ABSTRACT

MAGNETOMETER-LESS STATE-ESTIMATION OF A UNICYCLE MODEL MOBILE ROBOT USING A CASCADED KALMAN FILTER FRAMEWORK

Tommy Le, M.S.
Department of Mechanical Engineering
Northern Illinois University, 2023
Dr. Ji-Chul Ryu, Director

NORTHERN ILLINOIS UNIVERSITY
DE KALB, ILLINOIS

DEC 2023

MAGNETOMETER-LESS STATE-ESTIMATION OF A UNICYCLE MODEL MOBILE ROBOT USING A CASCADED KALMAN FILTER FRAMEWORK

BY

TOMMY LE
© 2023 Tommy Le

A THESIS SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE
MASTERS OF SCIENCE

DEPARTMENT OF MECHANICAL ENGINEERING

Thesis Director:
Dr. Ji-Chul Ryu

ACKNOWLEDGEMENTS

Here's where you acknowledge folks who helped.

DEDICATION

To all of the fluffy kitties.

TABLE OF CONTENTS

Page

LIST OF FIGURES. vi

Chapter

1 INTRODUCTION 1

1.1 Motivation 1

1.2 Literature Review 1

1.3 Objective 1

2 METHODS 3

2.1 IMU Preprocessing 3

2.1.1 Accelerometer Signal Processing. 3

2.1.1.1 Displacement Measurement from Acceleration Data 4

2.1.1.2 No Displacement Zero Translational Accumulation (NDZTA) 4

2.1.2 Gyroscope Signal Processing 6

2.1.2.1 Bias Compensation 6

2.1.2.2 No Motion No Integration (NMNI) 6

2.2 First KF: Course Correction Extended Kalman Filter. 7

2.2.1 Process Model 8

2.2.2 Measurement Model 9

2.3 Second KF: Bias Estimation Kalman Filter 11

2.3.1 Process Model 11

2.3.2 Measurement Model 12

Chapter Page

2.4 Estimation with Additional Sensor 13

3 FUTURE WORK 14

3.1 Simulation 14

3.2 Physical Implementation 14

LIST OF FIGURES

Figure Page

2.1 NDZTA Flow 5

2.2 NMNI Flow 7

CHAPTER 1
INTRODUCTION

This section will cover background information on mobile robot kinematic models, IMU state estimation, Kalman Filtering methods, and cascading KF framework.

1.1 Motivation

Talk about the development of miniature robots. Talk about our spherical robot design. The small size of the robot makes it difficult to use a magnetometer due to the close proximity of the motors and sensors. Using an external sensor is clunky or not realistic.

1.2 Literature Review

This section should go over the papers I have gone over. This includes NMNI, NDZTA, Song 2018 on the cascaded KF framework, as well as other papers using external sensors to get more robust heading.

1.3 Objective

To tackle these challenges, this thesis aims to find a novel way to estimate a mobile robots states using only a 6DOF IMU with a unicycle model wheeled mobile robot. The thesis will have an algorithm with only a 6DOF IMU as well as having the option to add an additional sensor to provide more robust localization.

CHAPTER 2
METHODS

This part will be an overview of the overall framework of the algorithm. It will first discuss preprocessing methods. These methods involve NMNI, NDZTA, and error compensation in the gyroscope and accelerometer. Afterwards it will discuss the first and second kalman filter models, then the optional block containing the additional sensor to improve the performance of the filter.

2.1 IMU Preprocessing

Here I will first talk about what an IMU is, 6DOF vs. 9DOF, how each sensor works, and the inherent errors in each sensor.

It will then cover the planned signal processing methods for each sensor on the IMU and go in depth about each algorithms implementation.

2.1.1 Accelerometer Signal Processing

This subsection talks about the preprocessing methods that will take place on the accelerometer data coming from the IMU as well as the numerical integration methods used for displacement calculation.

2.1.1.1 Displacement Measurement from Acceleration Data

Euler integration, trapezoidal, RK4. (Better integration = better data) Don't think I really need to explain this right now.

2.1.1.2 No Displacement Zero Translational Accumulation (NDZTA)

Sci-hub Link: https://sci-hub.live/https://ieeexplore.ieee.org/ document/9268038

Removing the noise in accelerometer data during stationary points in time is done by finding a way to have the system understand when it is stationary. The No Displacement Zero Translation Accumulation (NDZTA) algorithm accounts for this by calculating a threshold value that indicates whether the system is in motion or not. Figure 2.1 illustrates the logic of the algorithm.

First a low pass filter is applied to the accelerometer data to eliminate the inherent high frequency noise. After the filter is applied, a maximum value is collected from a sampling window during an initial static position. This maximum value becomes the threshold that helps the system realize it is not in motion. Now the displacement is calculated based on this found threshold which signifies the boundary between static and dynamic motions. The algorithm works by comparing the real-time acceleration data a with the threshold a_{th} to decide if displacement calculation is needed. This comparison is demonstrated in equations 2.1 and 2.2.

$$a > a_{th}, \text{ Sensor is in motion. Carryout displacement calculation} \tag{2.1}$$

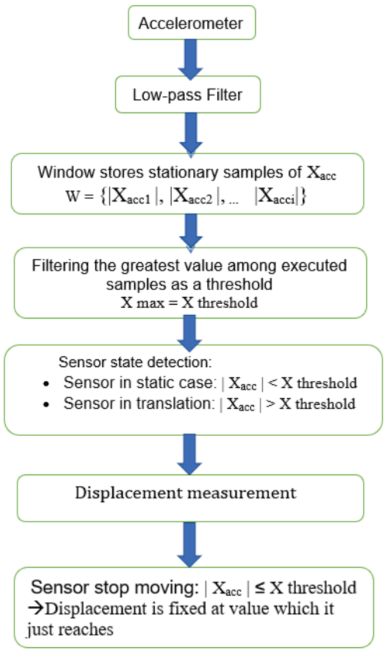
Accelerometer
Low-pass Filter
Window stores stationary samples of X_{acc}
$W = \{|X_{acc1}|, |X_{acc2}|, \ldots |X_{acci}|\}$
Filtering the greatest value among executed samples as a threshold
X max = X threshold
Sensor state detection:
Sensor in static case: $|X_{acc}| <$ X threshold
Sensor in translation: $|X_{acc}| >$ X threshold
Displacement measurement
Sensor stop moving: $|X_{acc}| \leq$ X threshold
→Displacement is fixed at value which it just reaches

Figure 2.1: NDZTA flow

$$a \leq a_{th}, \text{ Sensor is static. Set acc} = \text{vel} = \text{pos} = 0 \tag{2.2}$$

Note that the displacement calculation has many errors due to noise in the sensor signals and numerical integration causing the error to grow. In order to eliminate the error, the NDZTA algorithm proposes a method composed of these 3 main steps.

Step 1: Threshold Calculation

Step 2: Motion Detection

Step 3: Displacement Measurement

During each detection of the static case, the threshold value is updated depending on equation 2.3.

Equation 2.3 here!

some conditional statement that I cannot remember.

2.1.2 Gyroscope Signal Processing

This subsection talks about the preprocessing methods that will take place on the gyroscope data coming from the IMU.

2.1.2.1 Bias Compensation

Don't think I need to really explain this right now.

2.1.2.2 No Motion No Integration (NMNI)

Sci-hub link: https://sci-hub.live/https://ieeexplore. ieee.org/document/9247295 https://sci-hub.live/https://www.sciencedirect. com/science/article/abs/pii/S0924424721001540

To find roll, pitch, and yaw angles, the angular rate data from the gyroscope is numerically integrated. Due to multiple different error sources, noise and drift problems will cause the gyroscope data to vary.

To compensate for these errors, the No Motion No Integration (NMNI) algorithm is used. This algorithm only allows numerical integration when dynamic motion is detected. Similar to the NDZTA algorithm discussed in previous sections, this algorithm finds the maximum value within sampled data during an initial window of static motion. This max value is set

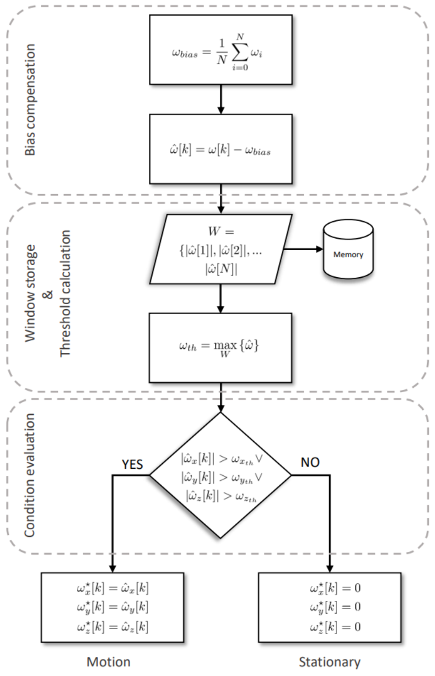
Bias compensation
$\omega_{bias} = \frac{1}{N}\sum_{i=0}^{N}\omega_i$
$\hat{\omega}[k] = \omega[k] - \omega_{bias}$
Window storage & Threshold calculation
$W = \{|\hat{\omega}[1]|, |\hat{\omega}[2]|, \ldots |\hat{\omega}[N]|$
Memory
$\omega_{th} = \max_W \{\hat{\omega}\}$
Condition evaluation
$|\hat{\omega}_x[k]| > \omega_{x_{th}} \vee |\hat{\omega}_y[k]| > \omega_{y_{th}} \vee |\hat{\omega}_z[k]| > \omega_{z_{th}}$
YES
NO
$\omega_x^\star[k] = \hat{\omega}_x[k]$ $\omega_y^\star[k] = \hat{\omega}_y[k]$ $\omega_z^\star[k] = \hat{\omega}_z[k]$
$\omega_x^\star[k] = 0$ $\omega_y^\star[k] = 0$ $\omega_z^\star[k] = 0$
Motion
Stationary

Figure 2.2: NMNI flow

as a threshold value, and the real-time data collected by the gyroscope is compared with the threshold value to determine if the system is in static or dynamic motion. The overall logic of the algorithm can be seen in figure 2.2 below.

Maybe enter equations here too. But seems redundant.

2.2 First KF: Course Correction Extended Kalman Filter

This section covers the first Kalman Filter in the cascaded Kalman Filter framework. This Kalman Filter uses the unicycle kinematic model as the process model, and gets bias compensated feedback position data from the accelerometer as the measurement model.

2.2.1 Process Model

The kinematic model for the robot follows the common unicycle model for a differentially driven wheeled-mobile robots (WMRs). This thesis is assuming 2D planar motion so the states are x, y, and theta. Equation 2.4 below describes the state transition matrix for this system.

$$x = z = [x, y, \theta]^T \tag{2.3}$$

$$\dot{x} = f(x)u = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} \tag{2.4}$$

where v and w are the linear and angular velocity of the mobile robot. Since we are working with a differential drive robot, the linear and angular velocities need to be related to the individual wheel speeds of the robot. The relationship between v and w and v_L and v_R, the left and right wheel speeds of the robot can be seen in equations 2.5 and 2.6, where r is the radius of the robot's wheel and l is the distance between the wheels.

$$v = \frac{r}{2}(v_R + v_L) \tag{2.5}$$

$$w = \frac{r}{l}(v_R - v_L) \tag{2.6}$$

Since this model is non-linear, the Extended Kalman Filter (EKF) is used to estimate the states. To implement the EKF, the Jacobian of the state transition matrix must be found to create a linear approximation of our non-linear equations. The EKF works by using a first order taylor approximation to linearize the model around the current state and uses the Jacobian of the model equations to calculate the covariance. To find the covariance, equation 2.7 is used where F is the Jacobian of the unicycle robot model, P is the current covariance, and Q is the process noise.

$$P^{-} = FPF^{T} + Q \tag{2.7}$$

$$F = \begin{bmatrix} 1 & 0 & -vsin(\theta) \\ 0 & 1 & vcos(\theta) \\ 0 & 0 & 1 \end{bmatrix} \tag{2.8}$$

2.2.2 Measurement Model

$$z = h(x)x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{2.9}$$

The measurement model as well as its Jacobian are identity because the states can be directly accessed with intermediate calculations. Here the Jacobian is used to find the innovation covariance and can be found through equation 2.11, where P is the covariance found from the prediction state of the Kalman filter and R is the measurement noise.

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.10}$$

$$S = HPH^T + R \tag{2.11}$$

Going into this system will be the corrected and compensated position and orientation data that is found from our second Kalman Filter. In other words, the measurement for the first Kalman Filter is the estimation from the second Kalman Filter. These are then used to estimate the states taking into account the robots kinematic constraints along with the sensor measurements.

To solve for heading given the gyroscope data, w_g a simple Euler integration is carried out and can be seen in equation 2.11.

$$\theta = \theta_0 + w_g \Delta t \tag{2.12}$$

To get global x and y position data from accelerometer readings, the local acceleration readings, a^b, are transformed using C_b^g which is a rotation matrix about the z-axis, and become a^g, then double integration is carried out to arrive at global position. These conversions can be seen from equations 2.12-2.14. The overall conversion from the compensated sensor readings to the appropriate states are handled externally from the Kalman Filters and can be described through equations 2.11–2.14.

$$a^g = C_b^g a^b \tag{2.13}$$

$$v = v_0 + a^g \Delta t \tag{2.14}$$

$$p = p_0 + v \Delta t \tag{2.15}$$

2.3 Second KF: Bias Estimation Kalman Filter

This section covers the second Kalman Filter in the cascaded Kalman Filter framework.This Kalman filter uses the gyroscope model as the process model, and uses the accelerometer data as the measurement model to estimate our states as well as correction terms to compensate for sensor errors.

2.3.1 Process Model

The states are orientation error ($\delta\phi$), velocity error (δv), gyroscope bias (b_g), and accelerometer bias (b_a). Each state mentioned has 3 components relating to the north-east-down (NED) coordinate frame which gives a total of 12 states for the second Kalman filter.

$$\boldsymbol{x}_2 = [\delta\boldsymbol{\phi}^T, \delta\boldsymbol{v}^T, \boldsymbol{b}_g^T, \boldsymbol{b}_a^T]^T \tag{2.16}$$

The state transition matrix is one adopted from the popular pedestrian dead reckoning algorithm (PDR) for pedestrian rather than mobile robot navigation. The process model can be described through equations 2.17–2.21. The heading error is

$$f = \begin{bmatrix} \mathbf{0}_{3x3} & \mathbf{0}_{3x3} & -\boldsymbol{C}_b^n & \mathbf{0}_{3x3} \\ \boldsymbol{S} & \mathbf{0}_{3x3} & \mathbf{0}_{3x3} & \boldsymbol{C}_b^n \\ \mathbf{0}_{3x3} & \mathbf{0}_{3x3} & -\beta_g \boldsymbol{I}_{3x3} & \mathbf{0}_{3x3} \\ \mathbf{0}_{3x3} & \mathbf{0}_{3x3} & \mathbf{0}_{3x3} & -\beta_a \boldsymbol{I}_{3x3} \end{bmatrix} \tag{2.17}$$

C (don't know if definition is correct) is a rotation matrix. S is a skew symmetric matrix of acceleration representing vector cross product. β are the Markov process time constants that model our biases.

$$\boldsymbol{C}_b^n = \begin{bmatrix} cos(\phi_D) & -sin(\phi_D) & 0 \\ sin(\phi_D) & cos(\phi_D) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.18}$$

$$S = \begin{bmatrix} 0 & -a_D & a_E \\ a_D & 0 & -a_N \\ -a_E & a_N & 0 \end{bmatrix} \tag{2.19}$$

2.3.2 Measurement Model

Measurements are course angle error and velocity measurements. Course angle error is found by taking the difference between the KF estimated heading angle from the gyro and unicycle model and the estimated heading found from using the previous and current position data that provides the heading for the robot to linearly move from the previous point to the current point.

$$\delta\theta = \theta_{est} - \theta_{acc} \tag{2.20}$$

where θ_{est} is the heading estimated by the kalman filter and θ_{acc} is the heading found from the position data.

COMMENT

But doesn't heading from kinematics come from acc and gyro already? So instead, take the improved heading from fusing kinematics with acc and gyro, and pass to gyro KF instead of heading error.

So first KF takes x, y, θ as measurement and outputs θ into KF2. KF2 then takes in v and θ to estimate errors to correct/compensate sensor measurements and vel and orientation errors. These improvements of measurement errors and state errors should improve x, y, θ and improve estimates in the first KF. Those will be the states that we care more about.

But if we are just going to use the states estimated by the first KF, why don't we just find orientations and pass it to first KF rather than find errors, then externally correct them then pass them to the first KF.

COMMENT

$$z_2 = [\delta\theta, \boldsymbol{v}]^T \tag{2.21}$$

$$h = \begin{bmatrix} \begin{bmatrix} tan(\phi_N)cos(\phi_D) & tan(\phi_N)sin(\phi_D) & -1 \end{bmatrix} & \mathbf{0}_{1x3} & \mathbf{0}_{1x3} & \mathbf{0}_{1x3} \\ \mathbf{0}_{3x3} & \boldsymbol{I}_{3x3} & \mathbf{0}_{3x3} & \mathbf{0}_{3x3} \end{bmatrix} \tag{2.22}$$

2.4 Estimation with Additional Sensor

This sensor should be one that can give position data. GPS, camera, encoder. From this extra sensor, you can set up a cost function that models the error in the measurements and minimize it using gradient descent or something to hopefully get the best estimate in position/heading.

CHAPTER 3
FUTURE WORK

The work left involves simulating the entire framework of the algorithm. Currently the preprocessing methods are implemented on to the KF blocks and are simulated separately. In the future they need to be combined and the performance needs to be analyzed.

Afterwards, how the optional sensor is to be implemented into the algorithm needs to be figured out. (Thinking of optimizing a cost function between sensor and position from accelerometer to provide best postion estimate then feeding that into the KF.)

Finally, implementation on a physical system is left. The planned robot for implementation is the Sphero-mini.

3.1 Simulation

Specifications of simulation. Errors modeled, results of simulation.

3.2 Physical Implementation

Sphero-mini's specifications. Sensors. Workspace specifications. Results.