# Neural Networks in Robotics: Application of a Recurrent Neural Network to Robot State Estimation

Naomi Green, Soham Naik, Aaron Campbell, Jeremy Goens, Jeff McGough
Department of Computer Science and Engineering
South Dakota School of Mines and Technology
Rapid City, SD 57701

March 29, 2019

## Abstract

Robots make use of variety of sensors to gather information about their environment. Sensors are far from perfect and there are many ways errors arise. To eliminate these errors and increase accuracy, multiple measurements are fused together. The Extended Kalman Filter is a popular tool used to provide a more accurate estimation of the robot state given environmental uncertainties. The cost of the enhanced measurement afforded by the family of Kalman Filters is that they require motion and measurement models. In a number of cases the models are not always available or lack the required accuracy.

Neural networks have been successful in learning functions which would suggest that they could learn the motion and observation models as well as perform the required sensor fusion and state estimation. This paper will analyze existing methods of signal processing, and focus on using a Recurrent Neural Network in place of the Extended Kalman Filter. Using a neural network to learn the kinematics of the robot and errors in the system to provide a clearer estimate of state.

# 1 Introduction

Finding an accurate position estimate from sensors in robotics can be challenging. The sensor output is noisy due to intrinsic limitations of the sensor, connection of the sensor to the world as well as noise injected by the electronics, sampling and aliasing, interference, precision, and accuracy [4]. There are many approaches to removing the noise from a signal. Common tools are low and high pass filers or complementary filters. A very robust and powerful method of removing noise is known as the Kalman Filter (KF). The Kalman Filter is designed for linear processes which do not arise in robotics. The extension of the Kalman Filter to nonlinear processes is known as the Extended Kalman Filter (EKF) which will be our benchmark filter.

As mentioned above, the EKF requires motion and measurement models which can be difficult to accurately determine. In this work, a Recurrent Neural Network is proposed as an alternative to the EKF for state estimation for our robot.

We use a specific robot model called the differential drive robot. It is one of the most common hobby robot designs and is easily modeled. We will assume that the robot has some positioning sensors on it but not focus on the specifics of the sensors. For this work, we will have access to position and orientation information.

Several neural networks were looked into for filtering but due to its ability to make use of sequential data, a Recurrent Neural Net was used. This paper explores if a Recurrent Neural Network would work as a good replacement to the EKF. Once the network is trained on the robot, there would be very little computation necessary to get the resulting estimate. This would be more important as robots need to respond quickly to their environment. For example, self driving cars receive a lot of different sensor input, and need to be able to determine where they are and how they need to react in real time. A basic LSTM network was used for the preliminary research, and this paper documents the effectiveness and overall results.

# 2 Differential Drive Robots

Our case study examines using a neural network to determine the position of a differential drive robot. A differential drive robot is a simple ground robot with two independently driven wheels, Figure 1. By moving the wheels a different rates, the robot can steer. The motion of the robot is given by a simple kinematic model, Equations 1. The wheel radius is given by $r$ and the distance from the axle center to the wheel is given by $L$. The right and left wheel speeds are given by $\dot{\phi}_1$, $\dot{\phi}_2$ respectively (in radians per unit time). The robot is tracked using its position in the inertial or global coordinate system $(x, y)$ and the angle off of the inertial $x$ axis, $\theta$. Normally this bundled into a pose tuple $(x, y, \theta)$.

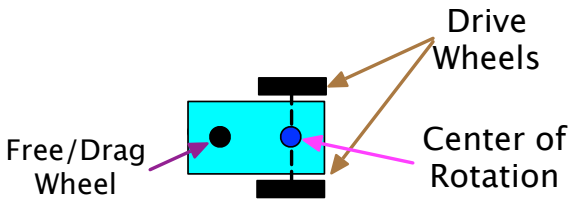


Figure 1: The Differential Drive Robot

$$\boxed{\begin{aligned} \dot{x} &= \tfrac{r}{2}(\dot{\phi}_1 + \dot{\phi}_2)\cos(\theta) \\ \dot{y} &= \tfrac{r}{2}(\dot{\phi}_1 + \dot{\phi}_2)\sin(\theta) \\ \dot{\theta} &= \tfrac{r}{2L}(\dot{\phi}_1 - \dot{\phi}_2) \end{aligned}} \tag{1}$$

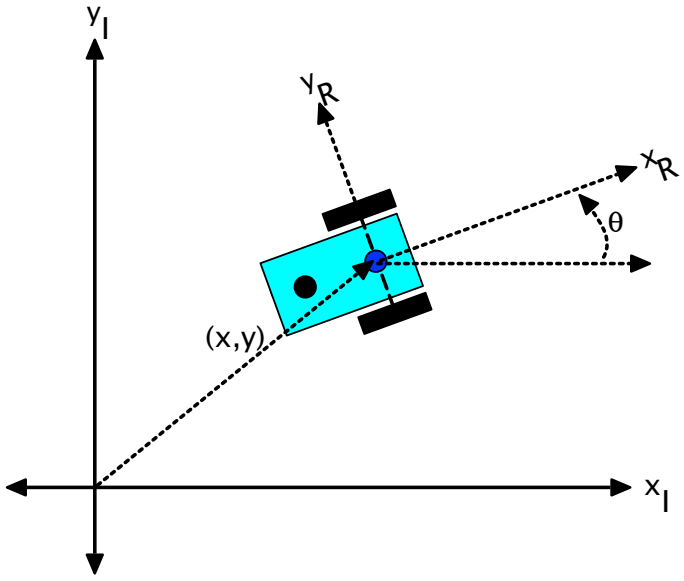


Figure 2: The coordinates used in the Differential Drive Robot model. $x_I$ and $y_I$ are the global or inertial coordinates. $x_R$ and $y_R$ are the robot coordinates. $\theta$ is the orientation of the robot in the global frame.

Since the computer world is discrete, simulation of the robot's location (and what we use to create the test data with) is constructed from the discretized kinematics, Eqn 2.

$$\boxed{\begin{aligned} x_{k+1} &= x_k + \tfrac{r\Delta t}{2}(\omega_{1,k} + \omega_{2,k})\cos(\theta_k) \\ y_{k+1} &= y_k + \tfrac{r\Delta t}{2}(\omega_{1,k} + \omega_{2,k})\sin(\theta_k) \\ \theta_{k+1} &= \theta_k + \tfrac{r\Delta t}{2L}(\omega_{1,k} - \omega_{2,k}) \end{aligned}} \tag{2}$$

# 3 The Extended Kalman Filter

The process of filtering out the noise from a signal is commonly called state estimation. In our case, the robot state is the vector of all of the unknown variables related to position and orientation. So for the differential drive example, the two variables for position and one for orientation: $(x, y, \theta)$ are the state we are attempting to estimate. In the digital world, we don't know these values for all time, we have them at specific times: $(x_k, y_k, \theta_k)$ where $k$ is the discrete time variable. We will abuse our notation by just referring to the state as $x_k$ in the EKF algorithm (but knowing from context this is a vector of position and orientation values).

EKFs are state estimators that assume a Gaussian distribution of the noise, the current state and the initial state. So the algorithm tracks the means and covariances of the states where the mean is the state estimate. There are plenty of good references on the Kalman Filter, Extended Kalman Filter and various Filter implementations [8, 7, 6].

The EKF assumes a nonlinear process model of the form

$$x_k = f(x_{k-1}, u_k)$$

and an observation of the state via

$$z_k = h(x_k).$$

The EKF Algorithm is given in Alg. 1. The variable $x$ is the state estimate, $F_k$ is the Jacobian of the system dynamics evaluated at the current step. For this application we will assume that the observation is linear and the observation matrix is denoted $H_k$. The input $z_k$ is the sensor measurement or observation at the current time step. $V$ and $W$ are the noise covariance matrices for the process and the observation respectively which we will assume are constant in this application. The variable $P_k$ tracks the covariance of the estimate $x_k$ and can be used as a measure of certainty of the estimate.

The filter consists of two stages, a predictive step and an update step. During the predictive step, the system dynamics and the model are used to predict where the robot should be. During the update step, the system takes the prediction and the observation to come up with an estimate of the system state [4]. Our goal is to have the EKF replace both of the stages.

**Algorithm 1** Extended Kalman Filter [6]

**Input:** $x_0$, $P_0$
**Output:** Estimates of $x_k$, $P_k$
$k = 0$
**while** Not Terminated **do**
  $k = k + 1$
  $x_k = f_k(x_{k-1}, u_k)$
  $F_k = \text{Jac}(f_k)|_{x_k}$
  $P_k = F_k P_{k-1} F_k^T + V_k$
  $y_k = z_k - H_k x_k$
  $S_k = H_k P_k H_k^T + W_k$
  $K_k = P_k H_k^T S_k^{-1}$
  $x_k = x_k + K_k y_k$
  $P_k = (I - K_k H_k) P_k$
**end while**

# 4 Long Short-Term Memory

A Recurrent Neural Network was selected due to its ability to make use of sequential data. This is due to the output of those nodes being used as input for the next run. This allows the network to retain context results for any number of runs that is defined by the weights[5]. The long short-term memory (LSTM) is the most commonly used version of a RNN and is what was used. Since the information in an RNN goes in a loop, the error gradients accumulate during an update which results in large updates. This results in an unstable network and is known as the vanishing gradient problem[5]. The LSTM was developed to overcome the difficulty of training a RNN. It does this by having an architecture that allows it to forget the unnecessary information through a forget gate.

# 5 Implementation details

Keras, a high level API for neural networks in python, was used to quickly build and test the network. Tensorflow was used for the back end of Keras. The network that was used to start with contains 3 hidden nodes in the LSTM layer, and 3 hidden nodes in a densely connected layer. The activation function used was a linear function. The input to the network is the current sensor input, and the sensor input from the previous run.

The input to the network is a simulated differential drive robot which has an x, y location and an angle. The DD robot that was simulated had an r of 4 and an L of 6. For the DD robot that was being modeled, a random path generator was used to generate realistic robot motion for training and evaluation data sets. For the random path, bounds are sent in to restrict the velocity of each wheel. This is done to keep the robot moving in a realistic

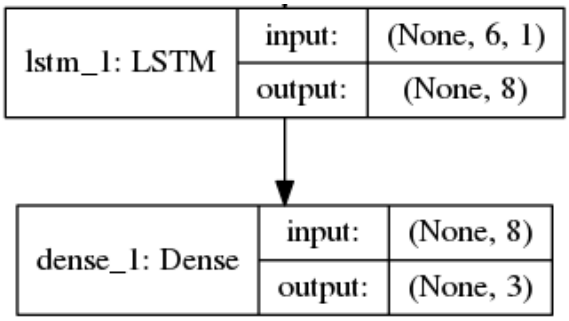


Figure 3: The network layout provided by Keras

pattern. Generating the data allows for there to be no limit to the data set size. Additionally, with generating new data each time we didn't have to worry about over fitting the data.

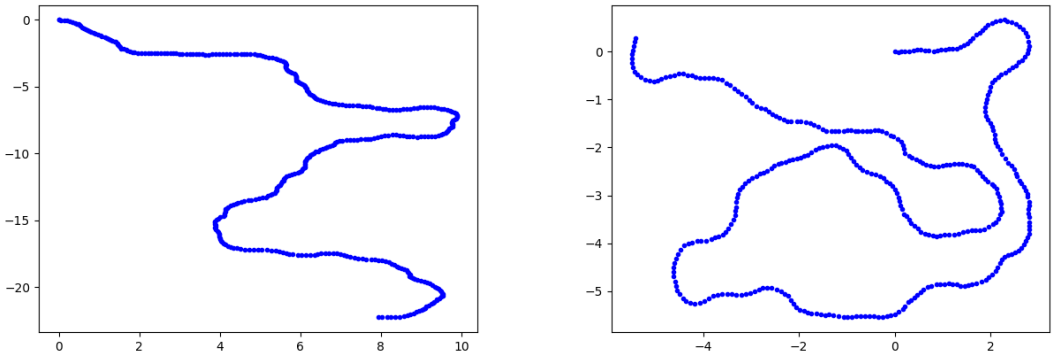

Figure 4: Sample paths of the random path generator

Training the network consists of a clean signal (position and orientation) in which noise is introduced, the network is meant to return the clean signal (actual state of the robot). The noise is introduced using a Gaussian distribution. There were two types of noise, process error and observation error. The observation error is the error from the sensors that was being filtered out. Both were Gaussian distributions, the process error had a standard deviation of .025, and the observation error had a standard deviation of .84. Testing included getting the average loss when evaluating on multiple different data sets. The loss is calculated using the mean squared error.

# 6 Results

Figure 5 shows graphs of the result of running the Recurrent Neural Network. The red is the data sent in, the blue is the actual position, the green is the prediction, and the purple in the second image is the result the Extended Kalman Filter gets. The picture on the left

has the x and y positions and the angle, the picture on the right shows the position of the robot. Figure 6 shows that the neural net was able to be accurate. These results indicate that further research and testing could prove successful.

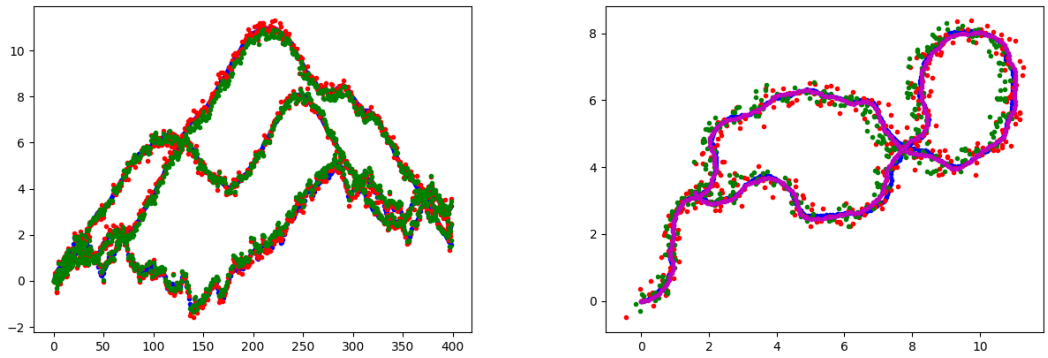


Figure 5: Graph of the results of the neural net when compared to the input, and the EKF result.

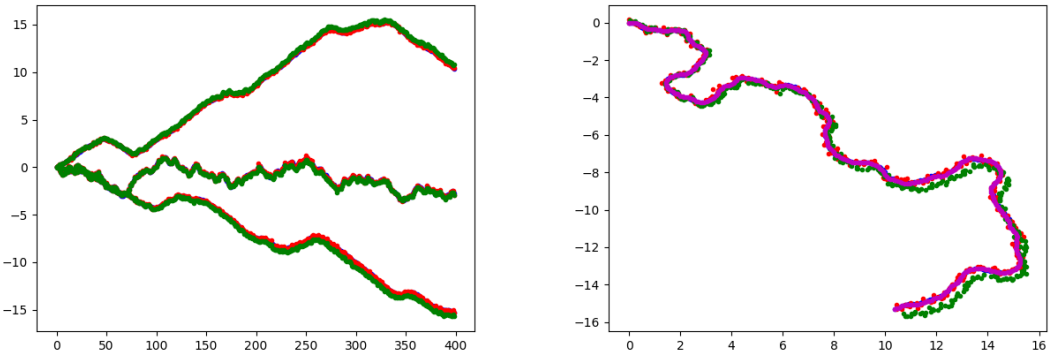


Figure 6: Graph of results that were similar to the EKF. This was achieved by introducing less noise into the input data.

## 6.1 Effect of hidden nodes in LSTM layer

The number of hidden nodes in the LSTM layer were changed to see what the optimal size would be for the network. The multiple tests showed that the loss was minimized at 8 hidden nodes. However this was only tested up to 100 hidden nodes. This was due to time constraints on the training process. When testing on different sized networks, the error was initially high then quickly dropped. The low point was hit at 8 hidden nodes in the LSTM layer, then the error leveled off a little above 1.

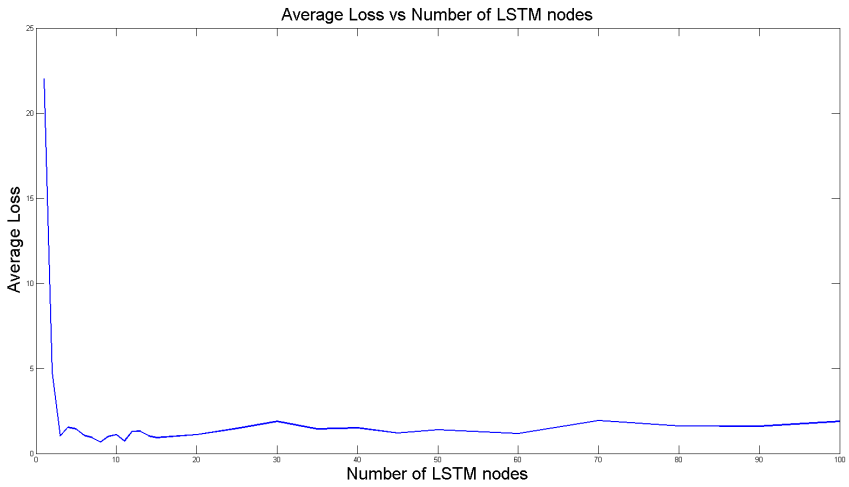


Figure 7: Effect of the number of hidden nodes in the LSTM layer

The number of hidden nodes in the LSTM layer was set to 8 for the rest of the experiments since that had the best result.

## 6.2 Effect of noise in training sets

For the data presented in section 4.1, the data used to train the network and the data used to evaluate the trained network had the same distribution of process and observation noise. In this section, we trained the network using smaller distributions of noise but kept the data used for evaluating loss of the network at the full amount of noise. These tests simulate training a network on simulated ideal or close to ideal data, then evaluating its effectiveness on real, noisy data on a real robot.

As can be seen in the figure above, the neural network performs best when trained with less noisy training data than the data it is eventually evaluated against. In our experiments, we found that a noise level of 20% the level of that used for evaluation performs best for training data. This may suggest that the neural network is having trouble learning the equations of motion for the robot in the presence of greater levels of noise, which is why it performs better with little to no noise in the training sets.

## 6.3 Overall effect of input noise

To see the effect noise has on the results of the network, the observation and process error were modified. This showed that the lower the process error and observation error, the better the prediction. There also appears to be a knee in the average loss curve when the

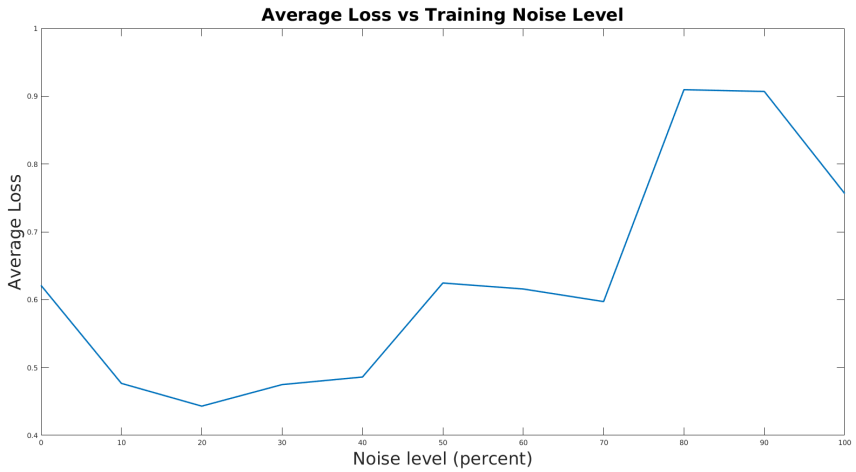


Figure 8: Effect of training noise level of sensor data

noise level goes above 30% of maximum, which may suggest that this particular neural network is optimal for robotic systems with high quality sensors that have low noise.

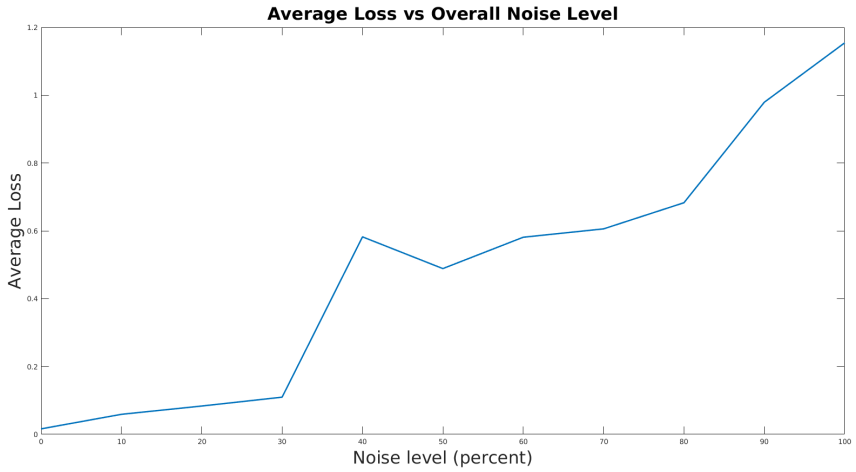


Figure 9: Effect of the overall noise level of sensor data

## 6.4 Effect of epoch size

This was the measure of how many times the the network was updated on a single path that was sent in. To get as consistent of results as possible, the epochs were reduced while the number of data sets trained on went up. This ensured that the network was being updated the same number of times for each run. The network performed better when the epoch size was small showing that the number of data sets was more important then making it more accurate on the data sets sent in.

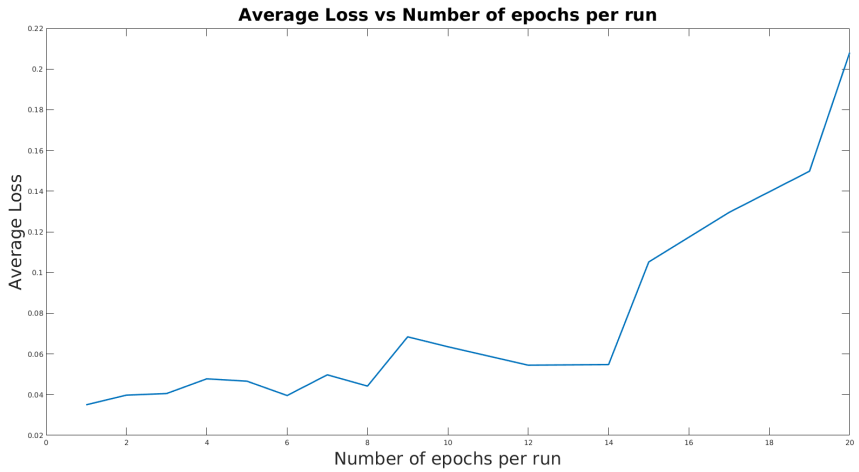


Figure 10: Effect of the epoch size on the accuracy of the estimate

# 7 Conclusion and Future Directions

This work looked into the possibility of using a Recurrent Neural Network in place of the Extended Kalman Filter. Using a neural net would allow the user to not need to know the models ahead of time. The network would be able to learn them when training. Each of the graphs of the data measured had a single minimum that was used to improve the results in the other tests. While the EKF is still a better option for most cases, there were some interesting results in changing different variables. In some cases where there is low error in the measurements, and the model isn't known this could prove to be a good alternative. With some more adjustments to the network it would be expected to be a better estimation.

The results indicated that this is a fruitful direction. Using a Neural Net to replace the Extended Kalman Filter would be valuable. The neural net didn't require any knowledge of the process model, it was able to learn it. This would be a very powerful technique when the process model isn't well known.

For future work, more investigation into the overall structure of the neural network could improve the results. Possible changes to the network could be the addition of more LSTM or dense layers, a dropout layer to avoid over fitting, or some combination of these changes. Additionally, further investigation into the activation function used by the different layers in the network could yield improved results over the linear activation function used in this paper.

# References


[1] Huseyin Coskun, Felix Achilles, Robert DiPietro, Nassir Navab, Federico Tombari. *Long Short-Term Memory Kalman Filters: Recurrent Neural Estimators for Pose Regularization*. 2017 IEEE International Conference on Computer Vision (ICCV), 2017, doi:10.1109/iccv.2017.589.

[2] Robert C. Wilson, Leif H. Finkel. *A Neural Implementation of the Kalman Filter* NIPS2009

[3] Peter Trebaticky. *Recurrent Neural Network Training with the Extended Kalman Filter*. M. Bielikov (Ed.), IIT.SRC 2005, April 27, 2005, pp. 57-64

[4] Jeff McGough. *Robotics*. RoboScience, 2 Dec. 2018, www.roboscience.org.

[5] Y. Bengio, P. Simard, and P. Frasconi. *Learning long-term dependencies with gradient descent is difficult*. IEEE Transactions on Neural Networks, 5(2):157166, 1994.

[6] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George A Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Cambridge, MA, June 2005.

[7] Roland Siegwart and Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. Bradford Company, Scituate, MA, USA, 2004.

[8] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. Intelligent robotics and autonomous agents series. Mit Press, 2005.