# IN-SITU SENSOR CALIBRATION USING NOISE CONSISTENCY

by

SHRIIESH VAR SHARMA

THESIS

Submitted in partial fulfillment of the requirements

for the degree of Master of Science in Computer Science at

The University of Texas at Arlington

August 2020

Arlington, Texas

Supervising Committee:

Manfred Huber, Supervising Professor

David Levine

Vamsikrishna Gopikrishna

Copyright © by Shriiesh Var Sharma 2020

All Rights Reserved

To Mom and Dad for their constant support.

Without them I would not exist!

Acknowledgements

I thank my advisor Dr. Manfred Huber, for accepting me as his mentee and guiding me in this research. He always pointed me in the right direction while still leaving enough space for exploring various related topics. I thank him for being patient with me and for ensuring that I understand the art of research.

I thank my committee members David Levine and Dr. Vamsikrishna Gopikrishna for taking the time to serve on my committee.

I thank my friends Sandeep, Kalyan, Megha, Tejas and Safwan for their support. You have been a part of my journey at UTA and have become like a family away from home.

September 3, 2020

## Abstract

IN-SITU SENSOR CALIBRATION

USING NOISE CONSISTENCY

Shriiesh Var Sharma, M.S.

The University of Texas at Arlington,2020

Supervising Professor: Dr. Manfred Huber

Robots rely on sensors to map their surroundings. As a result, the accuracy of the map depends heavily on the sensor noise and in particular on accurate knowledge of it. The common way to minimize the impact of sensor noise is to use filtering algorithms.

Accuracy of these filtering algorithms (like the Kalman filter) relies on the accuracy of the user supplied measurement noise model. Inaccurate noise models lead to higher residual noise in state estimates and errors in the estimate of the precision of the state estimate. It is therefore important to have precise noise models and thus accurately calibrated sensors.

Most current methods for estimating noise models require a knowledge of 'ground truth' labels for sensor data and often require either to remove the sensor from the system or the presence of particular, sensor-specific calibration targets. This method can be expensive and require modifications to the system or the environment.

In this research, we present a method for estimating noise models for multiple sensors without prior knowledge of ground truth and without the use of calibration targets. In contrast, this method takes advantage of identifiable targets in the environment to calibrate sensors against each other using a sensor noise consistency measure based on KL Divergence. This algorithm can be run periodically to update model estimates in unforeseen environments.

Table Of Contents

# Chapter 1

# INTRODUCTION AND RELATED WORK

## 1.1 Introduction

Intelligent robots have gained popularity in today’s world. They are slowly becoming a part of our daily lives. Autonomous cars, automated manufacturing lines, or robotic arms for surgery are just a few examples. Efforts have been made to make these autonomous systems flexible so that they can operate in the real world without any special concessions. For example, in some cases, specialized environments are made for robots to carry out very specific tasks. These robot specific environments can escalate the cost of automation. Additionally, robots designed to work here will experience a drastic drop in productivity or even completely fail to operate in a new environment. To address this , extensive research has been proposed in the field of robotics perception.

Akin to humans, robots need to observe/perceive their environment to perform any task. Hence, perception is one of the most important tasks in order to achieve autonomy and consistent performance in changing environments. Sensors are used to observe various physical variables in the environment.

Two important tasks when applying perception to navigation are Localization and Mapping. Localization is the task of state estimation given a map of the environment. This can be achieved by using measurements from one or multiple sensors and finding the distance of the robot to landmarks in the environment. Mapping is the task of estimating the environment or map, given observations and state estimates.

In most real world applications, it is expected from a system to be flexible and work in any new or unseen environments. In such a case, information about our location or the map is not always available. Here, a technique called Simultaneous Localization and Mapping(SLAM) is used. In a more formal description , SLAM can be defined as the problem of constructing and updating map of the environment while simultaneously keeping track of the robot's location within this map.

On the surface, this might seem like a chicken and egg problem. But, there are several algorithms that can find an approximate solution within acceptable time constraints. Most widely used algorithms are derivations of Bayesian filters, such as Kalman Filter, Extended Kalman filter, Particle filter etc.

Sensors are noisy. There are multiple factors that contribute to noise, for example, improper zero reference, physical damage, random changes in environment etc. An important consideration here is to handle the sensor noise. States can be predicted based on a motion model, but sensor observations are needed to update our belief in this predicted state. If there existed a sensor with zero noise, one could simply use those sensor readings as the state estimate. But, zero noise sensors are a myth, at least in today’s world. Hence, the need to estimate how noisy the sensor is. The accuracy of this assumed noise model has a significant impact on performance of the filter and the state estimates.

The process of identifying and modelling the sensor noise is referred to as sensor calibration. It is important to calibrate the sensors before deploying them in any real world scenario. Inaccurate noise models may lead to higher residual noise in state estimates and errors in the belief of the precision of state estimates. Most current methods for calibration require a knowledge of 'ground truth' labels for sensor data and some sensor specific calibration targets.

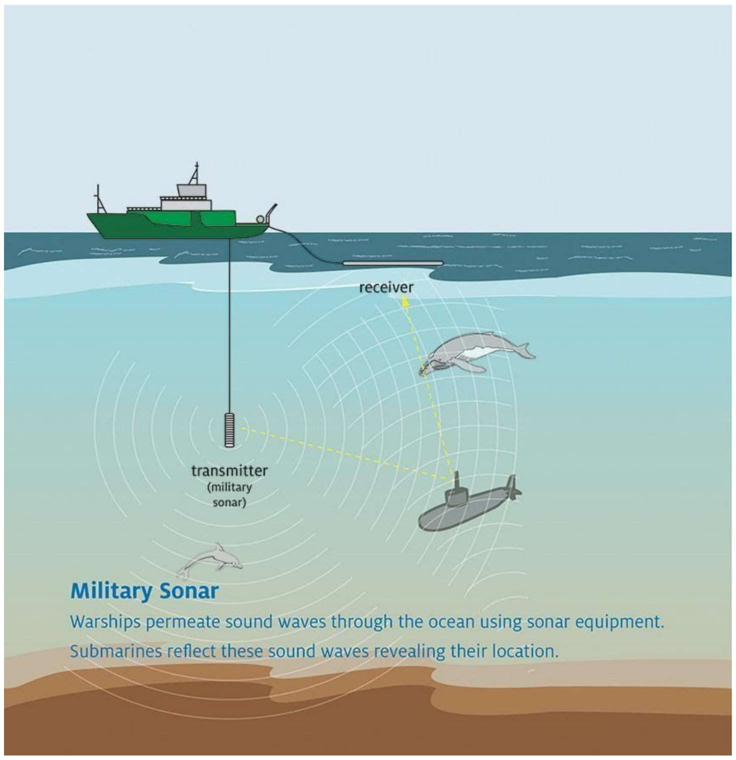

receiver
transmitter
(military
sonar)
Military Sonar
Warships permeate sound waves through the ocean using sonar equipment.
Submarines reflect these sound waves revealing their location.

Figure 1 : Workings of a military sonar. Presence of random objects or animals may cause erroneous readings.

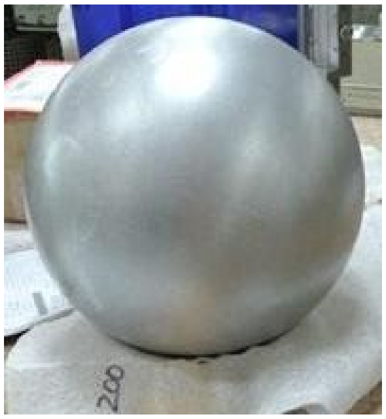
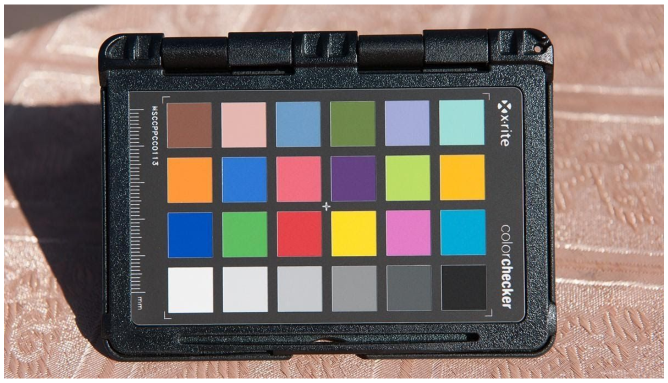



Figure 2: Radar Calibration Sphere

Figure 3: True Color calibration for cameras

The Main contribution of this research is to investigate novel techniques for sensor calibration. Specifically, trying to eliminate the need for sensor specific calibration targets. To achieve this, a sensor noise consistency measure based on KL Divergence is used.

## 1.2 Related Work

Sensors are an important part of any robot, but their applications are not limited to just this. Use of sensors is very widespread today. Hence, people have tried to solve the problem of sensor calibration with the knowledge and experience in their respective fields.

Bychkovskiy et al [10] talks about challenges faced in a large scale deployment of sensors over large distances. It is mentioned that calibrating each and every single sensor becomes intractable. Also, sensor maintenance can add another challenge if each time sensors need to be recalibrated. They try to solve a very similar problem of calibrating the sensors while they are deployed. Since physical locations of sensors are known and fixed, they employ observation bias from sensors in the vicinity to calibrate any given sensor in their network.

In Nikolic et al [12], the proposed methodology identifies noise processes across a large range of strength and time-scales. For example, they consider weak gyroscope bias fluctuations buried in broadband noise. This is accomplished with a classical maximum likelihood estimator, based on the integrated process (i.e., the angle, velocity, or position), rather than on the angular rate or acceleration as is standard in the literature. This simple modification allows the authors to capture noise processes according to their effect on the integrated process, irrespective of their contribution to rate or acceleration noise. The cause of the noise is not discussed in this article.

Muhammad and Lacroix [13] talks about using ‘ground truth’ labels for sensor data. LIDAR specific targets are used to perform calibration.

There are many similar examples of research on sensor calibration. It appears that no one has attempted to calibrate sensors by employing statistical analysis or noise consistency measures based on KL divergence.

# Chapter 2

# TECHNICAL BACKGROUND

## 2.1 Simultaneous Localization and Mapping (SLAM)

For robots to perform tasks, it is important to know where they are, i.e. their location in the world map. This process is called localization.

Also, the knowledge about the world and surrounding objects is important and can be achieved by observing and storing their distance from the robot's current location. This is called mapping.

The simultaneous localization and mapping (SLAM) problem asks if it is possible for a mobile robot to be placed at an unknown location in an unknown environment and for the robot to incrementally build a consistent map of this environment while simultaneously determining its location within this map.[3]

When building a robot that can perform in different environments, it is safe to assume that location and map information will not be available. Hence, it is required to perform simultaneous localization and mapping. An estimate of the observable environment is created using the sensor observations, and based on this, an estimate of the robot's location within this map is created. Although this might seem like a chicken and egg problem, there are algorithms that can perform SLAM within acceptable time constraints. Most widely used algorithms are from a family of filters called Bayesian filters. For example: Kalman filter, extended Kalman filter and particle filters. The Kalman filter was used in the Apollo missions for moon landing. This serves as a testament to their robustness and reliability.

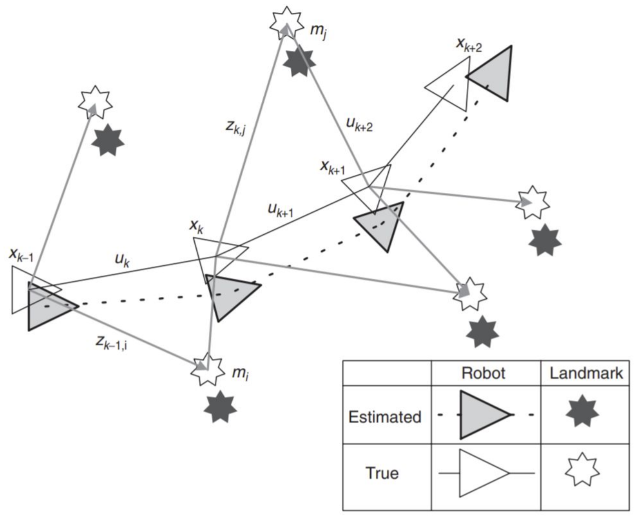


Figure 4: simultaneous estimate of both robot and landmark locations is required. The true locations are never known or measured directly. Observations are made between true robot and landmark locations.[3]

## 2.2 Bayesian Filtering

Bayesian theory[5] is a branch of mathematical probability theory that allows people to model the uncertainty about the world and the outcomes of interest by incorporating prior knowledge and observational evidence. Bayesian analysis, interpreting the probability as a conditional measure of uncertainty, is one of the popular methods to solve the inference problems. In Bayesian inference, all uncertainties (including states, parameters which are either time-varying or fixed but unknown, and priors) are treated as random variables. The inference is performed within the Bayesian framework given all of the available information. The objective of Bayesian inference is to use priors and causal knowledge, quantitatively and qualitatively, to infer the conditional probability, given finite observations.[5]

In robotics, Bayesian filtering is an algorithm that helps in calculating probabilities of multiple state beliefs, and allows the robot to infer its state. It recursively estimates the robot's state and updates the belief in state using observations.

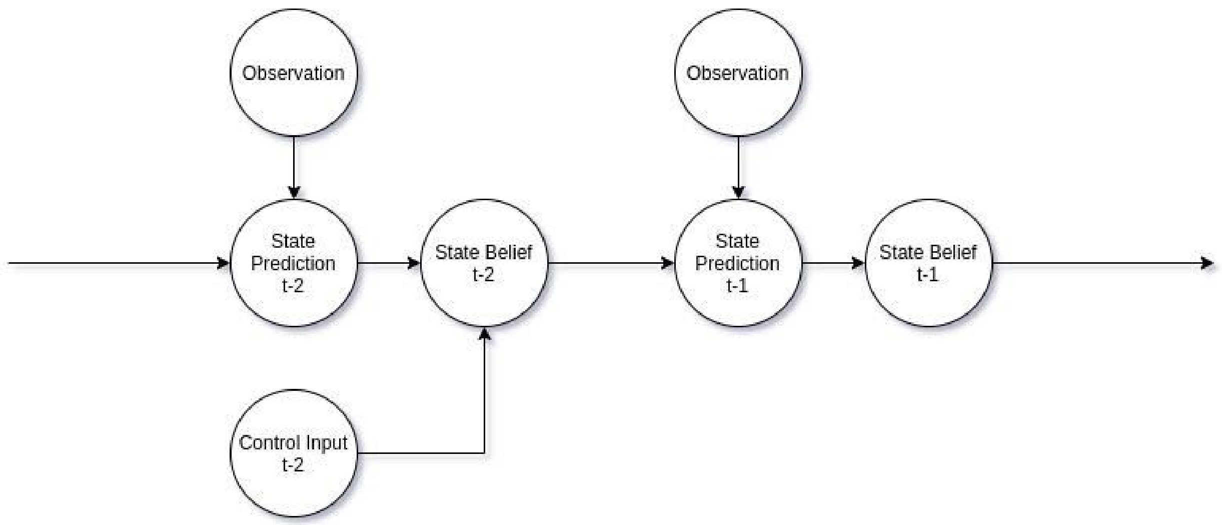


Figure 5: Recursive predict and update cycle for Bayesian filtering

The goal is to estimate the true state x. The state is not directly accessible, but there exist sensor observations of the underlying Hidden Markov Model. This means that the true states are not observed but another process which is dependent on true states is observed. By using a motion model, control inputs, sensor observations, estimate of sensor noise and process noise; a state estimate and belief in that estimate can be computed. A special case of Bayes filter called particle filter is used in this research.

## 2.3 Sensor Noise Model

Sensors are devices that observe some physical attribute in the environment and convert that into a reading or a digital signal that can be easily interpreted. There are various types of sensors that can broadly be categorized into active and passive sensors. Sensors that send energy out into the environment are called active, eg: Sonar. Sensors that only observe environmental signals, such as cameras, are called passive sensors.

Sensors are noisy. Noise is the algebraic difference between observed value and true value of observed variable. For a value of ground truth, the sensor may return a different value each time. For example, for a true distance of 5, the sensor may return 5.05, 4.99, 5, etc. The difference between the sensor value and the true value is sensor error. This error can be computed for different ground truth labels and a distribution can be estimated from which these error values are drawn. This estimated noise distribution is called the sensor noise model.

**Observation = true state + random value from noise distribution**

We need an estimate of this noise distribution to update our belief in predicted state.

The process of modelling the sensor noise involves predicting the true distribution parameters from which the noise is generated. Most current methods for doing this require the knowledge of ‘ground truth’ labels for sensor data and require sensor specific calibration targets. For example to calibrate a lidar, specialized spheres are used; to calibrate cameras, special camera calibration devices are needed.

The user supplied noise models directly affects the accuracy of state estimates. In this research, the algorithm changes the value of this user supplied noise model to compare noise consistency in the predicted states.

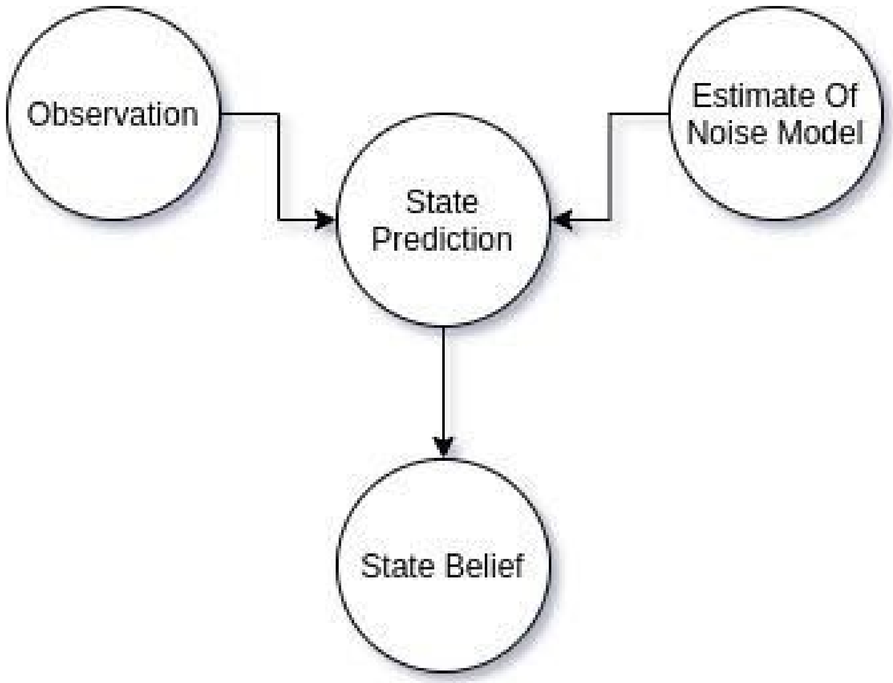


Figure 6: Assumption of sensor noise model along with the actual observations are used to compute the belief in state.

## 2.4 KL Divergence

KL divergence*[2]* measures how different one distribution is from another. A KL divergence of zero implies that both the distributions are the same. Intuitively, this is a measure of area under the curve of distribution 1 not covered by the curve of distribution 2.

Consider two probability distributions P and Q. P represents the true distribution from which data is sampled. Q represents an approximation of the true distribution. KL divergence is then interpreted as the difference between the number of bits required to encode samples of P using a code optimized for Q.

$$\sum_{x \in X} P(x) * log(P(x) - Q(x))$$

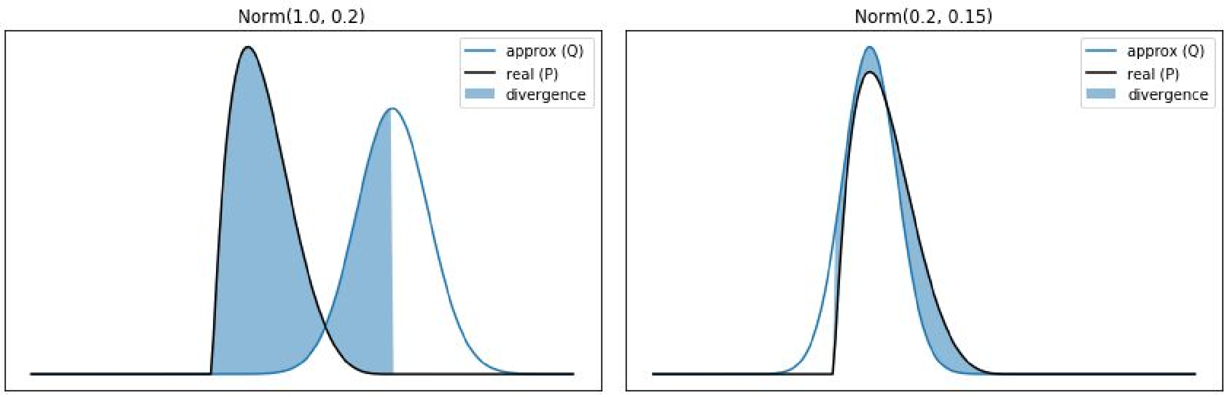


Figure 7: KL divergence between distributions represented as area under the curve. Area in blue represents the area difference

# Chapter 3

# PROPOSED APPROACH

In this research, a novel method for sensor calibration is introduced, in which the need for sensor specific calibration targets is eliminated. By statistically analyzing sensor data and computing noise consistency metrics, an approximation for true noise distribution is achieved. For each sensor being calibrated, observations for identifiable targets in the environment are used to calibrate the sensors against each other. A noise consistency metric based on KL divergence is used. This algorithm can be run periodically to update the sensor noise model.

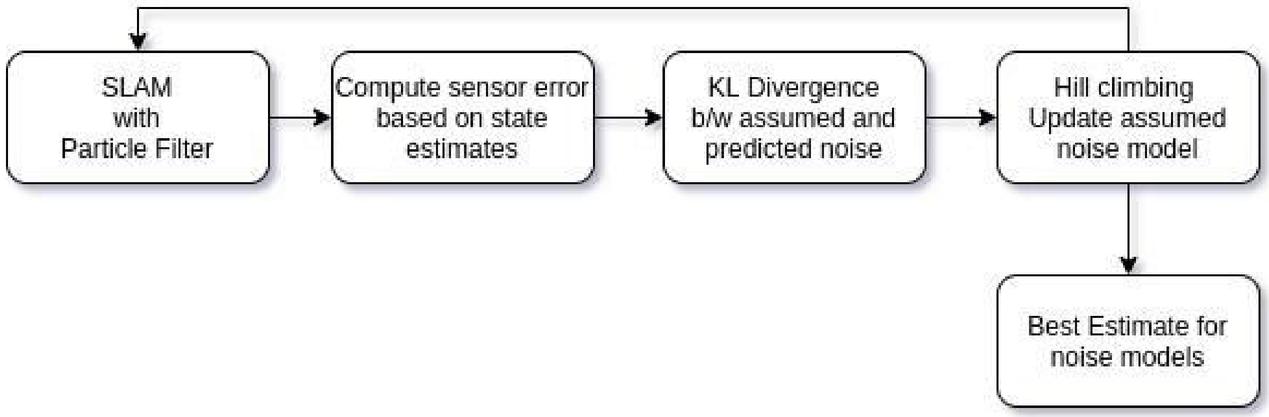

SLAM with Particle Filter
Compute sensor error based on state estimates
KL Divergence b/w assumed and predicted noise
Hill climbing Update assumed noise model
Best Estimate for noise models

Figure 8: Proposed architecture overview

## 3.1 SLAM with Particle Filter

When operating in an unknown environment, the robot has no knowledge about its own location or the environment map. By using the sensor observations and by moving around in the world, the robot will incrementally create a map estimate and localize itself within that map. This task of simultaneously doing two tasks is known as SLAM( simultaneous localization and mapping)

The motion of the robot can be predicted using a motion model. Motion model captures the motion of a robot in a mathematical equation. Given a previous location, and a control input to move, the motion model predicts where the robot is. Assuming the robot started at origin(0, 0) , and a control input to move by (1, 1) , the state is predicted as follows:

**New_state = old_state + control_input + process_noise**

Process noise can be caused due to multiple reasons. Including but not limited to , uneven surfaces, difference in tire size, motor power, battery level etc. Due to process noise, we can assume that the robot may never execute control inputs with perfection, hence the state prediction needs to take this process noise into consideration.

Formally, the motion model of the robot is given by:

**x(t) = A*x(t-1) + B*u + w.**

A is the state transition model, B is the control input model applied to control input u, w is the process noise from a normal distribution with zero mean and covariance $\mathcal{N}(0, Q)$

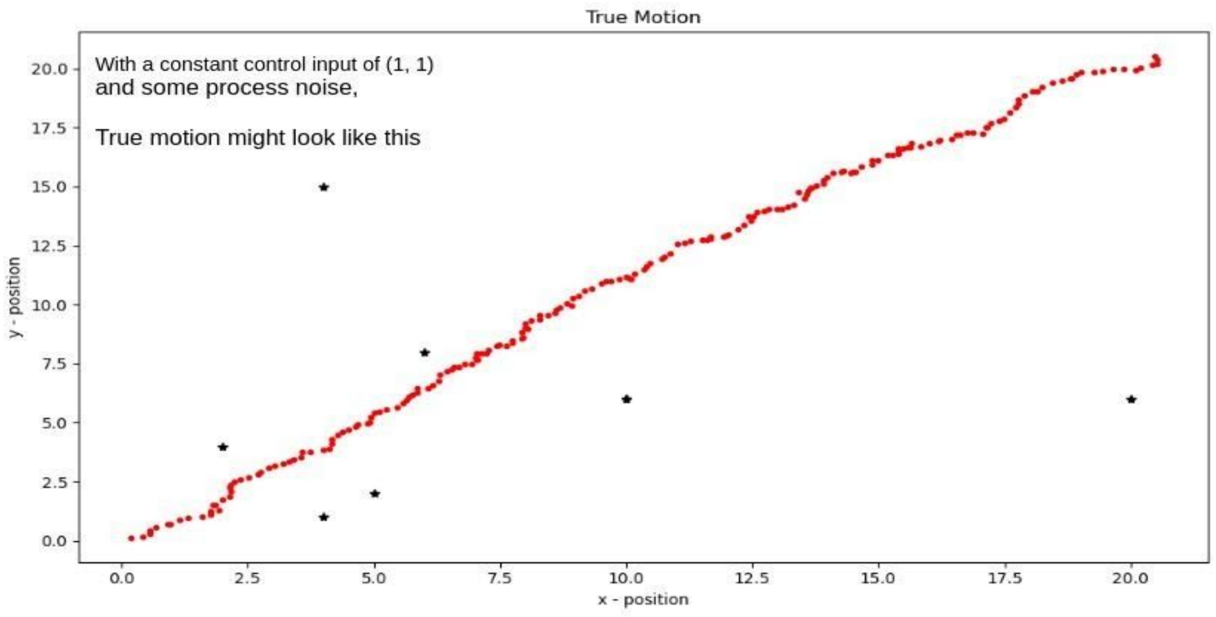


Figure 9: What the trajectory may look like given some process noise

In a particle filter, at the start, when predicting the next state, multiple predictions are made. Each prediction assumes a slightly different value for process noise. Each prediction is called a particle. The number of particles is decided by the user. For example,

Robot starts at origin - (0, 0)

Apply control input of - (1, 1)

Process noise - mean(0, 0) and cov[(0.1, 0), (0, 0.2)]

N state predictions - A*(0, 0) + B(1, 1) + N samples from process noise.

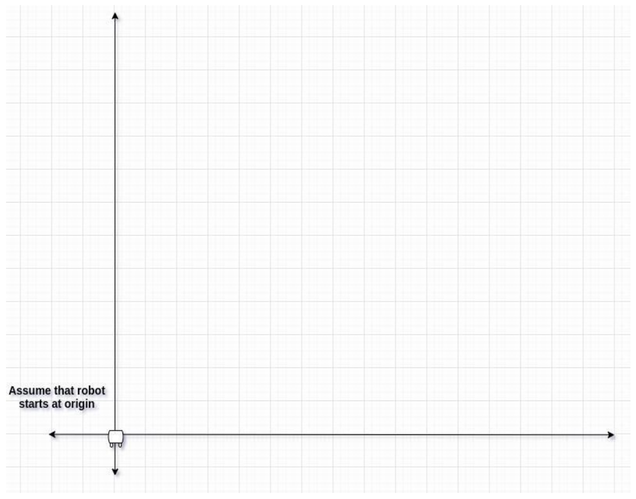


Figure 10: Robot Spawn

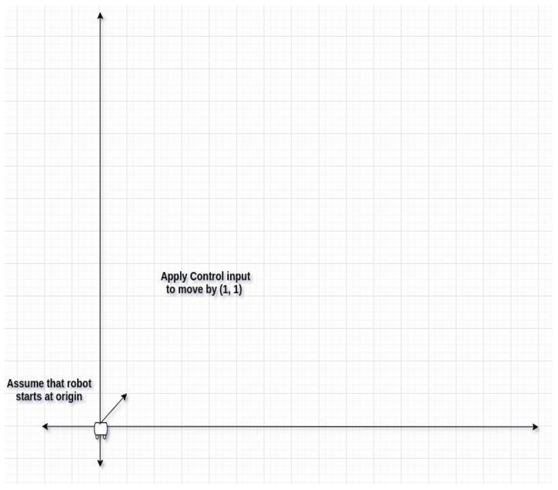
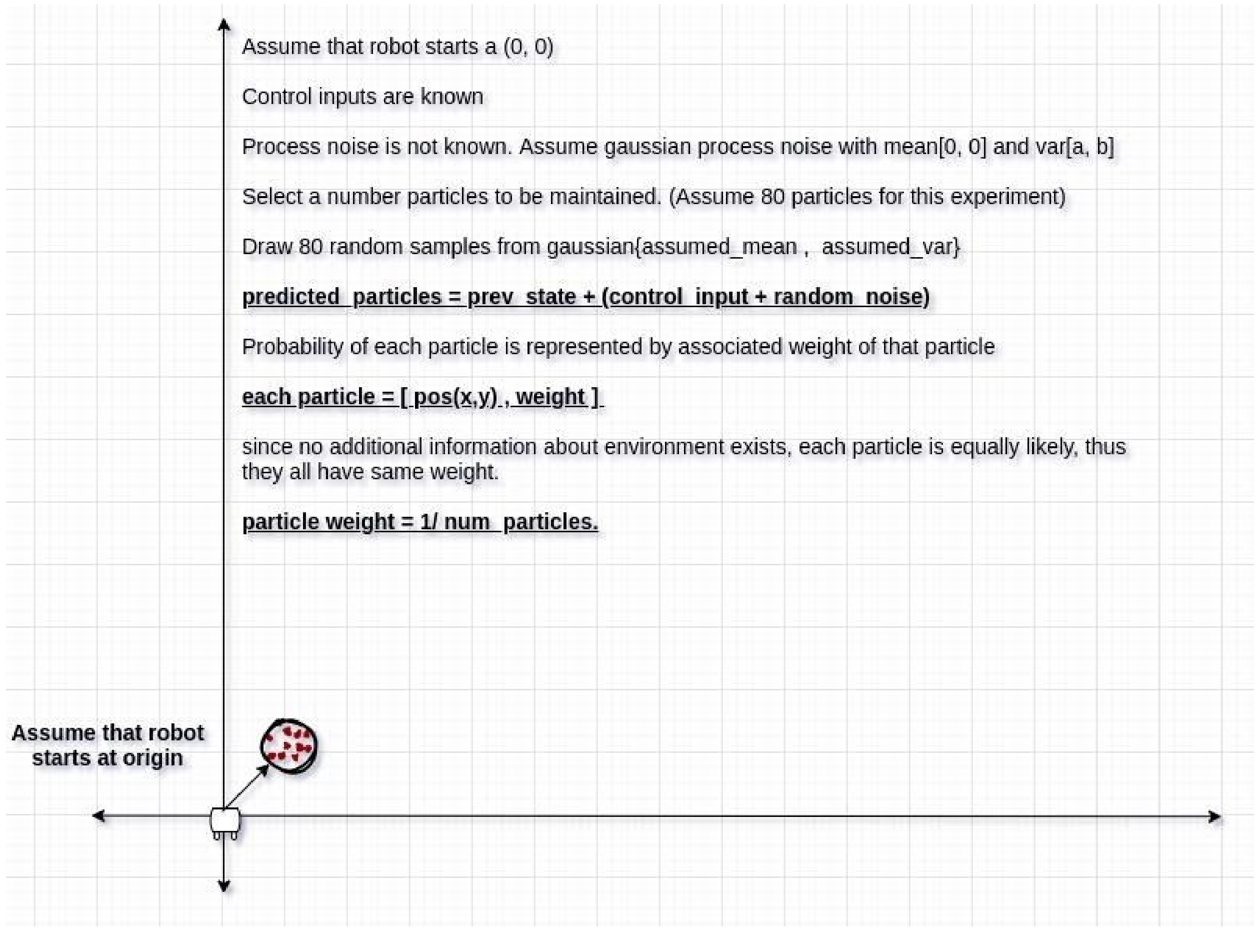


Figure 11: Control input to robot



Assume that robot starts a (0, 0)

Control inputs are known

Process noise is not known. Assume gaussian process noise with mean[0, 0] and var[a, b]

Select a number particles to be maintained. (Assume 80 particles for this experiment)

Draw 80 random samples from gaussian{assumed_mean , assumed_var}

**predicted_particles = prev_state + (control_input + random_noise)**

Probability of each particle is represented by associated weight of that particle

**each particle = [ pos(x,y) , weight ]**

since no additional information about environment exists, each particle is equally likely, thus they all have same weight.

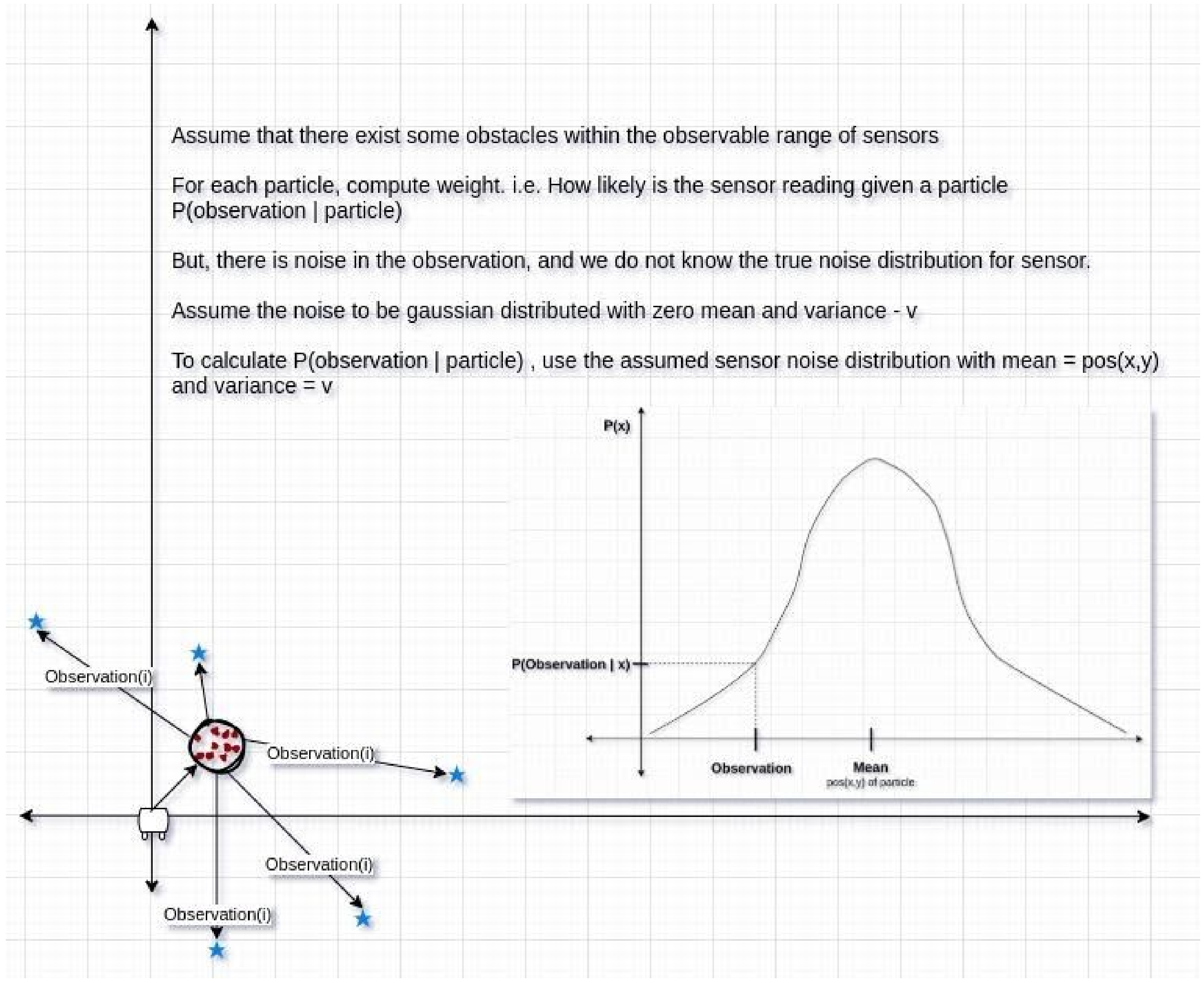
**particle weight = 1/ num_particles.**

Figure 12: Particle state and weight prediction step

Each of the predicted states is assigned a weight. The weight represents the probability of that particle occurring. In the absence of any observations or information about the environment, each particle is assigned the same weight. These weights will be updated when we receive a sensor reading and we know something about the environment.

As soon as the robot receives an observation from the sensor, the probability of particles will change. Depending on the reading, the robot can compute the likelihood of being at a position given some estimate of the environment and a sensor reading.

Likelihood is calculated by computing the probability of observation given a state estimate P(observation | particle). For a given state estimate (x, y), the observation can be in the range: State ± Sensor noise. Intuitively, this represents a Gaussian with state estimate as mean and sensor noise as variance. This process is explained in the figure below:


Assume that there exist some obstacles within the observable range of sensors

For each particle, compute weight. i.e. How likely is the sensor reading given a particle P(observation | particle)

But, there is noise in the observation, and we do not know the true noise distribution for sensor.

Assume the noise to be gaussian distributed with zero mean and variance - v

To calculate P(observation | particle) , use the assumed sensor noise distribution with mean = pos(x,y) and variance = v

Figure 13: Updating weights based on observations

The robot moves in the environment based on control inputs and keeps updating the state estimate and the map estimate based on the observations. Note that the weights of particles depend heavily on the assumption of sensor noise. An assumption of noise lower than the true noise will result in a lower weight for particles which deserve a higher weight or higher weight for ones that deserve a lower weight.

This effect can be observed in the figure below. For a gaussian with mean as the position estimate and assumed noise parameter as the variance vs true noise parameters, the probability of getting a sensor reading will vary.

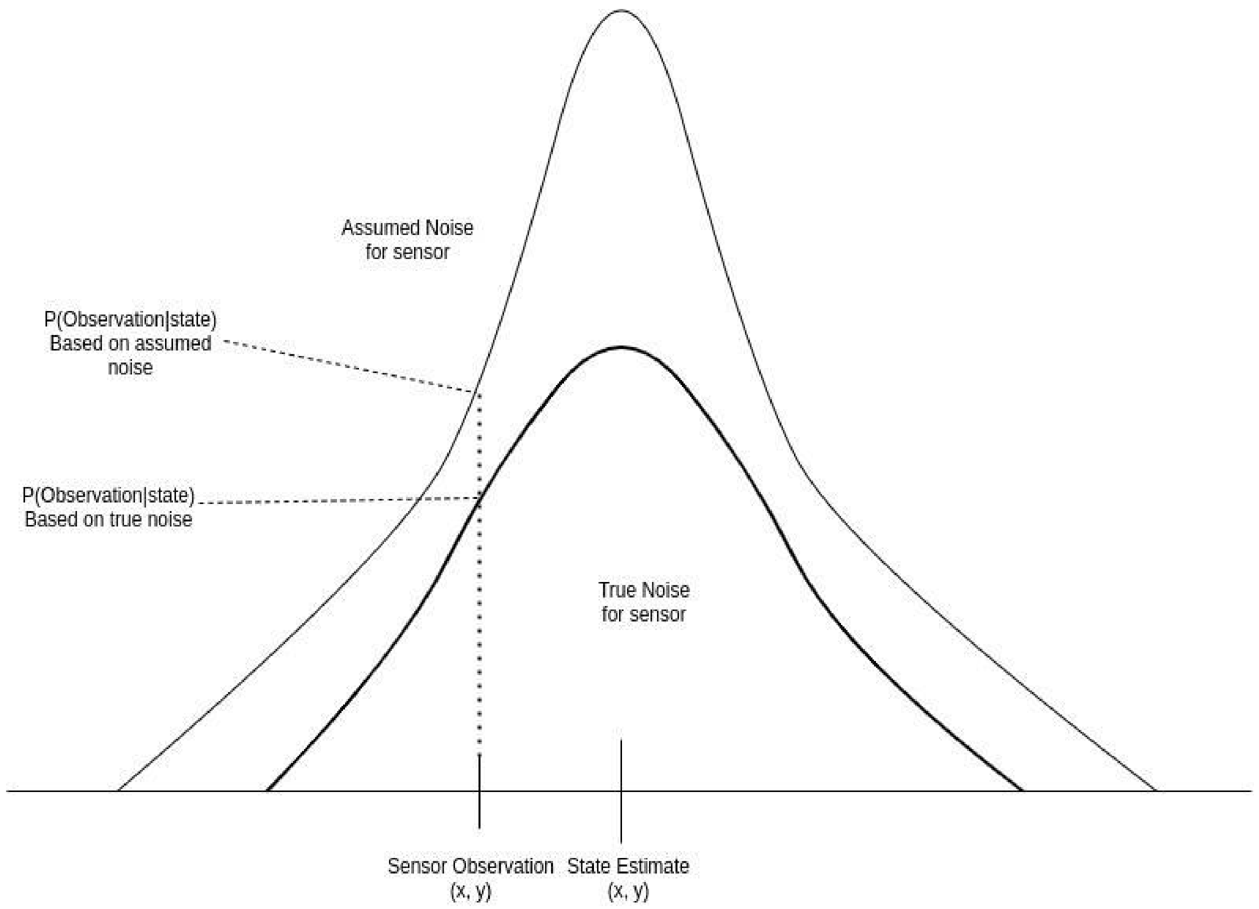


Figure 14: Probability of observation varies depending on assumption of noise model

After this step, the robot will have multiple state estimates for a given timestep. Each estimate will have an associated weight. For most practical applications, there needs to be one estimate for each timestep. To compute one state estimate, the weighted sum of all possible states is used.

$$\sum_{n=1}^{nParticles} w * pos(x, y)$$

State Estimate =

The particle filter recursively performs these steps to get a state estimate for each timestep.

Particle filter takes in:

| | |
|---|---|
| $z_1, z_2$ | - observations from sensor1, sensor2 |
| $S_{N1}, S_{N2}$ | - assumed noise parameters for both sensors |

Outputs:

N particles - $\hat{S}_t$ (state estimates $s_i$ , with their weights $w_i$)

$$PF(z_1, z_2, S_{N1}, S_{N2}) = \hat{S}_t = \begin{bmatrix} s_0, s_1, \ldots\ldots, s_n \\ w_0, w_1, \ldots\ldots, w_n \end{bmatrix}$$

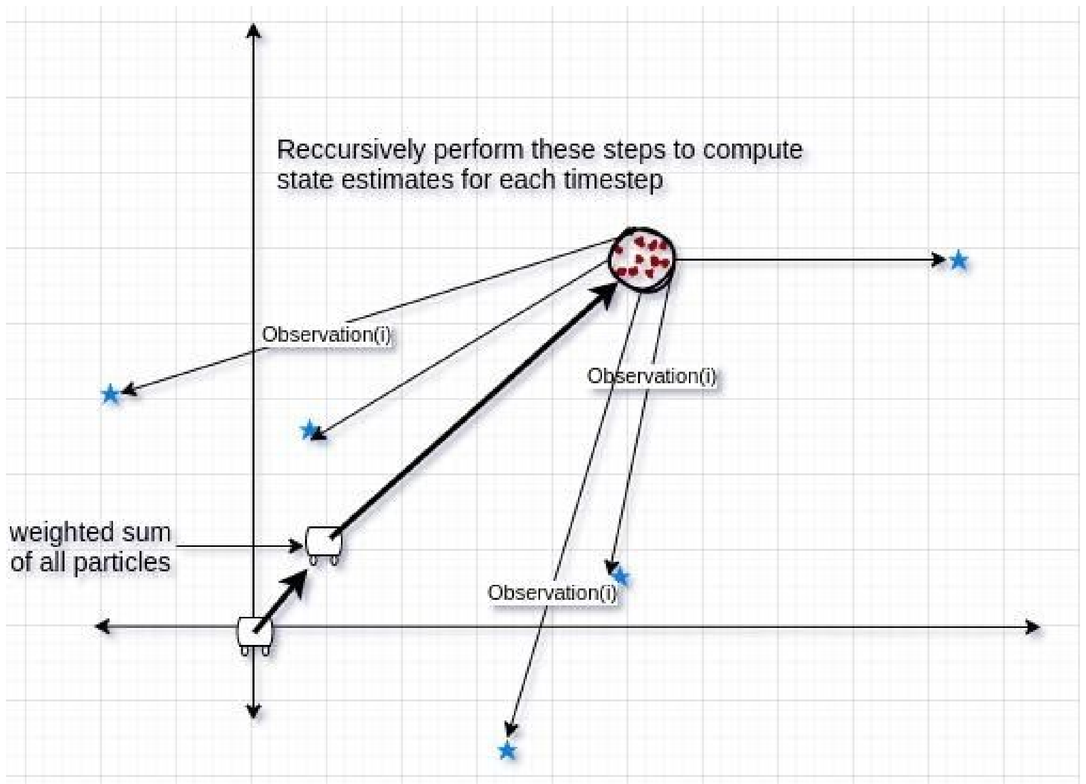

Reccursively perform these steps to compute state estimates for each timestep
Observation(i)
Observation(i)
weighted sum of all particles
Observation(i)

Figure 15: Recursive predictions and updates to estimate the trajectory

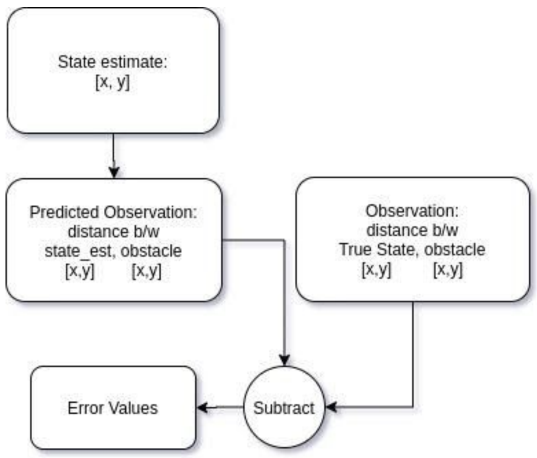
## 3.2 Compute Sensor Error

Sensor error is the algebraic difference between the observation corresponding to the true state value and the reading returned by sensor. For this experiment, the sensor error is the difference between actual sensor reading and the sensor state estimate. The state is represented as cartesian coordinates ( x, y) whereas the sensor reading is for a distance sensor, such as a LIDAR sensor and thus is in terms of distance to unknown obstacles in the world. Hence, this difference cannot be calculated without some transformations representing the relation between the state and the sensor reading. For this, state estimates are converted into predicted sensor readings. Predicted sensor readings, $z_i$, are here the distance between state estimate( x, y ) and landmark position estimate(x, y). Since both are now in the same state space, their difference can be computed.

$$\widehat{err}_1 = \hat{S}_t - z_{1_i} = \begin{bmatrix} \widehat{e_0}, \widehat{e_1}, \ldots\ldots, \widehat{e_n} \\ w_0, w_1, \ldots\ldots, w_n \end{bmatrix} \quad \forall\, i\,, \widehat{e_i} = s_i - z_{1_i}$$

$$\widehat{err}_2 = \hat{S}_t - z_{2_i} = \begin{bmatrix} \widehat{e_0}, \widehat{e_1}, \ldots\ldots, \widehat{e_n} \\ w_0, w_1, \ldots\ldots, w_n \end{bmatrix} \quad \forall\, i\,, \widehat{e_i} = s_i - z_{2_i}$$

err = state_estime_transformed_to_sensor_reading - z



State estimate: [x, y]
Predicted Observation: distance b/w state_est, obstacle [x,y] [x,y]
Observation: distance b/w True State, obstacle [x,y] [x,y]
Subtract
Error Values

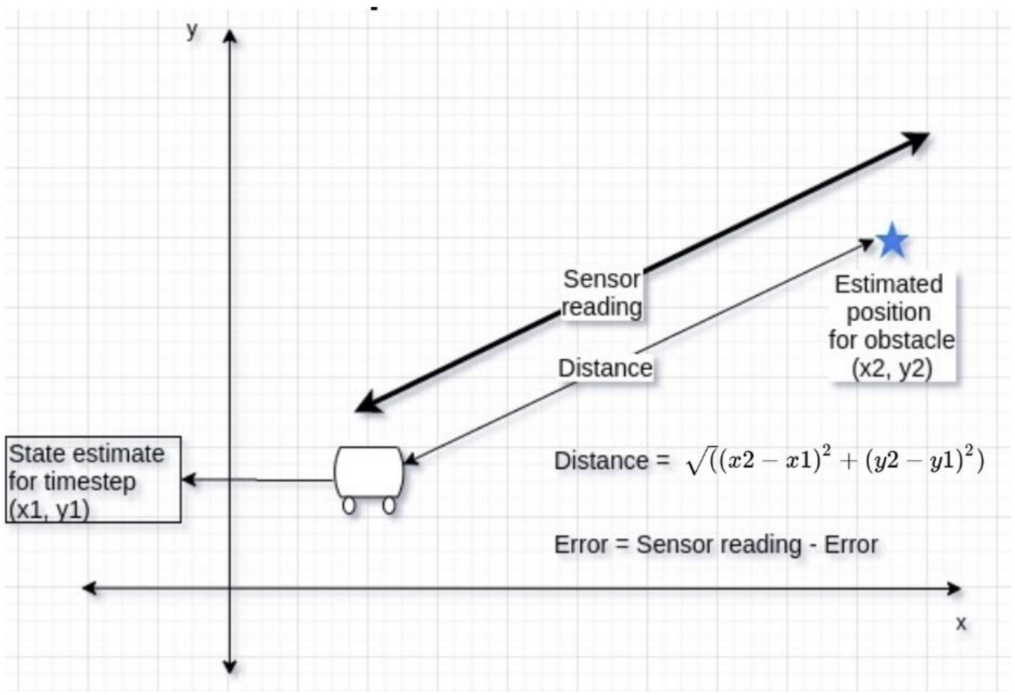

y
x
Sensor reading
Distance
Estimated position for obstacle (x2, y2)
State estimate for timestep (x1, y1)
Distance = $\sqrt{((x2-x1)^2+(y2-y1)^2)}$
Error = Sensor reading - Error

Figure 16: Computing error between actual and predicted observations

An error value for each particle (state estimate) is computed against each sensor. This results in n error values and associated weights. The total error value of a timestep is given by the weighted sum of all errors. This is represented in the figure below:

$$predicted\ state = position(x1, y1)$$

$$predicted\ map = landmark\ position(x2, y2)$$

$$predicted\ sensor\ reading = distance(predicted\ state,\ predicted\ map)$$

$$distance = \sqrt{((x2-x1)^{2\,\hat{}} + (y2-y1)^2)} \quad predicted\ error = true\ sensor\ reading - predicted\ sensor\ reading$$

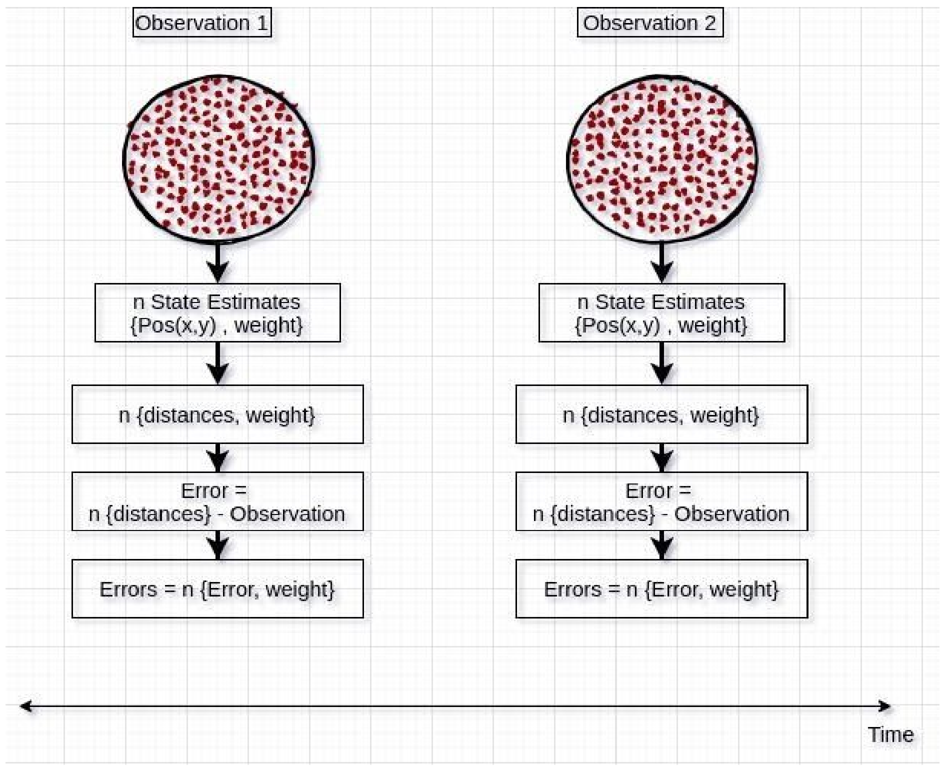

Observation 1
n State Estimates {Pos(x,y) , weight}
n {distances, weight}
Error = n {distances} - Observation
Errors = n {Error, weight}
Observation 2
n State Estimates {Pos(x,y) , weight}
n {distances, weight}
Error = n {distances} - Observation
Errors = n {Error, weight}
Time

Figure 17: Recursive prediction and updates to estimate the trajectory

## 3.3 KL Divergence

After computing a list of discrete error values for all sensors, the robot needs to analyze how likely these error values are given the knowledge of assumed parameters for sensor noise. The list of discrete error values can be treated as a sample set drawn from the true noise distribution, but biased by the assumption of the noise distribution. Assuming that the error samples are drawn from a Gaussian distribution, its probability density is calculated. As depicted in Figure 14, the total distance between the assumed and the predicted sensor noise models is dependent on the assumed noise parameters. Hence by changing these values total distance can be manipulated.

Assuming we use two sensors and that the senor noise models are gaussian for both, the total distance between assumed and predicted sensor noise models for all sensors can be computed using KL divergence:

$$PDF\,(assumed_i) = \frac{1}{\sigma_{a_i}\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma_{a_i}}\right)}$$

$$PDF\,(predicted_i) = \frac{1}{\sigma_{p_i}\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma_{p_i}}\right)}$$

$p1 = PDF\,(assumed_1)\,,\ p2 = PDF(assumed_2)$

$q1 = PDF\,(predicted_1)\,,\ q2 = PDF(predicted_2)$

$KL1(p1||q1) = \sum_{x \in X} p1(x) * (\log p1(x) - \log q1(x))$

$KL2(p2||q2) = \sum_{x \in X} p2(x) * (\log p2(x) - \log q2(x))$

$TotalDistance = KL1 + KL2$

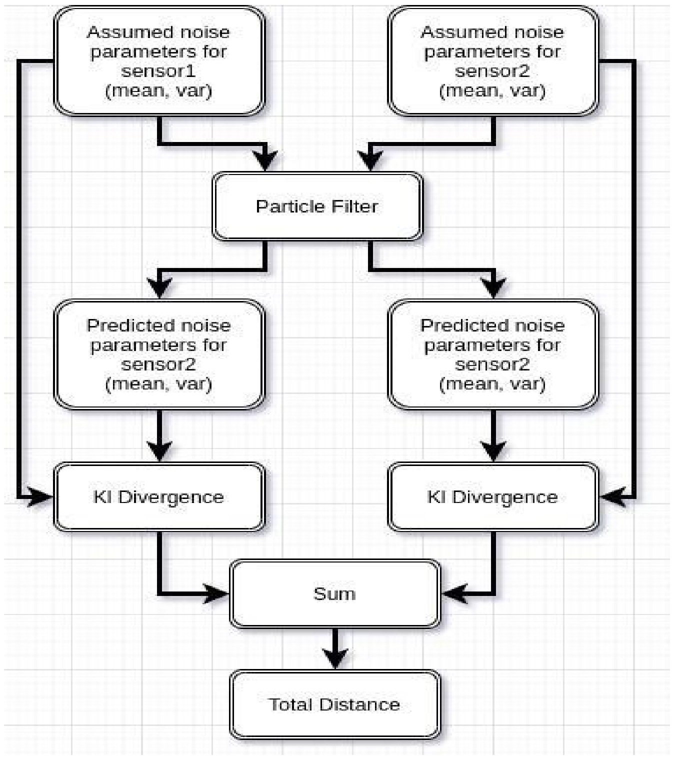

Assumed noise parameters for sensor1 (mean, var)
Assumed noise parameters for sensor2 (mean, var)
Particle Filter
Predicted noise parameters for sensor2 (mean, var)
Predicted noise parameters for sensor2 (mean, var)
Kl Divergence
Kl Divergence
Sum
Total Distance

Figure 18: Algorithm Recap

## 3.4 Hill Climbing

If the assumed noise model for each of the sensors is the correct noise model (i.e. if the sensors are correctly calibrated), we would expect that the predicted (observed) sensor noise would follow the same distribution and thus that the total distance (the KL divergence between the assumed and predicted distributions) is small. To achieve this, we need to update the assumed sensor noise model such that the total distance is minimized (and thus the noise models are consistent).

Hill climbing is an iterative algorithm to find a solution for an optimization problem. It starts with a random solution to the problem and computes the value of the function by making small changes to input parameters. When a parameter returns a better solution to the problem the value for the best solution is updated. This process is executed iteratively until a global/local minima is found.

In the sensor calibration problem, the assumed noise parameters serve as the input to the optimization problem. A change in these parameters will result in a change in total KL divergence. The parameters are sensor1(mean, var) , sensor2(mean, var). 4 neighbors are computed for each parameter, hence the algorithm computes KL Divergence for: sensor1(mean+step, var);

sensor1(mean-step, var);

sensor1(mean, var+step);

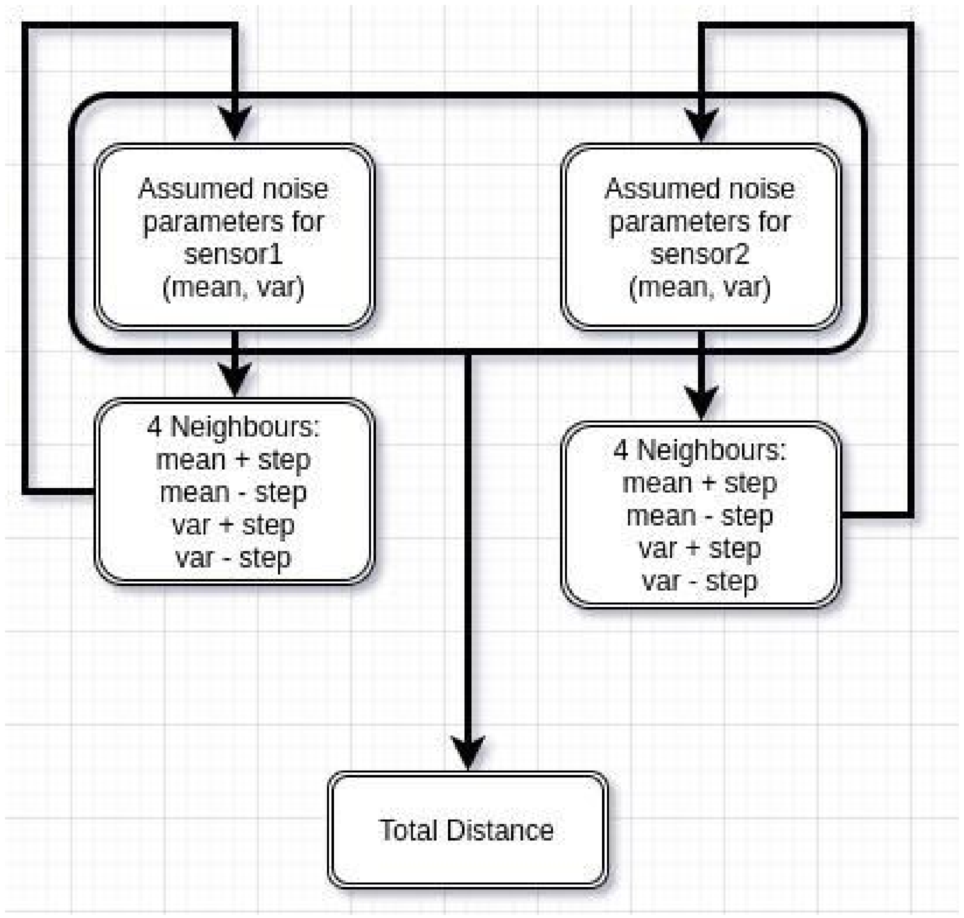
sensor1(mean, var-step);

sensor2(mean+step, var);

sensor2(mean-step, var);

sensor2(mean, var+step);

sensor2(mean, var-step)


Assumed noise parameters for sensor1 (mean, var)
Assumed noise parameters for sensor2 (mean, var)
4 Neighbours: mean + step mean - step var + step var - step
4 Neighbours: mean + step mean - step var + step var - step
Total Distance

If total distance is lower for any of the neighbors, that set of parameters will become the best solution and the algorithm explores the distance for its neighbors. When the algorithm gets stuck in a minima, it makes a random jump in an attempt to break free. Higher numbers of random jumps will result in a higher probability of breaking free but also increases the computation time.

If after a few random jumps, the value of best parameters does not change, that set of parameters is accepted as the solution.

To test the validity of this method, several experiments are conducted. The results from these experiments are discussed in Chapter 5.

## 3.5 Algorithm Summary

In this algorithm, a trajectory is generated for the robot. This trajectory is based on the control inputs. The true states are generated using the control inputs and a motion model. In the real world, there always exists some process noise in the motion of the robot and thus the same control inputs can result in different trajectories. The true trajectory states are generally not observable by the robot but will be estimated as part of the Bayesian filtering process. In our experiments we are using simulation data for this experiment, and we thus use random noise parameters to generate true states. These true states are used to generate true sensor readings based on a true sensor noise model as part of the simulation. Using this sequence of sensor readings, the proposed approach uses initial assumed sensor models to estimate the distribution of trajectories using a particle filter, computes a predicted sensor noise distribution from this trajectory distribution and the sensor readings, and then updates the assumed sensor noise model using a hill climbing step on the total distance of the assumed and the predicted sensor noise models for all the sensors. This improvement process is repeated until no further improvement can be achieve and thus until the most consistent sensor noise models have been found. More detailed, the algorithms goes through the following steps:

**Prediction Step:** Based on known control inputs, an assumed process noise and a motion model, robot states are predicted.

**Update Step:** Based on the predicted state, an assumed noise model and sensor readings, the belief in that predicted state is updated. The algorithm computes how likely the robot is at a predicted location given a sensor reading. The likelihood of a robot being in a state is dependent on the belief of how much noise exists in the sensor reading or in this case, the assumed noise model.

**Compute Expected Sensor Readings:** Now, there exists a list of state estimates for each timestep and an estimate of the map (landmark locations). Based on these estimates, the algorithm computes expected sensor readings for each timestep and sate estimate. Note that there also exists a list of actual sensor readings. If the assumed sensor noise is the same as true sensor noise, then the sensor error distributions based on the state predictions is consistent with the original assumption and thus the assumed sensor noise models are consistent. In the case where assumed noise parameters are not the same as true noise parameters, the predicted states are not consistent with the noise assumptions and thus the noise assumptions should be adjusted to yield more consistent estimates. The predicted sensor error values can be interpreted as samples drawn from the true noise distribution , but biased by the assumed noise distribution.

**Compute Distribution Parameters:** Parameters are computed for the distribution from which these predicted error values are most likely to be drawn. The algorithm already knows the value for assumed noise parameters.

**Compute KL Divergence:** Divergence between the assumed error distribution and the predicted error distribution gives us a measure of how far we are from the true noise distribution. The goal is to minimize this distance.

**Noise model optimization:** The assumed noise parameters are changed recursively and the whole algorithm is run again. If a lower value of KL Divergence is found for some assumed noise parameters, then the values for best parameters are updated until no smaller value can be found.

In the results, in each iteration, the algorithm computes 4 neighbors for both the assumed noise parameters. It only prints KL divergence values if a lower value is found. A list of KL divergence is updated every time a lower value is found. At the end of program execution, the KL divergence values are plotted.

# Chapter 4

# DRY RUN

In this section we will assume some random values for our parameters and try to see how the algorithm pushes the parameters towards true noise distribution. Let's assume this case for sensor 1:

True Noise: Mean(0, 0) Var(0.2, 0.1)
Iteration 1:
Assumed Noise: Mean(0, 0) Var(0.6, 0.8)

| True State | (0, 0) | (1, 1) | (2, 2) |
|---|---|---|---|
| True observation | 10 | 12 | 14 |
| Predicted state | (0, 0) | (1.02, 0.97) | (2.6, 1.91) |
| Predicted observation | 9.5 | 11.4 | 14.4 |
| Predicted error | 0.5 | 0.6 | -0.4 |

Predicted Noise: Mean(0, 0) Var(0.4, 0.6)
KLDiv(assumed noise, predicted noise) = 4
Assumed parameters are far away from true noise and hence we see a large value for kl divergence.

--------------------------------------------------------------------------------

True Noise: Mean(0, 0) Var(0.2, 0.1)
Iteration 2:
Assumed Noise: Mean(0, 0) Var(0.4, 0.3)

| True State | (0, 0) | (1, 1) | (2, 2) |
|---|---|---|---|
| True observation | 10 | 12 | 14 |

| | | | |
|---|---|---|---|
| Predicted state | (0, 0) | (1.01, 0.99) | (1.8, 2.01) |
| Predicted observation | 9.8 | 12.5 | 14.4 |
| Predicted error | 0.2 | 0.5 | -0.4 |

Predicted Noise: Mean(0, 0) Var(0.3, 0.5)
KLDiv(assumed noise, predicted noise) = 2
Assumed parameters are far away from true noise but clo were in the last iteration. Hence we see a smaller value for kl divergence.
-------------------------------------------------------------------------------------
We observe the same trend for sensor 2.

Assume this case for sensor 2:
True Noise: Mean(0, 0) Var(0.2, 0.1)
Iteration 1:
Assumed Noise: Mean(0, 0) Var(0.4, 0.3)

| True State | (0, 0) | (1, 1) | (2, 2) |
|---|---|---|---|
| True observation | 10 | 12 | 14 |
| Predicted state | (0, 0) | (1.01, 0.99) | (1.8, 2.01) |
| Predicted observation | 9.8 | 12.5 | 14.4 |
| Predicted error | 0.2 | 0.5 | -0.4 |

Predicted Noise: Mean(0, 0) Var(0.3, 0.5)
KLDiv(assumed noise, predicted noise) = 1.5
-------------------------------------------------------------------------------------
True Noise: Mean(0, 0) Var(0.2, 0.1)
Iteration 2:
Assumed Noise: Mean(0, 0) Var(0.6, 0.8)

| True State | (0, 0) | (1, 1) | (2, 2) |
|---|---|---|---|

|  |  |  |  |
|---|---|---|---|
| True observation | 10 | 12 | 14 |
| Predicted state | (0, 0) | (1.02, 0.97) | (2.6, 1.91) |
| Predicted observation | 9.5 | 11.4 | 14.4 |
| Predicted error | 0.5 | 0.6 | -0.4 |

```
Predicted Noise:                Mean(0, 0) Var(0.4, 0.6)
KLDiv(assumed noise, predicted noise) = 3
True Noise Sensor1:                  Mean(0, 0) Var(0.2, 0.1)
Assumed Noise Sensor1:               Mean(0, 0) Var(0.4, 0.3)
Predicted Noise Sensor1:             Mean(0, 0) Var(0.3, 0.5)
```
**KL1**(assumed noise, predicted noise) = 4

```
True Noise Sensor2:                  Mean(0, 0) Var(0.2, 0.1)
Assumed Noise Sensor2:               Mean(0, 0) Var(0.6, 0.8)
Predicted Noise Sensor2:             Mean(0, 0) Var(0.3, 0.5)
```
**KL2**(assumed noise, predicted noise) = 1.5
--------------------------------------------------------------------------------

**Total KL = KL1 + KL2 = 5.5**

Update assumed parameters and compute total KL divergence again
------------------------------------------------------------------------------

```
True Noise Sensor1:                  Mean(0, 0) Var(0.2, 0.1)
Assumed Noise Sensor1:               Mean(0, 0) Var(0.6, 0.8)
Predicted Noise Sensor1:             Mean(0, 0) Var(0.4, 0.6)
KLDiv(assumed noise, predicted noise) = 3

True Noise Sensor2:                  Mean(0, 0) Var(0.2, 0.1)
Assumed Noise Sensor2:               Mean(0, 0) Var(0.4, 0.3)
Predicted Noise Sensor2:             Mean(0, 0) Var(0.3, 0.5)
KLDiv(assumed noise, predicted noise) = 2
```

**Total KL = KL1 + KL2 = 5.0**

Lower value found for KL Divergence. This implies that the assumption for sensor noise is better than the previous one. We continue these steps until there is no lower value found for kl divergence.

# Chapter 5

# EXPERIMENT AND RESULTS

To validate the algorithm, several experiments are conducted and their results compared against the expected output. Different values for hyperparameters are tested and the KL Divergence graph is analyzed.

## 5.1 Experiment 1 Details

In this experiment, we use a small process noise to generate a true trajectory. These true states are used to generate true sensor readings based on a true sensor noise model. Then , based on an assumed sensor noise, control inputs and an assumed process noise, robot states are predicted and updated. These states are then used to calculate predicted sensor readings. The assumed noise model will affect the predicted sensor readings. These predicted sensor readings are used to calculate the error values in true sensor readings. Parameters are computed for the distribution from which these predicted error values are most likely to be drawn. KL Divergence between the assumed error distribution and predicted error distribution gives us a measure of how far we are from the true noise distribution. The goal is to minimize this distance. The assumed noise parameters are changed and the whole algorithm is run again. If a lower value of KL Divergence is found for some parameters, then the values for best parameters are updated until no smaller value can be found.

In the results, in each iteration, the algorithm computes 4 neighbors for both the assumed noise parameters. It only prints KL divergence values if a lower value is found.

A list of KL divergence is updated every time a lower value is found. At the end of program execution, the kl divergence values are plotted.

- Robot’s state space
    - cartesian space
    - i observable landmarks at position [xi, yi]
    - state - [x, y]
- Observation space
    - distance in cartesian space
    - dist(state, landmark)
- true process noise
    - mean - [0, 0]
    - var - [0.00, 0.00]
- true noise sensor 1
    - mean - [0, 0]
    - var - [0.3, 0.4]
- true noise sensor 2
    - mean - [0, 0]

        - var - [0.5, 0.6]

- Particle Filter Parameters
    - Number of particles - 80
    - assumed process noise
        - mean - [0, 0], var - [0.00, 0.00]
    - assumed noise sensor 1
        - mean - [0, 0], var - [0.3, 0.4]
    - assumed noise sensor 2
        - mean - [0, 0], var - [0.5, 0.6]
- Hill Climbing Parameters
    - Step size - [0.01, 0.01]
    - Compute 4 neighbours
    - 5 random jumps if stuck in minima

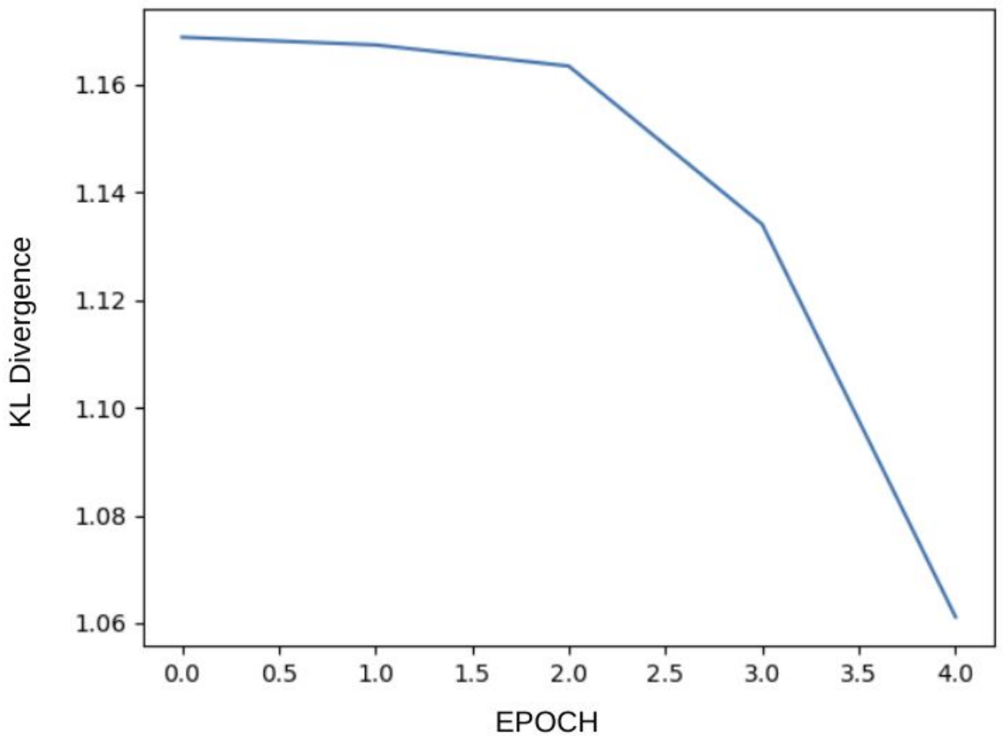
These parameters result in the following result:

```
True_var1 [0.3 0.4] Predicted_var1: [0.3 0.4]

True_var2 [0.5 0.6] Predicted_var2: [0.5 0.6]

Total dist: 1.0983091777875669
```

```
        True_var1 [0.3 0.4] Predicted_var1: [0.31 0.4 ]

        True_var2 [0.5 0.6] Predicted_var2: [0.5 0.6]

        Total dist: 1.0533198017373118


        True_var1 [0.3 0.4] Predicted_var1: [0.31 0.4 ]

        True_var2 [0.5 0.6] Predicted_var2: [0.51 0.6 ]

        Total dist: 1.0066273799027172


Best params after iteration: 1

        True_var1 [0.3 0.4] Predicted_var1: [0.31 0.4 ]

        True_var2 [0.5 0.6] Predicted_var2: [0.51 0.6 ]

        Total dist: 1.0066273799027172


        True_var1 [0.3 0.4] Predicted_var1: [0.31 0.39]

        True_var2 [0.5 0.6] Predicted_var2: [0.51 0.6 ]

        Total dist: 0.9904466820051904


Best params after iteration: 2

        True_var1 [0.3 0.4] Predicted_var1: [0.31 0.39]

        True_var2 [0.5 0.6] Predicted_var2: [0.51 0.6 ]

        Total dist: 0.9904466820051904
        True_var1 [0.3 0.4] Predicted_var1: [0.31 0.39]

        True_var2 [0.5 0.6] Predicted_var2: [0.5 0.6]

        Total dist: 0.9729403231840754


Best params after iteration: 3

        True_var1 [0.3 0.4] Predicted_var1: [0.31 0.39]

        True_var2 [0.5 0.6] Predicted_var2: [0.5 0.6]

        Total dist: 0.9729403231840754


Best params after iteration: 4

        True_var1 [0.3 0.4] Predicted_var1: [0.31 0.39]

        True_var2 [0.5 0.6] Predicted_var2: [0.5 0.6]

        Total dist: 0.9729403231840754

        Found local/global minima. Making a random jump


Best params after iteration: 5

        True_var1 [0.3 0.4] Predicted_var1: [0.31 0.39]

        True_var2 [0.5 0.6] Predicted_var2: [0.5 0.6]

        Total dist: 0.9729403231840754

        Found local/global minima. Making a random jump


Best params after iteration: 6
```

```
            True_var1 [0.3 0.4] Predicted_var1: [0.31 0.39]

            True_var2 [0.5 0.6] Predicted_var2: [0.5 0.6]

            Total dist: 0.9729403231840754

            Found local/global minima. Making a random jump


Best params after iteration: 7

            True_var1 [0.3 0.4] Predicted_var1: [0.31 0.39]

            True_var2 [0.5 0.6] Predicted_var2: [0.5 0.6]

            Total dist: 0.9729403231840754

            Found local/global minima. Making a random jump


Best params after iteration: 8

            True_var1 [0.3 0.4] Predicted_var1: [0.31 0.39]

            True_var2 [0.5 0.6] Predicted_var2: [0.5 0.6]

            Total dist: 0.9729403231840754


Found  local/global  minima.


Process  finished.
```

KL  Divergence  Graph:



With each epoch, the algorithm explores neighbors for the current parameters. The values of parameters are updated only when a lower KL Divergence is found. Once a minimum value is found, the algorithm makes 5 random jumps and exits if no lower value for total distance is found.

In this case, where

true var1: [0.3 0.4]

true var2: [0.5 0.6],

starting from:

var1: [0.3 0.4],

var2: [0.5 0.6],

Total dist: 1.0983091777875669

The algorithm converges to

predicted_var1 [0.31 0.39],

predicted_var2: [0.5 0.6],

Total dist: 0.9729403231840754

This is the expected result. As we initialized true noise to be the same as assumed noise, it should not move.

## 5.2 Experiment 2 Details

We use zero process noise in this experiment. But, the assumed noise parameters are not the same as true noise. The expected result here is for the algorithm to push these assumed parameter values towards the true distribution.

- Robot's state space
  - cartesian space
  - i observable landmarks at position [xi, yi]
  - state - [x, y]
- Observation space

        - distance in cartesian space
        - dist(state, landmark)
    - true process noise
        - mean - [0, 0]
        - var - [0.0, 0.0]
    - true noise sensor 1
        - mean - [0, 0]
        - var - [0.3, 0.4]
    - true noise sensor 2
        - mean - [0, 0]
        - var - [0.5, 0.6]
- Particle Filter Parameters
    - Number of particles - 80
    - assumed process noise
        - mean - [0, 0], var - [0.0, 0.0]
    - assumed noise sensor 1
        - mean - [0, 0], var - [0.4, 0.6]

    - assumed noise sensor 2
        - mean - [0, 0], var - [0.7, 0.9]
- Hill Climbing Parameters
    - Step size - [0.01, 0.01]
    - Compute 4 neighbours
    - 5 random jumps if stuck in minima

These parameters result in the following result:

```
True_var1 [0.3 0.4] Predicted_var1: [0.4 0.6]

True_var2 [0.5 0.6] Predicted_var2: [0.7 0.9]

Total dist: 1.0440139771656285


Best params after iteration: 1

True_var1 [0.3 0.4] Predicted_var1: [0.4 0.6]

True_var2 [0.5 0.6] Predicted_var2: [0.7 0.9]

Total dist: 1.0440139771656285

Found local/global minima. Making a random jump


True_var1 [0.3 0.4] Predicted_var1: [0.399 0.599]

True_var2 [0.5 0.6] Predicted_var2: [0.6 0.8]

Total dist: 1.0440139771656285
```

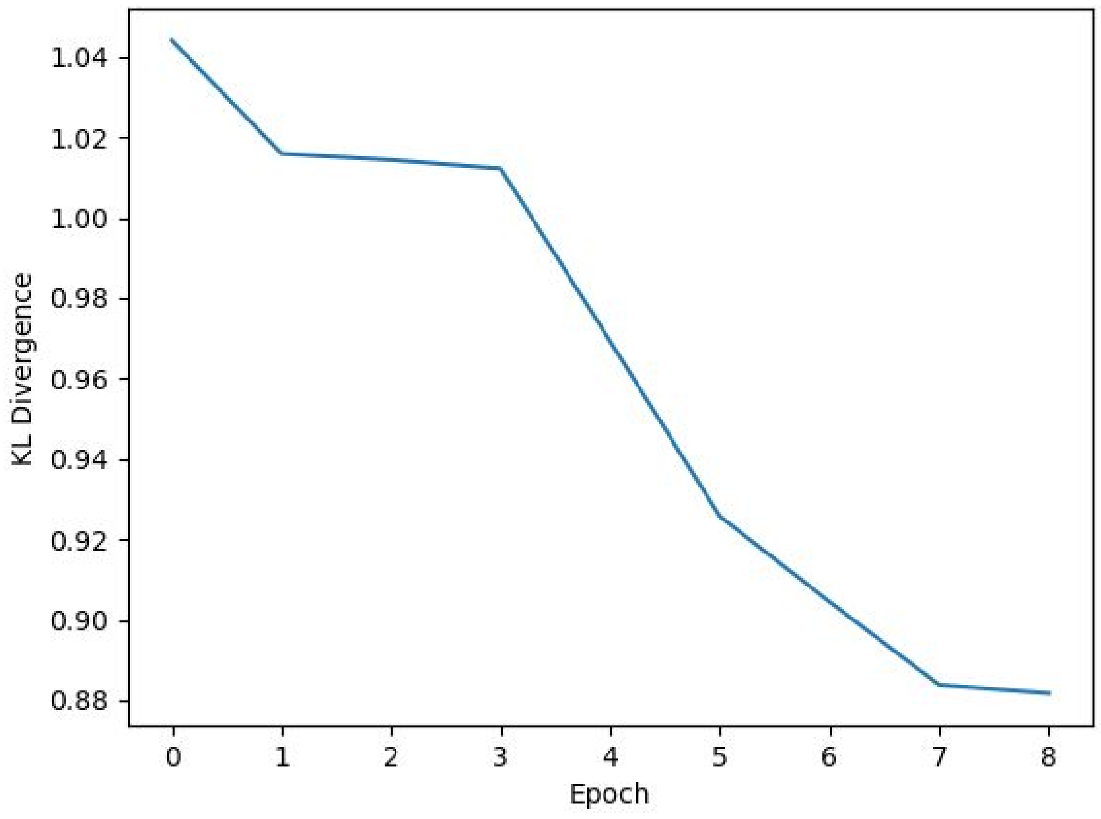
```
Best params after iteration: 2

        True_var1 [0.3 0.4] Predicted_var1: [0.4 0.6]

        True_var2 [0.5 0.6] Predicted_var2: [0.7 0.9]

        Total dist: 1.0440139771656285

Found local/global minima. Making a random jump


        True_var1 [0.3 0.4] Predicted_var1: [0.399 0.599]

        True_var2 [0.5 0.6] Predicted_var2: [0.6 0.8]

        Total dist: 1.0440139771656285

        True_var1 [0.3 0.4] Predicted_var1: [0.29900000000000004, 0.599]

        True_var2 [0.5 0.6] Predicted_var2: [0.6 0.8]

        Total dist: 1.015869536390541


        True_var1 [0.3 0.4] Predicted_var1: [0.29900000000000004, 0.599]

        True_var2 [0.5 0.6] Predicted_var2: [0.6, 0.7000000000000001]

        Total dist: 1.0143010997249888


Best params after iteration: 3

        True_var1 [0.3 0.4] Predicted_var1: [0.29900000000000004, 0.599]

        True_var2 [0.5 0.6] Predicted_var2: [0.6, 0.7000000000000001]

        Total dist: 1.0143010997249888
        True_var1 [0.3 0.4] Predicted_var1: [0.19900000000000004, 0.599]

        True_var2 [0.5 0.6] Predicted_var2: [0.6, 0.7000000000000001]

        Total dist: 1.0121511319180874


        True_var1 [0.3 0.4] Predicted_var1: [0.29900000000000004, 0.499]

        True_var2 [0.5 0.6] Predicted_var2: [0.6, 0.7000000000000001]

        Total dist: 0.96917096948818


Best params after iteration: 4

        True_var1 [0.3 0.4] Predicted_var1: [0.29900000000000004, 0.499]

        True_var2 [0.5 0.6] Predicted_var2: [0.6, 0.7000000000000001]

        Total dist: 0.96917096948818


        True_var1 [0.3 0.4] Predicted_var1: [0.29900000000000004, 0.399]

        True_var2 [0.5 0.6] Predicted_var2: [0.6, 0.7000000000000001]

        Total dist: 0.9256901485206264


        True_var1 [0.3 0.4] Predicted_var1: [0.29900000000000004, 0.399]

        True_var2 [0.5 0.6] Predicted_var2: [0.6, 0.6000000000000001]

        Total dist: 0.9044666632844132
```

```
Best params after iteration: 5

        True_var1 [0.3 0.4] Predicted_var1: [0.29900000000000004, 0.399]

        True_var2 [0.5 0.6] Predicted_var2: [0.6, 0.6000000000000001]

        Total dist: 0.9044666632844132


Best params after iteration: 6

        True_var1 [0.3 0.4] Predicted_var1: [0.29900000000000004, 0.399]

        True_var2 [0.5 0.6] Predicted_var2: [0.6, 0.6000000000000001]

        Total dist: 0.9044666632844132

Found local/global minima. Making a random jump


        True_var1 [0.3 0.4] Predicted_var1: [0.298 0.398]

        True_var2 [0.5 0.6] Predicted_var2: [0.5 0.5]

        Total dist: 0.9044666632844132


        True_var1 [0.3 0.4] Predicted_var1: [0.29900000000000004, 0.399]

        True_var2 [0.5 0.6] Predicted_var2: [0.6, 0.5000000000000001]

        Total dist: 0.8838221398105923


        True_var1 [0.3 0.4] Predicted_var1: [0.29900000000000004, 0.399]

        True_var2 [0.5 0.6] Predicted_var2: [0.5, 0.40000000000000013]

        Total dist: 0.8817247532095723
```

```
Best params after iteration: 7

        True_var1 [0.3 0.4] Predicted_var1: [0.29900000000000004, 0.399]
        True_var2 [0.5 0.6] Predicted_var2: [0.5, 0.40000000000000013]
        Total dist: 0.8817247532095723

Best params after iteration: 8
        True_var1 [0.3 0.4] Predicted_var1: [0.29900000000000004, 0.399]
        True_var2 [0.5 0.6] Predicted_var2: [0.5, 0.40000000000000013]
        Total dist: 0.8817247532095723

Found local/global minima. Making a random jump
        True_var1 [0.3 0.4] Predicted_var1: [0.298 0.398]
        True_var2 [0.5 0.6] Predicted_var2: [0.4 0.3]
        Total dist: 0.8817247532095723

Best params after iteration: 9
        True_var1 [0.3 0.4] Predicted_var1: [0.29900000000000004, 0.399]
        True_var2 [0.5 0.6] Predicted_var2: [0.5, 0.40000000000000013]
        Total dist: 0.8817247532095723
Found local/global minima
```

Process finished.

KL Divergence Graph:



This experiment shows that the proposed approach correctly identifies the noise parameters as optimal. The small update step in the first iteration is the result of the predicted sensor noise model being represented on a particle set and thus corresponding to the mixture model, slightly biasing it compared to the assumed model which is a single Gaussian.

## 5.3 Experiment 3 Details

We use non zero process noise. And assumed parameters that are far off from true noise parameters.

- Robot’s state space
    - cartesian space
    - i observable landmarks at

      position [xi, yi]
    - state - [x, y]
- Observation space
    - distance in cartesian space
    - dist(state, landmark)
- true process noise
    - mean - [0, 0]
    - var - [0.01, 0.002]
- true noise sensor 1

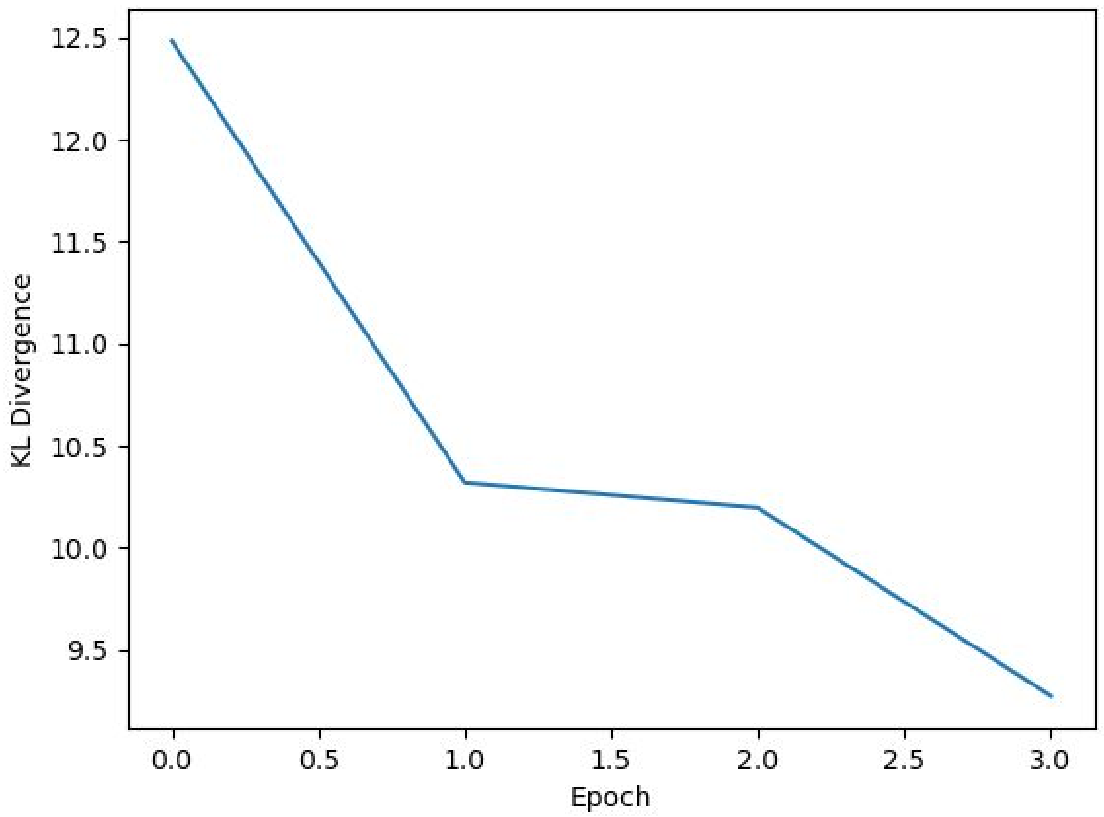
        - mean - [0, 0]
        - var - [0.3, 0.4]
    - true noise sensor 2
        - mean - [0, 0]
        - var - [0.5, 0.3]
- Particle Filter Parameters
    - Number of particles - 80
    - assumed process noise
        - mean - [0, 0], var - [0.01, 0.002]
    - assumed noise sensor 1
        - mean - [0, 0], var - [0.4, 0.5]
    - assumed noise sensor 2
        - mean - [0, 0], var - [0.6, 0.7]
- Hill Climbing Parameters
    - Step size - [0.01, 0.01]
    - Compute 4 neighbours
    - 5 random jumps if stuck in minima

```
        True_var1 [0.3 0.4] Predicted_var1: [0.4 0.5]
        True_var2 [0.5 0.3] Predicted_var2: [0.6 0.7]
        Total dist: 12.485092397101111

        True_var1 [0.3 0.4] Predicted_var1: [0.41000000000000003, 0.5]
        True_var2 [0.5 0.3] Predicted_var2: [0.6 0.7]
        Total dist: 10.320646004520357

        True_var1 [0.3 0.4] Predicted_var1: [0.39, 0.5]
        True_var2 [0.5 0.3] Predicted_var2: [0.6 0.7]
        Total dist: 10.196268827266818

        Best params after iteration: 1

        True_var1 [0.3 0.4] Predicted_var1: [0.39, 0.5]
        True_var2 [0.5 0.3] Predicted_var2: [0.6 0.7]
        Total dist: 10.196268827266818

        True_var1 [0.3 0.4] Predicted_var1: [0.39, 0.5]
        True_var2 [0.5 0.3] Predicted_var2: [0.61, 0.7]
        Total dist: 9.273738685136298

Best params after iteration: 2

        True_var1 [0.3 0.4] Predicted_var1: [0.39, 0.5]
        True_var2 [0.5 0.3] Predicted_var2: [0.61, 0.7]
        Total dist: 9.273738685136298

Best params after iteration: 3

        True_var1 [0.3 0.4] Predicted_var1: [0.39, 0.5]
        True_var2 [0.5 0.3] Predicted_var2: [0.61, 0.7]
        Total dist: 9.273738685136298

Found local/global minima. Making a random jump

        True_var1 [0.3 0.4] Predicted_var1: [0.38 0.49]
        True_var2 [0.5 0.3] Predicted_var2: [0.51 0.6 ]
        Total dist: 9.273738685136298

Best params after iteration: 4
```

```
True_var1 [0.3 0.4] Predicted_var1: [0.39, 0.5]
True_var2 [0.5 0.3] Predicted_var2: [0.61, 0.7]
Total dist: 9.273738685136298
```

Found local/global minima. Making a random jump

```
True_var1 [0.3 0.4] Predicted_var1: [0.38 0.49]
True_var2 [0.5 0.3] Predicted_var2: [0.51 0.6 ]
Total dist: 9.273738685136298
```



This is not converging on to true noise but, this is expected as we do not know the true process noise.

# Chapter 6

# CONCLUSION AND FUTURE WORK

In this research, we have successfully created an algorithm that approximates sensor noise models.

This is particularly useful in cases where sensor specific calibration targets are not available. In such cases, often this algorithm can be used to approximate a noise model, which (in most cases) will be better than making a random guess.

This algorithm also addresses the problem where sensors are deployed in large scale/ spread over a large area and calibrating each sensor individually is intractable. In such cases, this also makes maintenance of such sensor networks easier as calibration is automated.

In case of unforeseen environments, where the physics parameters might change, for example in space exploration, estimating a noise model may be cheaper than carrying sensor specific calibration targets.

In future, we want to implement this for all kinds of distributions and not just Gaussians. Also, the current implementation and mathematics only handles motion in 2D. This can be improved to implement in higher dimensions.

In the current implementation, the algorithm will converge close to but not exactly on the true prior distribution. The reason for this is that there exists a process noise and we do not know the distribution of this process noise. Hence, we model a noise that also incorporates process noise.

## BIBLIOGRAPHY

[1] http://hdl.handle.net/10068/56439 Non-linear filtering : Monte Carlo particle resolution

[2] Kullback, S.; Leibler, R. A. On Information and Sufficiency. Ann. Math. Statist. 22 (1951), no. 1, 79--86. doi:10.1214/aoms/1177729694. https://projecteuclid.org/euclid.aoms/1177729694

[3] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I," in *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99-110, June 2006, doi: 10.1109/MRA.2006.1638022.

[4] Whitehouse, K., Culler, D.: Calibration as parameter estimation in sensor networks. In: 2002 ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02), Atlanta, Georgia (2002) 67

[5] Z. Chen, *Bayesian Filtering: From Kalman filters to particle filters and beyond*.

[6] Cevher, V., McClellan, J.: Sensor array calibration via tracking with the extended kalman filter. In: 2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Volume 5. (2001) 2817–2820

[7] S. Thrun. Is robotics going statistics? The field of probabilistic robotics. Communications of the ACM, 2001

[8] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In AAAI-02.

[9] S. Majumder, H. Durrant-Whyte, S. Thrun, and M. de Battista. An approximate Bayesian method for simultaneous localisation and mapping. Submitted for publication, 2002.

[10] Bychkovskiy V., Megerian S., Estrin D., Potkonjak M. (2003) A Collaborative Approach to In-Place Sensor Calibration. In: Zhao F., Guibas L. (eds) Information Processing in Sensor Networks.

[11] H. Attias, “Inferring parameters and structure of latent variable models by variational Bayes,” in Proc. 15th Conf. UAI, UAI’99, 1999.

[12] J. Nikolic, P. Furgale, A. Melzer and R. Siegwart, "Maximum Likelihood Identification of Inertial Sensor Noise Model Parameters," in *IEEE Sensors Journal*, vol. 16, no. 1, pp. 163-176, Jan.1, 2016, doi: 10.1109/JSEN.2015.2476668.

[13] N. Muhammad and S. Lacroix, "Calibration of a rotating multi-beam lidar," *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Taipei, 2010, pp. 5648-5653, doi: 10.1109/IROS.2010.5651382.