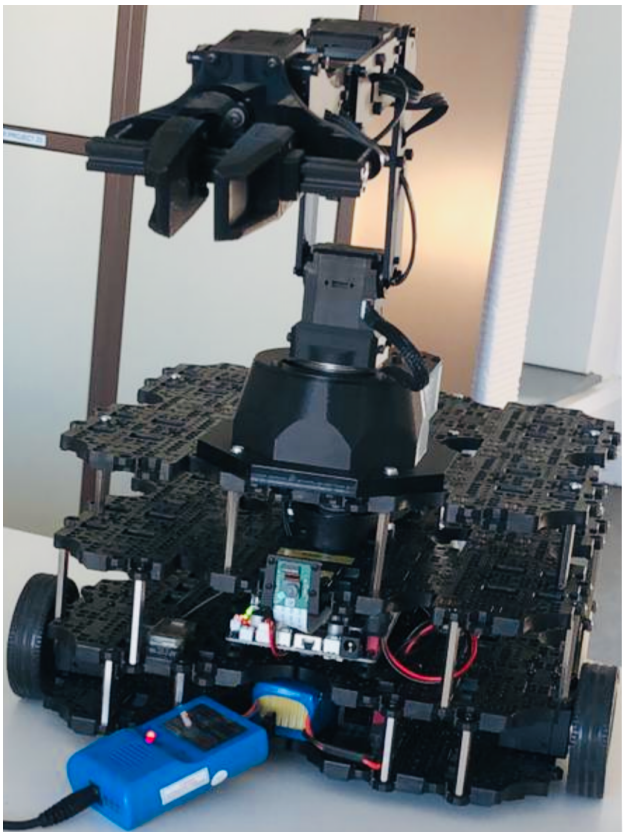

*Figure 3.* Turtlebot 3 with PX-100 arm

odometry motion model (Thrun et al., 2005). The error parameters of this model ($a1 - a4$) were determined from experiments conducted in the first major milestone of the challenge which required the robot to travel forward 1m, turn 90° clockwise, continue 1m forward, turn to face the original heading and then continue for a further 1m. These results are detailed in table 2. Each particle's importance was then weighted by calculating the probability of its state estimation by sampling the measurement model. The measurement model implemented consisted of a beam-model as described in Thrun et al. (2005) which compared each Lidar beams reading to a 'true' measurement obtained via ray tracing. The number of beams used to measure position was reduced to 8, with an interval of 45° between each. This was necessary to minimize the latency of the localization step in the control process and avoid erroneous readings since some of the Lidar beam's paths were occluded by struts used to mount the robotic arm above the turtlebot model (figure 3).

The probability values for expected measurements($P_{hit}$)), short measurements ($P_{short}$), random values ($P_{rand}$) and maximum measurements ($P_{max}$) are drawn from Gaussian, exponential, uniform and point-mass distributions, each of which is discounted by a weighting co-efficient ($z_{hit}, z_{short}, z_{rand}$ and $z_{max}$), to account for noise and sensor failures as well as determining the importance of each probability value in calculating the overall likelihood of the

reading. These weights were initially set to arbitrary values, with the highest importance being given to $z_{hit}$ and $z_{short}$ since these accounted for the expected measurement and any unexpected objects. $z_{max}$ was given a lower weight due to their being few points where the maximum measurement would be obtained validly given the size and occupancy of the arena. Finally, $z_{rand}$ was given the lowest weighting since these random values are obtained through erroneous and noisy readings and therefore should not be a large factor in determining the overall likelihood of a reading at obtained at a given state. Localization was also the element

**Algorithm 1** Ray Tracing algorithm to obtain true measurement where $X_{t-1}$ consists of $x, y, \theta$ and $O_{start}$ and $O_{end}$ are matrices containing the start and end co-ordinates of each obstacle on the map. $dist_{max}$ is the maximum measurable distance of the sensor.

**function** MIN_INTERSECTION ($X_{t-1}$,$O_{start}$,$O_{end}$):
  $P = x, y$
  $\bar{P} = \cos\theta * dist_{max} + x, \sin\theta * dist_{max} + y$

  **function** FIND_INTERSECTION($a1, a2, b1, b2$)
    $da = a2 - a1$
    $db = b2 - b1$
    $dp = a1 - b1$
    $dap = -a^{-1}$
    **return** $(\frac{dap \cdot dp^T}{dap \cdot db^T} \times I_n) \cdot db + b1$
  **end function**

  $int_{all}$ = FIND_INTERSECTION($O_{start}$,$O_{end}$,$P, \bar{P}$)
  $int_{dist} = \sqrt{(int_{all_x} - P_x)^2 + (int_{all_y} - P_y)^2}$

  **function** IS_BETWEEN($a$,$b$,$c$)
    $dist_{ab} = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2}$
    $dist_{ac} = \sqrt{(a_x - c_x)^2 + (a_y - c_y)^2}$
    $dist_{bc} = \sqrt{(c_x - b_x)^2 + (c_y - b_y)^2}$
    **return** True where $ab = ac + bc$
  **end function**

  **function** CHECK_ANGLE($a$,$b$,$yaw$)
    **return** True where:
    $yaw = \arctan 2(b_y - a_y, b_x - a_x)$
  **end function**

  $int_{true}$ = IS_BETWEEN($O_{start}$,$O_{end}$,$P, \bar{P}$)
  $int_{valid}$ = CHECK_ANGLE($Int_{all}$,$P$)
  $int_{pos} = int_{all}$ Where $int_{valid}$ & $int_{true} = True$
  **return** $min(int_{dist}$ where $int_{pos} = True)$
**end function**

that had the highest time complexity. While MCL itself had a complexity of $O(n)$, the measurement model sampled within MCL also had a time complexity of $O(n^2)$ leading to a total complexity of $O(n^3)$. This led to poor run-times which meant only a small number of particles could be used in real-time to estimate the current pose without incurring large amounts of latency in the control system. In order to solve this issue, the most inefficient function within the particle filter algorithm was optimized by replacing nested for loops with matrix multiplications and dot-products. Algorithm 1 details the full implementation of the ray-tracing function within the measurement model that returns the minimum intersection for a ray from a given pose. This function was called iteratively for each beam angle at a given location.

Experiments were then conducted using a simulated environment (Gazebo) to measure localization accuracy and tune the error parameters for both the measurement and odometry model. These were then validated by real-world tests which confirmed any issues and the level of accuracy displayed by the robot.

The aforementioned ray-tracing algorithm was also used to smooth the paths created by the path-finding script. An implementation of the A* algorithm was used to calculate a path to a given point based on a map of the arena. This map was represented as a matrix of size $(h/c, w/)$ where $h$ and $w$ are the width and height of the arena and $c$ is the cell size or resolution. This was set to $0.2m^2$ to help ensure that the robot would able to navigate any path obtained from A* since the grid would consider a cell occupied if it contained any co-ordinate of a line describing an obstacle in it. These values were padded by 10cm when the map was generated to again ensure the robot would not encounter any obstacles if it followed the generated path.

Since the robot's motion model has a high cost for each turn and demonstrated higher rotational error when tested, it is necessary to optimize the path by first listing only the points at which a turn was required in the grid map and then determining if intermediate points could be skipped in order to reach the goal. Similar to localization, the end point was tested to check that its path to the first point (point 1) did not intersect with an obstacle. If an intersection was found, the algorithm would then check whether the end point and point 2 could be reached directly and so on. When no intersection is found, the end point is added to the optimal path and its value is set to the newly identified endpoint. The process is repeated from this new end-point (and any subsequently identified points) until the start is reached. Due to no yaw value being passed to the path-finding algorithm, to check the angle is valid as in localization the sign of the distance values is checked for both points and the start point/obstacles, if these match we can conclude that an intersection is true. Once the optimal version of the A* path has been computed, the goal pose is added since the coordinfate transformation of a grid cell will only give its central point which, due to the low resolution, is not sufficiently accurate for our requirements.

The path planning function was tested and evaluated based on its efficiency in terms of path cost between goals and average run-time (table 1).

Inverse Kinematics was implemented by the other member of our team and was simplified by locking the rotational degrees of freedom (df) on our robotic arm and constraining it to 2df overall. This was done to reduce error that was

predominantly manifesting in the z-axis as it was deemed even minor errors could result in inaccuracies that would stop the robot from successfully picking up the small $3cm^2$ Lego block. The number of iterations was kept at around 100 to ensure smooth motion and minimal overshoot. The desired end-effector point has a constant y- axis(height), and thus, given our constrained movement, we only have to compute the distance from our point of origin and input it to IK to retrieve high-precision joint angles. The accuracy of the IK algorithm was tested by measuring error for each command manually. The average error was deemed to be 0.0147m for x axis motion and 0.0144 for movements in the y axis.

## 3. Results

All results presented in this section were obtained using the algorithms implemented as described in section 2 from a combination of real-world or simulated environments unless otherwise stated.

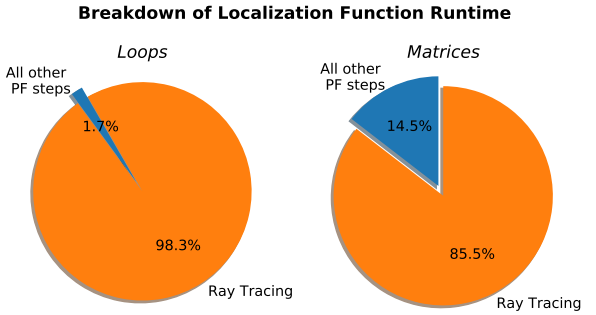


*Figure 4.* Proportion of localization run-times devoted to ray-tracing for both looped and dot product (matrix) approaches

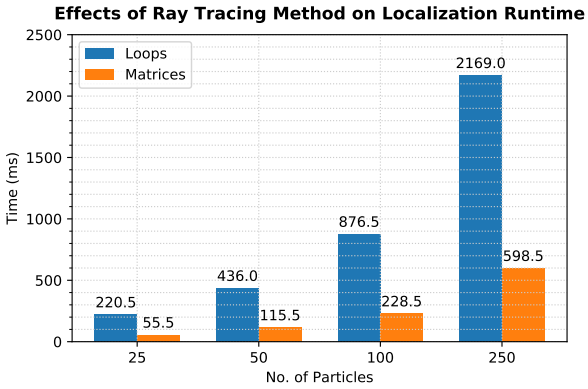


*Figure 5.* Comparison of localization run-times for an 8-beam ray-tracing measurement model using matrices or loops for sets of 25-250 particles

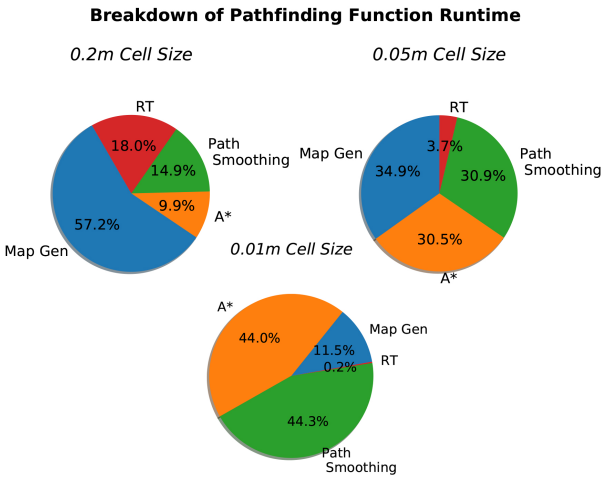


*Figure 6.* Summary of average function run-times in the A-star algorithm for various grid cell size maps

In order to initially investigate the issues of latency that were preventing localization from working in real-time with a high number of particles, the run-times of each function within the localization algorithm were examined. Figure 4 shows that almost all (98.3%) of the localization run-time is spent within the ray-tracing algorithm. This allowed for the targeted optimization of this algorithm and, once the majority of loops were replaced with Matrix Multiplications or Dot Product operations, the ray-tracing run-time reduced to 85.5% of the localization total.

Figure 5 demonstrates the effect this modification had on the overall run-time of localization for a single step with 25, 50, 100 and 250 particles. The optimized algorithm showed a 363% - 396% improvement in run-time, allowing for faster-localization which could use higher quantities of particles for more accurate state estimation. The same approach was used to evaluate the path planning algorithm. Since A* uses a graph-search to find paths it has an exponential run-time of $O(b^d)$ where b is the branching factor. In order to smooth the paths to produce the minimum cost relative to our motion model which completed rotational and linear motions independently of one another, the optimized ray-tracing algorithm was used to avoid a significant increase in the run-time. Figure 6 breaks down the time spent on creating a map, the A* search, the initial path smoothing, and finally smoothing using ray-tracing. This showed the exponential increase in time taken by the path-planning operations as the grid resolution increased compared to the more linear increases of the ray-tracing and map-generation algorithm run-times. Overall the algorithm completed path planning in an average of 0.085 1.13 and 72.11 seconds for cell sizes of $0.2m^2$, $0.05m^2$ and $0.01m^2$ respectively. While it may be possible to further optimize this algorithm by removing the intermediate smoothing before ray-tracing however, since this algorithm was operating with a low-resolution occupancy grid due to collision concerns and

| Path Type | Base → 1 | 1 → 2 | 2 → 3 | 3 → Base |
|---|---|---|---|---|
| Optimal Path | 1 | 1 | 1 | 2 |
| Smoothed A* + Ray-Trace | 1 | 1 | 1 | 2 |
| | 1 | 1 | 1 | 2 |
| | 1 | 1 | 1 | 2 |
| Smoothed A* | 2 | 2 | 4 | 9 |
| | 2 | 6 | 5 | 21 |
| | 2 | 11 | 22 | 120 |
| A* | 7 | 17 | 12 | 22 |
| | 27 | 67 | 43 | 85 |
| | 132 | 331 | 211 | 468 |

*Table 1.* Average path lengths obtained by each function for navigating to the listed task points. Each sub-row in the results denotes the path length for 0.2m , 0.05m and 0.01m cell sized maps

| Navigation Method | Linear Drift (metres) | Angular Drift (radians) |
|---|---|---|
| Dead Reckoning | 0.11 | 0.51 |
| PID Controller | 0.05 | 0.34 |
| PID + MCL | 0.03 | 0.15 |

*Table 2.* Average drift for odometry based motion (dead reckoning), interpolated motion using a PID controller with hard-coded drift and control values for linear motions of kp = 0.08, $kd$ = 0.005 & $ki$ = 0.01 or $kp_{ang}$ = 0.3, $ki_{ang}$ = 0.05 & $kd_{ang}$ = 0.01 for rotational motions, versus a PID controller plus Monte Carlo Localization with 25 particles

the robot would only run this while stationary, further optimization was deemed unnecessary for this task. Table 1 details the path lengths obtained from the various path planning operations for each goal position as defined in the practical task at various grid map resolutions. The optimal path as computed manually is also included for comparison. Here it is evident that regardless of grid resolution, the ray-tracing algorithm can reliably compute the optimal path from the output of the A* algorithm. This confirmed that a high resolution grid map would be redundant for this task given the known goal points and obstacle co-ordinates were used in conjunction with A* pathing. Table 2 details the average drift for linear and angular motions as calculated during the completion of the first milestone challenge and when simulated in the arena using the same target motions. Dead reckoning, implemented without any motion control showed high degrees of error. This was due no only to the lack of control of motion as movement was initiated which caused wheel slip, but also due to target over/undershoot since there was no control implemented to slow the motion as the error decreased and the robot approached its target translational/rotational goal.

When a PID controller was used this solved both the initial slip issues and the over/undershoot problems, however, without an accurate state estimation, there was still a slight drift that accumulated which had to be accounted for by including hard-coded error values. These were set to +/- 0.25 radians for the angular motions and 0.05 for the linear motions and allowed the milestone to be completed with very high accuracy.

| Particle Set Size | Linear Drift (metres)* |
|---|---|
| 25 | 0.03 |
| 50 | 0.11 |
| 100 | 0.32 |

*Table 3.* Average drift for Monte Carlo Localization with 25, 50 and 100 particles.
*These values were obtained using the semi-optimized ray-tracing code

The use of localization improved performance even further and was shown to reduce the cumulative drift as seen in the odometry-dependant models. However, this was only if localization could be executed in a suitable amount of time so as not to overshoot the goal due to excessive latency between the state-estimation and control steps. Table 3 shows results obtained from experimenting with the particle set size and the average errors which manifested during the completion of each individual goal task. Unfortunately these results were obtained before the fully-optimized ray-tracing was implemented meaning that the latency was still of a level which would cause the robot to overshoot its target with any particle set size greater than 25. This could be accounted for by adjusting the target value relative to the error in the motion controller.

Individually, these various subsystems performed to a high level, demonstrating good overall accuracy and efficiency. However, due to several issues causing the robot's particle set to 'reset' between navigation goals. They were unable to be fully demonstrated during the challenge assessment. As an ad-hoc solution, the particle-filter was changed to reset every particle to the previous state-estimate at each time-step. This meant that, despite slightly decreased accuracy, the robot was still able to reliably navigate to the first two goals in the arena. Sadly, due to an error when implementing the aforementioned solution which required hard-coded path values to be used, a mistake with the pose given prevented us completing the full challenge. It has since been confirmed in subsequent tests that this was purely human-error. Re-implementation of the particle filter and motion-model with a more Object-Oriented approach solved the problems with the particle-sets not surviving between goals.

## 4. Discussion

This paper has presented an analysis of the implementation and design of the various components of an automatic navigation and inverse kinematic system for the Turtlebot 3 robot to complete the tasks detailed in 1. The approaches taken to solve each of the path-planning, control, IK and localization problems were defined and justified in 2 with quantitative analysis of their performance in terms of accuracy and efficiency presented in 2.

Overall, the results concerning the localization accuracy and path-finding optimality demonstrate the need for ef-

ficient algorithms. The improved ray-tracing which was implemented has enabled both path-finding and localization to out-perform naive implementations of A* and MCL by a factor of 4. These savings in terms of path cost or iterative computations allow for higher accuracy through an increased particle set size budget for localization as well as a reduced need for high resolution grids to be able to find the optimal route to a goal in path-finding, thus avoiding excessive computational costs.

In addition to the work completed in this paper, a collision avoidance algorithm (such as wall-following) could be implemented in order to safeguard against localization failures or unexpected objects in a dynamic environment.

It is also possible that localization could fail in larger environments unless error parameters are tuned correctly. For that reason, global localization could be implemented or the state-estimation obtained from position-tracking localization as implemented could be fused with the odometry readings to "ground" the particle set using an Extended Kalman Filter (EKF) or similar. Additional Sensors such as a camera that could detect visual landmarks could also be incorporated to improve localization accuracy. It would also be possible to experiment with using more beam-angles in the state estimation process but this would increase the run-time by a linear factor.

Further work on path-planning could include ensuring collision free paths, even at a high-resolution, by extending the ray-tracing algorithm to compute parallel measurements co-linear to both horizontal edges of the robot. These lines could be used to check that the full robot frame could fit across the full length of the defined path, reducing the need for collision avoidance to be engaged.

## References

Thrun, Sebastian, Burgard, Wolfram, and Fox, Dieter. *Probabilistic robotics*. MIT Press, Cambridge, Mass., 2005. ISBN 0262201623 9780262201629. URL http://www.amazon.de/gp/product/0262201623/102-8479661-9831324?v=glance&n=283155&n=507846&s=books&v=glance.