Master-Studiengang Robotics Engineering
FH Technikum Wien, Höchstädtplatz 5, A-1200 Wien

COMPARISON BETWEEN KALMAN FILTER, EXTENDED KALMAN FILTER AND PARTICLE FILTER IN POSE ESTIMATION FOR A TURTLEBOT3

Authors: Knebl Fabian

Abstract: The localization of a mobile robot is a key challenge in robotics, especially when influenced by sensor noise, motion blur, and limited observability. This paper examines and compares three established probabilistic methods for state estimation: the Kalman filter (KF), the extended Kalman filter (EKF), and the particle filter (PF). The aim is to evaluate their suitability for 2D position determination of a robot within the framework of a ROS2-based Gazebo simulation. All three filters were implemented as standalone C++ nodes and use only internal sensor data (odometry, IMU) and control commands. The resulting estimates are compared with the Gazebo ground truth and analyzed in terms of accuracy, robustness, and computational effort. The results show clear differences in terms of model assumptions and applicability: While the KF delivers efficient results for linear movements, the EKF and, in particular, the PF show advantages for nonlinear behavior and inaccurate initialization.

Keywords: Kalman filter, Extended Kalman Filter, Particle filter, State estimation, ROS, Probabilistic robotics

1. INTRODUCTION

Mobile robots operate in a world characterized by uncertainty. Precise determination of their own position, known as localization, is a key prerequisite for autonomous navigation and decision-making. However, this task is not trivial in real-world environments: measurement errors in sensors, inaccurate control commands, and unpredictable influences from dynamic environments mean that a robot cannot directly know its actual position. Instead, it must estimate it from uncertain observations and inaccurate motion models. Probabilistic methods that can explicitly deal with uncertainties have become established to overcome this challenge. The aim of these approaches is to determine the most probable position of the robot from a combination of control commands (e.g., speed commands) and sensor data (e.g., odometry, inertial sensors). [1]
Three of the best-known methods in this context are: the Kalman filter (KF), the extended Kalman filter (EKF), and the particle filter (PF). All three are based on the Bayesian filter principle and differ in their assumptions, complexity, and suitability for different application scenarios. The aim of this project is to implement these three methods in a realistic scenario and compare them systematically. A mobile robot is used in a simulated environment. State estimation is based exclusively on speed commands, odometry, IMU data, and laser data. The focus is on the following research questions: How accurately do the filters estimate the robot's position in typical movement patterns? How robust are they against noisy sensors and inaccurate initialization? Which methods deliver reliable results under which conditions? How do the algorithms differ in terms of computational effort and stability? The investigation does not pursue a purely theoretical approach, but rather a practical, application-oriented one: Implementation as independent software modules and integration into a simulation environment enables a well-founded comparison based on realistic data.

2. MATHEMATICAL FUNDAMENTALS

Probabilistic localization is based on the assumption that the true state of a robot (e.g., its position and orientation) cannot be directly observed, but must be estimated from a series of uncertain measurements and movement commands. This task can be formally formulated as a recursive Bayesian estimation problem. [1]

2.1. BAYES FILTER

The Bayesian filter forms the theoretical basis for all three algorithms considered here: Kalman filter (KF), extended Kalman filter (EKF), and particle filter (PF). The goal is to determine the probability distribution $bel(x_t)$ about the system state x_t based on all previous observations $z_{1:t}$ and control commands $u_{1:t}$.

The Bayes Filter provides the theoretical foundation for most modern probabilistic state estimation methods in robotics. It models the robot's belief over its current state as a probability distribution and updates this belief recursively using control inputs and sensor measurements.

Each of the steps corresponds to:
Line 2–3: Predict the new belief based on the motion model and prior belief.
- Line 2 begins a loop over all possible states xt in the state space. In practice, this is either implemented analytically (e.g., KF), approximately (e.g., PF), or discretely (e.g., in grid-based methods).
- Line 3 computes the predicted belief (prior) over state xt using the total probability theorem. It integrates over all possible previous states xt−1, weighting the transition probability p(xt | ut, xt−1) by the previous belief bel(xt−1). This is the prediction step.

Line 4: Update the belief using the current observation.
- This is the correction step. The predicted belief bel̄(xt) is updated based on the likelihood of the current observation zt given state xt using the sensor model p(zt | xt).
- η is a normalization factor that ensures the belief distribution sums to one.

Line 5: End of the state-space loop.
- All possible states xt have now been processed, and the updated belief bel(xt) is fully computed.

Line 6: Return the posterior belief bel(xt).
- The algorithm returns the updated probability distribution over the current state, which serves as the input for the next time step.

This general form is not directly implementable unless specific assumptions are made about the system. Specialized filters such as the Kalman Filter, Extended Kalman Filter, and Particle Filter represent tractable approximations of this recursive Bayes update.

2.2. KALMAN-FILTER (KF)

The Kalman filter is a special case of the Bayesian filter, The Kalman Filter assumes a linear Gaussian system and provides an optimal recursive solution for estimating the robot's state under such conditions. It maintains a Gaussian belief over the state, represented as a mean vector μ_t and a covariance matrix Σt, and performs two main operations: prediction and correction.
The complete update steps of the Kalman Filter are shown in the algorithm below:

```
1:  Algorithm Kalman_filter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):
2:      $\bar{\mu}_t = A_t\, \mu_{t-1} + B_t\, u_t$
3:      $\bar{\Sigma}_t = A_t\, \Sigma_{t-1}\, A_t^T + R_t$
4:      $K_t = \bar{\Sigma}_t\, C_t^T (C_t\, \bar{\Sigma}_t\, C_t^T + Q_t)^{-1}$
5:      $\mu_t = \bar{\mu}_t + K_t(z_t - C_t\, \bar{\mu}_t)$
6:      $\Sigma_t = (I - K_t\, C_t)\, \bar{\Sigma}_t$
7:      return $\mu_t, \Sigma_t$
```

Figure 1: Algorithm Kalman_Filter

Each of the steps corresponds to:

- **Line 2–3:** Predict the new mean and uncertainty using the motion model and process noise.
- **Line 4:** Compute the Kalman gain, balancing prediction vs. measurement trust.
- **Line 5–6:** Update the state estimate based on the measurement z_t
- **Line 7:** Return the corrected belief.

This implementation assumes known system matrices A_t, B_t, C_t and Gaussian noise covariances R_t, Q_t. It performs well for linear systems but is not applicable when nonlinear dynamics dominate, which leads to the Extended Kalman Filter.

2.3. EXTENDED KALMAN-FILTER (EKF)

The Extended Kalman Filter (EKF) is a nonlinear extension of the standard Kalman Filter designed to estimate the state of systems whose motion and measurement models are nonlinear but differentiable. Unlike the linear Kalman Filter, which assumes all models are linear and noise is Gaussian, the EKF accommodates nonlinear models by locally linearizing them at each time step. The EKF maintains a Gaussian belief over the state, defined by its mean vector μ_t and covariance matrix Σ_t. At each iteration, it performs two main operations: prediction and correction. [2] The generic algorithm is shown below:

```
1:  Algorithm Extended_Kalman_filter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):
2:      $\bar{\mu}_t = g(u_t, \mu_{t-1})$
3:      $\bar{\Sigma}_t = G_t\, \Sigma_{t-1}\, G_t^T + R_t$
4:      $K_t = \bar{\Sigma}_t\, H_t^T (H_t\, \bar{\Sigma}_t\, H_t^T + Q_t)^{-1}$
5:      $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
6:      $\Sigma_t = (I - K_t\, H_t)\, \bar{\Sigma}_t$
7:      return $\mu_t, \Sigma_t$
```

Figure 2: Algorithm Extended_Kalman_Filter

Line 2–3: Predict the new mean and uncertainty using the nonlinear motion model and process noise.
- Line 2 applies the motion model g(·) to propagate the state estimate forward, accounting for the control input ut.
- Line 3 uses the Jacobian Gt to linearly approximate the model and update the covariance Σ̄t, adding the process noise Rt.

Line 4: Compute the Kalman gain, balancing prediction versus measurement trust.
- This gain Kt depends on the predicted uncertainty Σ̄t, the measurement model Jacobian Ht, and the measurement noise Qt.
- It determines how much to adjust the prediction based on the new observation.

Line 5–6: Correct the predicted mean and covariance using the actual measurement zt.
- Line 5 computes the innovation (difference between observed zt and predicted h(μ̄t)), weighting it by Kt to update the mean μt.
- Line 6 reduces the uncertainty Σ̄t to reflect the new information, updating the covariance Σt.

Line 7: Return the corrected belief.
- The algorithm outputs the updated Gaussian belief (μt, Σt), ready for use in the next prediction-correction cycle.

This approach assumes differentiable motion and measurement models and locally approximates them using first-order linearization. While effective for smooth nonlinear systems, the EKF may diverge if the system is highly nonlinear or poorly initialized, motivating alternatives such as the Particle Filter.

2.4. PARTICLE FILTER (PF)

The Particle Filter (PF), also known as Monte Carlo Localization, is a probabilistic technique used for robot localization based on a set of weighted samples (particles). It is particularly suited to nonlinear and non-Gaussian systems. The PF approximates the belief distribution over the robot's state by maintaining and updating a set of samples.

The pseudocode for the Particle Filter algorithm is shown below:

```
1:    Algorithm Particle_filter($\mathcal{X}_{t-1}, u_t, z_t$):
2:        $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
3:        for $m = 1$ to $M$ do
4:            sample $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$
5:            $w_t^{[m]} = p(z_t \mid x_t^{[m]})$
6:            $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
7:        endfor
8:        for $m = 1$ to $M$ do
9:            draw $i$ with probability $\propto w_t^{[i]}$
10:           add $x_t^{[i]}$ to $\mathcal{X}_t$
11:       endfor
12:       return $\mathcal{X}_t$
```

Figure 3: Algorithm Particle-Filter

Each of the steps corresponds to:

Line 2–6: Sample new particles based on motion and compute their importance weights.
- Line 2 initializes the temporary set $\mathcal{X}$t that will hold the new weighted particles, and clears $\mathcal{X}$t.
- Line 3 begins a loop over all M particles.
- Line 4 draws a new particle x_t[m] by applying the motion model to the previous particle x_{t-1}[m] and the current control input ut. This introduces motion uncertainty.
- Line 5 evaluates how likely the observation zt is given this particle's state using the measurement model. The result is the importance weight w_t[m].
- Line 6 stores the new particle and its weight as a tuple in the intermediate set $\mathcal{X}$t.

Line 7–11: Resample the particles according to their weights.
- Line 7 begins a second loop over M particles to perform resampling.
- Line 8 randomly selects an index i from the weighted particle set $\mathcal{X}$t with probability proportional to the weight w_t[i]. This gives higher-probability particles a higher chance of being selected.
- Line 9 adds the selected particle x_t[i] to the final set $\mathcal{X}$t (note: the weights are discarded in this step).
- Line 10 completes the resampling loop.

Line 12: Return the final particle set $\mathcal{X}$t.
- The output is the new set of M particles $\mathcal{X}$t, representing the updated belief bel(x_t). These particles are now equally weighted and ready for the next iteration.

This implementation makes no assumptions about linearity or Gaussianity and is capable of tracking multimodal distributions. However, it requires a sufficient number of particles to approximate the true belief accurately, and its performance depends heavily on the quality of the motion and measurement models.

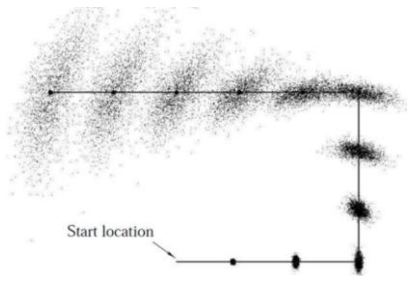


Figure 4: Particle Filter Behaviour

The figure above portrays the behaviour of a particle filter during robot localization. Initially, a large number of particles are randomly distributed across the environment to represent high uncertainty about the robot's position (global localization). As the robot begins to move and receives sensor observations, the particle cloud gradually converges toward the true pose.

At each time step, the motion model causes particles to spread according to the control input, introducing uncertainty due to odometry noise. Simultaneously, sensor measurements are used to weight the particles based on their consistency with observed features. The resampling step then concentrates particles in regions of high likelihood, discarding improbable hypotheses.

This effect is visible in the image:

- Near the "Start location", the particle set is widely dispersed.
- As the robot follows a path (black line), the cloud tightens and forms distinct clusters.
- Eventually, the particles converge to a compact distribution around the actual trajectory.

This convergence behaviour demonstrates the key strength of the particle filter: it is able to recover from global uncertainty, track multiple hypotheses simultaneously, and re-localize after failure, provided that sufficient observations are available.

2.5. LANDMARK-BASED MEASUREMENT UPDATE

In the EKF framework for mobile robot localization, landmarks provide known reference points in the environment, enabling the correction of pose estimates through sensor observations. [3]

Each landmark j is described by its position and signature:

$$\vec{m}_j = (x_j, y_j, \vec{s}_j)$$

where (x_j, y_j) defines the landmark's fixed location on the map, and $\vec{s}_j$ can represent additional signature information (e.g., type or ID).

The full map m is a collection of such landmarks:

$$m = \begin{bmatrix} \vec{m}_0 \\ \vdots \\ \vec{m}_n \end{bmatrix} = \begin{bmatrix} x_0 & y_0 & \vec{s}_0 \\ \vdots \\ x_n & y_n & \vec{s}_n \end{bmatrix}$$

During the **measurement update**, the EKF loop processes all observed features $\check{z}_t^i$. For each observed feature:

- The corresponding landmark ID c_t^i is used to **associate** the observation with the correct map landmark.
- The **expected observation** $\check{z}_t^i$ is computed based on the current predicted robot pose $\bar{\mu}_t$ and the landmark position.
- The innovation $(z_t^i - \check{z}_t^i)$ represents the difference between the actual measurement and its expected value given the current belief.
- The **Jacobian** H_t^i of the measurement model captures how small changes in robot pose affect the predicted observation.
- The **Kalman gain** K_t^i is computed to optimally weight the innovation, balancing predicted uncertainty and sensor noise.

These steps are repeated for all observed features, updating the robot's belief by integrating multiple landmark observations simultaneously.

The figure illustrates this concept: the robot senses multiple landmarks and uses their known positions on the map to refine its own position estimate, reducing uncertainty (represented by shrinking covariance ellipses).

3. IMPLEMENTATION DETAILS

This chapter describes how the three localization filters were implemented and integrated into the overall system. The goal was to create a common framework in which the Kalman Filter (KF), the Extended Kalman Filter (EKF) and the Particle Filter (PF) could be tested and compared under similar conditions. The entire project was built using ROS1 and tested in a simulated environment using Gazebo and RViz. A TurtleBot3 robot was used as the simulation platform.

3.1. OVERVIEW AND SETUP

To keep the system modular and easy to test, each of the three filters was implemented in its own ROS node. The core logic for each filter is written in C++ and follows the same basic structure:
each filter has a predict step, where motion updates are applied, and a correct step, where landmark measurements are processed. The robot is simulated in Gazebo and receives velocity commands from a simple controller that follows a list of predefined waypoints. All sensor data, such as odometry, IMU and landmark observations, are generated in the simulation and published over standard ROS topics. Each filter node subscribes to these topics and estimates the robot's pose based on the incoming data.

3.2. ROBOT AND SENSORS IN SIMULATION

The simulated robot is a TurtleBot3 Burger with differential drive. The robot publishes odometry and IMU data, both of which are affected by Gaussian noise. A set of landmarks is placed at fixed positions in the world. These landmarks are visible to the robot through a simulated landmark sensor that provides range, bearing, and landmark ID.

The filters do not rely on global position data. Instead, they estimate the pose based only on control commands and sensor readings. Ground truth data is only used for evaluation after the experiment.

3.3. FILTER IMPLEMENTATIONS

All three filters are launched from the same launch file and a separate pose-logging-node was developed in order to determine the Turtlebots position. The internal structure of the different filters differs:

- Kalman Filter (KF):
Assumes linear models for both motion and measurement. The system matrices A, B and C are defined manually. The filter uses odometry and landmark observations and applies the standard linear Kalman equations.

- Extended Kalman Filter (EKF):
Uses nonlinear models for motion and measurement. The robot's movement is modeled using differential drive kinematics. Landmark observations are modeled as range and bearing to a known landmark. Both models are linearized using Jacobians in each step.

- Particle Filter (PF):
Represents the belief using a set of particles. Each particle is updated using the motion model with noise. Landmark observations are used to assign a weight to each particle. Resampling is performed using low-variance sampling. The particle cloud is published and visualized in RViz.

Each filter node publishes its estimated pose and (where applicable) uncertainty to a common topic. This makes it possible to compare them directly. The particle filter additionally publishes the full set of particles for visualization.

3.4. NODE COMMUNICATION

All filters receive the same input data:

- /cmd_vel: velocity commands (from the waypoint controller)
- /odom and /imu: for motion updates
- /landmarks: list of visible landmarks with range, bearing, and ID

In each cycle, the filter performs the following steps:

1. Apply motion update based on velocity and sensor data
2. Process available landmark observations
3. Update the internal belief state
4. Publish the estimated pose
5. (Optional) log data for evaluation

All runs are performed in Gazebo, and RViz is used to visualize the results. The simulation environment is kept simple and controlled to ensure comparability between filters.

3.5. VISUALIZATION AND EVALUATION

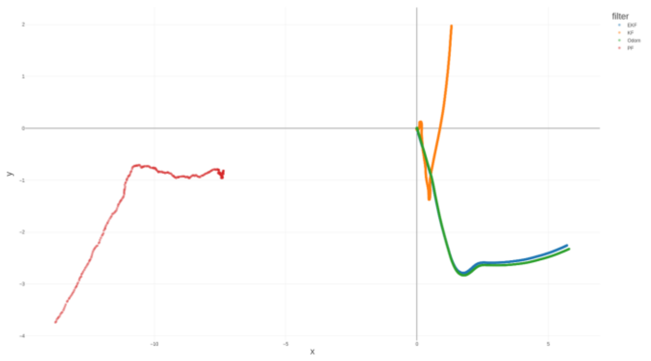
RViz is used to visualize the estimated pose of the robot, as well as additional information such as covariance ellipses (for KF and EKF) or particle clouds (for PF). The true pose of the robot (from Gazebo) is recorded and compared to the estimated pose in post-processing. The root mean square error (RMSE) is used as the main metric for evaluation.

4. EXPERIMENTAL RESULTS

To evaluate the performance of the three implemented filters, several test runs were carried out in simulation. The TurtleBot3 robot was placed in a known environment with fixed landmarks. The robot was commanded to follow a sequence of waypoints that led it along a rectangular trajectory with multiple curves and direction changes. Sensor noise was enabled, and the filter nodes had no access to global position information.

All three filters (KF, EKF, PF) received identical input data: velocity commands (from a waypoint controller), odometry, IMU data, and simulated landmark observations (range, bearing, ID). The filters estimated the robot's 2D pose (x, y, θ) in real-time. The true pose was obtained from Gazebo's internal model state and used only for evaluation after the run.

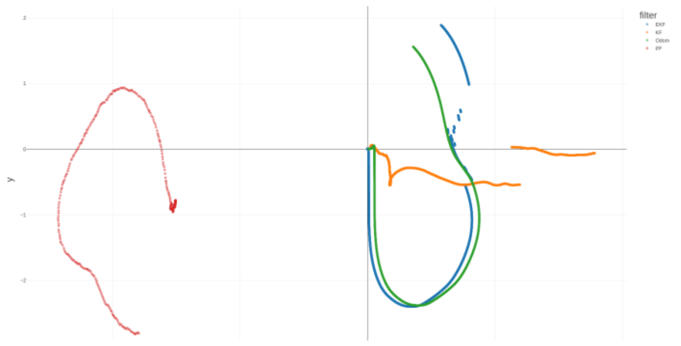
The estimated trajectories were visualized live in RViz and saved for offline comparison. The Root Mean Square Error (RMSE) was used to assess the accuracy of each filter. In addition, qualitative behavior was evaluated based on convergence speed, sensitivity to initial errors, and robustness in curved motion segments. The overall results were logged in a .csv-file and are shown in the following figure.



Green – odom; Orange – KF; Green – EKF; Red - PF

Figure 5: Plots over time linear

The Kalman Filter performed well in straight-line motion, especially when the initial estimate was accurate. Its linear model matched the actual motion of the robot in segments with constant velocity and heading. However, during turns and rotations, the filter tended to drift from the true pose, also does the actual position differ from the actual position while the algorithm tries to approach the actual pose. This was mainly due to the mismatch between the robot's nonlinear motion and the linear model assumptions. The uncertainty estimate (covariance ellipse) remained small but occasionally underestimated the actual position error.



Green – odom; Orange – KF; Green – EKF; Red - PF

Figure 6: Plots over time

The EKF handled curved motion better than the linear Kalman Filter. Its use of differential drive kinematics and nonlinear landmark measurements allowed it to remain close to the true trajectory in most parts of the test run. The uncertainty ellipses adapted well to the robot's heading and landmark visibility. In segments with poor landmark coverage, the EKF remained relatively stable.

The particle filter has shown very good results in tracking the robot's motion. Initially, the particle cloud was widely distributed, reflecting high uncertainty. After a few landmark observations, the particles began to converge towards the true pose. In areas with good landmark visibility, the estimated pose remained close to the ground truth.

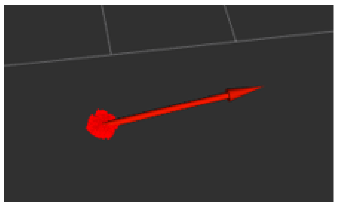


Figure 7: Particle Filter near landmark

In curved segments or when fewer landmarks were visible, the particle distribution became wider, especially with a lower number of particles. Increasing the particle count improved accuracy but also led to highered computation time. The filter recovered well after temporary divergence and was less sensitive to poor initialization compared to the Kalman-based filters.

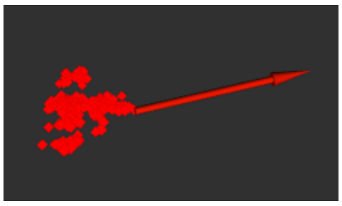


Figure 8: Particle Filter with distance to landmark

Visualizations in RViz confirmed expected behaviours. The Kalman Filter's function is given, but the results contain an error which leads to a significant expansion of the covariance matrix. The EKF performed better in general, provided landmark detection was available. The Particle Filter showed the most robustness overall but required careful tuning of the particle number and resampling strategy.

5. SUMMARY AND CONCLUSION

In this work, three well-established probabilistic localization methods were implemented and evaluated in a simulated mobile robotics environment: the Kalman Filter (KF), the Extended Kalman Filter (EKF), and the Particle Filter (PF). All three filters were developed independently in C++ and integrated into a modular ROS1 system using a TurtleBot3 robot simulated in Gazebo. The localization problem was formulated as a recursive state estimation task in 2D space, relying solely on control inputs, odometry, IMU data, and simulated landmark observations.

The Kalman Filter performed well under ideal conditions and in scenarios with nearly linear motion. However, its assumptions proved limiting in situations involving strong nonlinearities or inaccurate initialization. The Extended Kalman Filter showed improved robustness by incorporating nonlinear motion and measurement models. It produced more accurate pose estimates overall, although it was sensitive to tuning parameters and numerical stability.

The Particle Filter demonstrated high flexibility, particularly in ambiguous or uncertain environments. Its ability to represent non-Gaussian, multimodal beliefs made it robust to sensor noise and poor initial estimates. At the same time, the method required careful parameter tuning (e.g., particle count) and exhibited higher computational demand.

All filters were evaluated under comparable conditions using identical sensor inputs and test trajectories. The resulting estimates were visualized in RViz and quantitatively analyzed using RMSE metrics. The comparison confirmed that while each method has advantages, their performance strongly depends on the structure of the environment, available computational resources, and required estimation accuracy.

Future work may extend this system to real hardware or integrate more advanced data association techniques. The use of adaptive resampling in the PF or tighter integration with SLAM frameworks could further improve robustness and scalability. Overall, the project demonstrated the practical differences between the three methods and highlighted the importance of choosing a suitable localization approach depending on the application context.

6. REFERENCES

[1] W. B. a. D. F. S. Thrun, Probabilistic Robotics, Cambridge, MA: MIT Press, 2005.

[2] Y. K. a. H. Bang, Introduction to Kalman Filter and Its Applications, West Sussex, UK, 2018.

[3] A. L. J. Lazanas, Landmark-Based Robot Navigation. Algorithmica, 1995.

6.1. FIGURE TABLE