**eInfochips – An Arrow Company**

***Internship Documentation***

***By***

*Akshay Mahesh Laddha, Engineering Intern (Sensor Fusion)*

**Manager:**

*Trevor Bingham*

**Team:**

*Gurpreet Singh*

*Naitik Nakrani*

*Akash Parikh*

*Amit Gupta*

**Internship Completion Date:**

*August 11th, 2022*

Table of Content

1. **Introduction ----------------------------------------------------------------------------------------------------------** *3*
2. **Preface -----------------------------------------------------------------------------------------------------***---------- 4*
3. **State-Space Models -----------------------------------------------------------------------------------------------** *5*
4. **Sensor Fusion and Algorithms -----------------------------------------------------------------------------** *6 - 8*

4.1**.** Kalman Filter

4.1.1**.** Extended Kalman Filter

4.1.2. Unscented Kalman Filter

4.2. Particle Filter

4.3. Bayesian Networks

1. **Why Extended Kalman Filter? --------------------------------------------------------------------------------** *9*
2. **Implementing Kalman Filter --------------------------------------------------------------------------------** *10*

6.1. Python Implementation

6.2. MATLAB Implementation

1. **ROS Packages for EKF ----------------------------------------------------------------------------------** *11 - 12*

7.1. robot\_localization

7.2. robot\_pose\_ekf

1. **Simulating the turtlebot3 ----------------------------------------------------------------------------** *13 - 15*

8.1. Launching

8.2. Moving the turtlebot

8.3. Checking for topics

8.4. Incorporating EKF

1. **Introducing Noise -------------------------------------------------------------------------------------** *16 - 17*

9.1. Odometry Noise

9.2. IMU Noise

1. **Tuning EKF parameters -----------------------------------------------------------------------------** *18 - 20*

10.1. Covariance matrix

10.2. Sensor Configuration matrix

10.3. Sensor diffrential parameter

10.4 Rejection Threshold

1. **Surface friction of Gazebo ------------------------------------------------------------------------------***21*
2. **Conclusion -------------------------------------------------------------------------------------------------** *22*
3. **Future Scope ---------------------------------------------------------------------------------------------** *23*
4. **References -----------------------------------------------------------------------------------------------** *24*

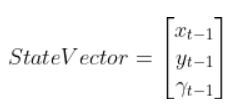
**Introduction**

Observing an autonomous vehicle that stops at the sign of red lights shows the rapid response of the vehicle and such a response is generated by the fusion the output from multiple sensors. AV and mobile robots are the latest players in this ecosystem of sensor fusion aiming at basically combining sensors that assists in tracking both stationary and moving objects in order to simulate human intelligence. In the context of sensor fusion, predict equation is the one that predicts the state of the car, and update equation is the one that continuously updates that prediction. The predict equation uses the previous prediction of the state (the range of possible state values calculated from the last round of predict-update equations) along with the motion model to predict the current state. This prediction is then updated (via the update equation) by combining the sensory input with the measurement model. Based on the above explaination, It can be deduced that performing sensor fusion provides us with much more accurate results as opposed to using an individual sensor for simulation. Therefore, the topic of focus will be performing Sensor fusion using ROS based packages on turtlebot3. For the given task, we will be using IMU and odometry sensor and aim to introduce certain type of elements that would present us with a more realistic simulation environment including noise addition to the sensors in any form including weather and environmental noise to surface friction and more.

**State-Space Models**

A state-space model is basically a mathematical equation that demonstrates how a robotic system might move from one timestep to the next. It also demonstrates adjustments to the robot’s control inputs, such as velocity in meters/second (typically represented by the variable v) that can affect the robot’s current position (e.g X,Y coordinate) and orientation (yaw heading) with respect to its environment.

Consider a robot moving around in an x-y coordinate plane, the position and orientation of the robot make up for state vector of the robot –



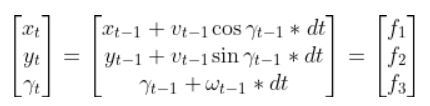
Here, **X** and **Y** as well as the **yaw** degree are represented in global frame

The yaw angle describes the rotation around z-axis in the counterclockwise direction and is represented in radians

The linear velocity in x and y-direction can be represented as -

In order to modify the equation to calculate the state forward for one timestep **dt**, the formula of

Which results in the following equation -



The matrix equation above is in non-linear form and to use it for applications like Kalman Filter or Particle Filter, it is converted to linear state-space model form –



**Xt**:Current state vector at timestep **t**

**Xt-1**: State of the robot at previous time-step**t-1**

**Ut-1** : represents the control input vector at previous timestep [forward velocity, angular velocity]

**A matrix**: describes how the system's state (x, y, and yaw angle) changes from time t-1 to time t when no control instruction is executed, or when we don't instruct the robot to move at a certain pace (or velocity)

B matrix:

**Sensor Fusion and Algorithms**

Combining sensors that track both stationary and moving objects in order to simulate human intelligence. Sensor fusion suggest fusing together signals of multiple sensors to determine the position, trajectory and the speed of an object which help in reducing the uncertainty in machine perception Sensor fusion relies on data from several of the same types of sensors known as competitive configuration Combining different types of sensors (fusing proximity sensor with speedometer data) usually yields a more comprehensive understanding of object and this kind of setup is known as complementary configuration Motion model deals with the dynamics of the object by predicting the current state of the car by drawing from a range of values that depend on its state during the last time step. Measurement model works on the dynamics of the car’s sensors.

The susceptibility to interference is one feature shared by all sensors. Sunlight can hide or blind a camera, and a radar can be jammed. These situations may provide sensory data that is distorted, spotty, or just incorrect. As a result, the majority of sensor data collected in the actual world consists of two components: a signal (the part we're interested in) and noise (the part we'd want to avoid). Sensor fusion analyzes multiple data sources at once in an effort to extract the noise from the facts and provide with a more filtered output

On the basis of abstraction level, Sensor fusion can be divided into –

**Low-Level**

Low-level sensor fusion takes raw data as input. We’re referring here to the sensor’s point data measurements. This approach makes sure we don’t add any noise to the data upon post-processing it. The downside to this method is that it requires the processing of an immense amount of data.

**Mid-Level**

At the intermediate level, data fusion operates on object hypotheses. It uses data that has been interpreted either within the sensor itself or by a different processing unit. For example, when a camera thinks an object is straight ahead, the Lidar might sense it slightly to the right. With mid-level sensor fusion, these two interpretations are weighted to arrive at a single projection.

**High-Level**

Tracks are hypotheses about an object’s movement in space. In high-level sensor fusion, we again see the merger of two hypotheses in a weighted manner. This time, however, the hypotheses aren’t just about an object’s position, but also about its trajectory, thus incorporating its past and future states

Some useful sensor fusion algorithms widely used for Noise filtering include –

**Kalman Filter**

A Kalman filter is an algorithm that takes data inputs from multiple sources and estimates unknown values, despite a potentially high level of signal noise. Often used in navigation and control technology, Kalman filters have the advantage of being able to predict unknown values more accurately than individual predictions using single methods of measurement.

The Kalman Filter is further divided into two categories on the basis of their application –

***Extended Kalman Filter***:

ROS Package: robot\_localization, robot\_pose\_ekf

The Extended Kalman filter is a state estimation and localization algorithm that makes an estimate the position of the robot with an estimated noise, then uses the information from the sensor as a measurement and update it further to get a more realistic value and continue the same steps in recursion in order to get a more accurate position of the robot. EKF help to predict the nonlinear quantity of the robot for instance, the pose of the robot which includes the orientation. With reference to the ROS packages, robot\_localization package provides 3D tacking based on Extended Kalman filter with mulitple sensors (multiple odom and multiple IMU..) and more functionality while robot\_pose\_ekf can take only two odometry data and one imu data