2019

Localizing a quadrotor with collisions: novel sensor design and applications to particle filtering

BOSTON UNIVERSITY

COLLEGE OF ENGINEERING

Thesis

LOCALIZING A QUADROTOR WITH COLLISIONS: NOVEL SENSOR DESIGN AND APPLICATIONS TO PARTICLE FILTERING

by

CHENG LIU

B.S., Shanghai Jiao Tong University, 2010
M.S., Nanjing University of Aeronautics and Astronautics, 2015

Submitted in partial fulfillment of the

requirements for the degree of

Master of Science

2019

Approved by

First Reader

Roberto Tron, PhD
Assistant Professor of Mechanical Engineering
Assistant Professor of Systems Engineering

Second Reader

John B. Baillieul, PhD
Distinguished Professor of Mechanical Engineering
Distinguished Professor of Systems Engineering
Distinguished Professor of Electrical and Computer Engineering

Third Reader

Sheryl Grace, PhD
Associate Professor of Mechanical Engineering

Und verloren sei uns der Tag, wo nicht Ein Mal getanzt wurde!
Und falsch heiße uns jede Wahrheit, bei der es nicht Ein Gelächter gab!

Friedrich Nietzsche

Acknowledgments

In the process of pursuing this master of science program, I have been supported by many members of the department, friends, family to whom I am truly grateful.

First and foremost, I would like to thank my academic supervisor, Dr. Roberto Tron, and committee members, Drs. John Baillieul and Sheryl Grace, for their technical expertise, patience and support during the process. My special thanks belongs to Dr. Tron, who over the past one year has always been very patient when advising me and discussing my issues.

I would additionally like to thank my faculty advisor, Prof. Sheryl Grace, who advised me to join Prof. Tron's research and project.

I also owe a considerable amount of gratitude to the Mechanical Engineering Department, its staff, faculty, and student body, who all have been extremely supportive. Special thanks belong to the program administrator, Anna, and the director of the master program, Prof. Park. They always handle my typical international students' issues timely. Thank you for your assistance.

Finally, I would like to thank my family and friends for their care and support. I could not have done it without you.

LOCALIZING A QUADROTOR WITH COLLISIONS: NOVEL SENSOR DESIGN AND APPLICATIONS TO PARTICLE FILTERING

CHENG LIU

ABSTRACT

Collisions are typically seen as adverse events for quadrotors, with numerous researchers designing cages for minimizing the effect of impacts on the vehicles. In this thesis, we reverse this paradigm, treating collisions as an additional opportunity for localization. In order to gather collision information, a touch-only sensor with a protective frame is designed to sense the 2-D relative displacement due to inertial force between vehicle and the frame while collision happens. We provide a mathematical model that represents the displacement in terms of the poses of the protective frame and quadrotor is constructed and solved numerically, which helps analyze the distance from obstacles to the vehicle and collision direction. At last, a particle filter based localization observing the collision status is simulated, which verifies the theoretical feasibility utilizing collision information to perform localization in a known environment.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Unmanned Aerial Vehicles (UAVs) have become very affordable and small in recent years, and an increasing number of them are used in a wide range of monitoring and surveillance applications. A major problem when operating with UAVs in a constrained environment, is the risk of collisions with obstacles or with other vehicles (when operating in a swarm); this has been the motivation of many important topics in UAV research, such obstacle avoidance, localization and motion path planning. While those tasks aim at satisfying some control objective subject to non-intersection or non-collision position constraints, typically obstacle avoidance is considered to be distinct from path planning in that one is usually implemented as a reactive control law while the other involves the pre-computation of an obstacle-free path which a controller will then implement. In either case, there is a need for sensor technologies that can detect the surrounding airspace in order to prevent collisions.

However, under some circumstances, collisions are inevitable in missions such as inspection and exploration of some inaccessible or extremely confined spaces. In addition, small UAVs are relatively lightweight and have small inertia, so that they are easily influenced by factors in the surrounding environment, such as aerodynamic forces due to atmospheric turbulence, or external forces due to slung loads or collisions with obstacles. For the latter, a detector specifically designed for observing the collision information could serve the UAV as an important role in improving stability.

At the same time, most of UAVs like quadcopters rely on active propulsion systems using rotors or propellers that cannot tolerate collisions and could damage the surrounding environment. It is therefore necessary to build a protective frame wrapping the UAV to secure the main aerodynamic components, or use advanced planing and control strategies for collision avoidance, so that the UAV could work under more intricate environment without risking damage to the vehicle or to the objects that it may strike. Nonetheless, traditional protective cages only contribute to protecting the integrity of the vehicles, with an increase in the effective body volume of vehicles, and with sacrifices in terms of effective payload.

In this thesis, we study a new cage design with the double purpose of protecting the quadrotor, while also providing collision information for self-localization purposes. As to most former investigations and conventional demonstrators with protection, the cage is customarily used for one purpose that is to prevent damage to the structure of key components from collision, conflict. The sensors installed on the vehicle are mostly applied to determine the relative position of obstacles in the airspace, especially while running autonomously without communication with the operator.

1.2 Research Overview

The main objective (see Figure 1.1) of this research is to build a collision sensor with a protective frame or cage that also protects the quadrotor, model the relative displacement between the sensor and the protective frame caused by collisions, and investigate a particle-filter-based localization that fuses these measurements into a position estimate in a known map.

From a vehicle integrity perspective, the protective frame encloses the quadrotor vehicle, making it capable of withstanding impacts and accessing complex, cluttered or indoor environments. Regarding the goal of collision detection, we measure the

relative displacement of the frame and the vehicle by using an assembly of inelastic strings and other mechanisms to measure the relative motion between the frame and a board that is solidly mounted on the top of the vehicle. The board houses electronic circuits to sense the displacements of the strings, and hence the changes in 2-D translations and 1-D rotations between vehicle and protective frame during collisions (while a fully 3-D design is possible, it is not pursued in this thesis). An on-board Arduino collects the data on-line, performing real-time processing in order to analyze the magnitude and direction of perturbation that the vehicle is experiencing.

The main novel aspect of this research is that the sensor and the cage not only serve as a protection for the vehicle, but also as a feasible approach to perform localization by sensing collision status. The collision information help vehicle locate itself within a 2-D known environment. Similarly, external disturbance forces experienced by the vehicle can be detected, recorded, and eventually used to help the control system to deal with the uncertainties in the working space. This would reduce the need for sophisticated sensor suits to reconstruct the surrounding airspace. Ideally, by using the sensing results, a control algorithm could also compensate in real time the effects of collisions while keeping the vehicle stable. However, in this thesis we focus on using the same information for localization, not control.

The most challenging part of this project is about how to design a device that could observe the external perturbation in three degrees of freedom, analyze the detected perturbation data, decouple and reveal what kind of relative motions between the vehicle and the protective frame are happening.

In this thesis, we consider only 2-D displacements in our system. The collision-caused pitch and rolling motions are ignored using a small angle assumption. In future work, we will consider more sophisticated models to analyze 3-D relative motions with six degrees of freedom (three for rotations, three for translation).

1.3 Statement of work

In this thesis, a collision sensor is designed to measure the distance between a sensor-mounted quadrotor while the protective frame collides with obstacles (Chapter 2). For the electrical part, a circuit schematic and its corresponding printed circuit board (PCB) are designed. In practical application, a soldered perf board onto which the equivalent circuit is mapped will be used to connect Arduino and potentiometers. For the mechanical part, we design a few of mechanical components to drive the encoders that sense the relative displacement due to collision with environment. In addition, a mathematical model (Chapter 3) that illustrates the geometric relationship in 2-D plane between the protective frame and the sensor is developed, which could numerically analyze the relative displacement. Finally, a particle filter based localization (Chapter 4) for a touch-only sensor mounted vehicle in a known 2-D environment is simulated. The result shows that the idea of applying a sensor capable of measuring the distance from obstacles to perform localization for mobile vehicles is theoretically feasible.

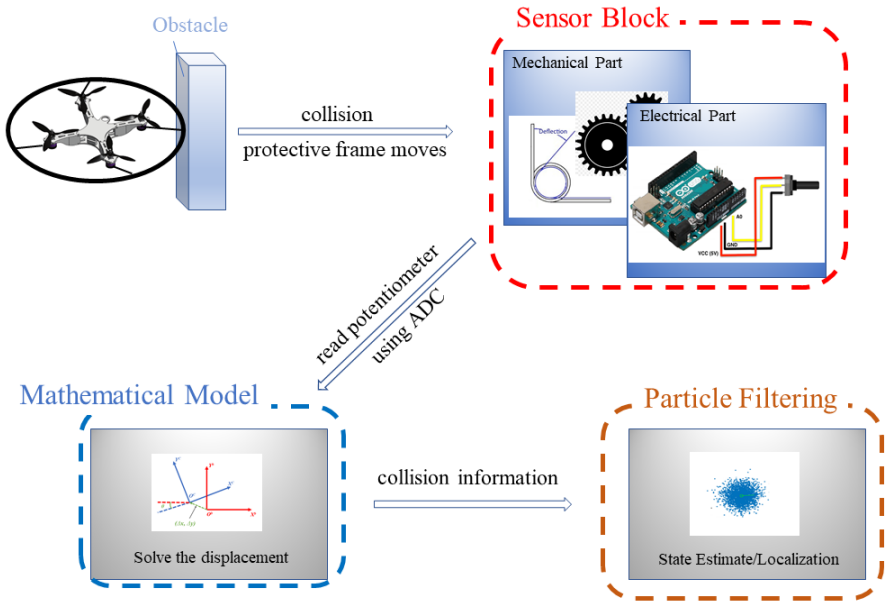


Figure 1·1: Interdependence of main work. The sensor block is illustrated in Chapter 2; Chapter 3 describes the mathematical model; The particle filtering part is implemented in Chapter 4.

Chapter 2

Electrical and mechanical design for the sensor

2.1 Introduction

Collisions usually have a negative influence on robots, especially for small UAV systems, since the aerodynamic components (such as propeller blades) are easy to damage. Once those parts are damaged, the aerodynamic behavior of the vehicle might become unpredictable, resulting in a loss of controllability of forces and torques, rendering the vehicle unoperable. In order to secure those components, previous work in (Kalantari and Spenko, 2013) and (Caroti, Piemonte, etc., 2018) proposes designs for protective cages to allow quadrotors to tolerate collisions (Figure 2.1).

(a) (b)

Figure 2·1: (a) Elios UAV - a collision tolerant structure (b) the HyTAQ - hybrid terrestrial and aerial quadrotor

In these cases, protective cages are designed to serve only one purpose – protection. However, at the same time, mounting protective cage sacrifices a part of payload, increases the take-off weight and reduces the flight performance of UAVs. A way to is to remedy this situation is to design the cage to absolve functions other than mere protection.

Henceforth, our aim is to investigate whether or not the cage could function as a collision sensor to gather collision information. To start, this chapter presents the design layout of an electrical circuit connecting to Arduino, and a mechanical design for the mechanical-based sensor drive system. The sensor is mounted on the bottom of a quadrotor vehicle, and connects to the protective carbon fiber frame. The sensor is able to sense the change of the displacement between the frame and the vehicle itself. The assembled parts (sensor, vehicle and protective frame) are shown in Figure 2.2.

Figure 2·2: Assembled parts

The interdependence between hardware modules is illustrated in Figure 2.3. Briefly speaking, when the collision happens, the protective frame is going to move due to the inertial force. Then, there will be a displacement between the frame and vehicle. Through the strings, the frame will drive the legs of torsion springs, making them deflect, driving, in turn, the gear train. At this point, the potentiometer shaft starts to

rotate, and numeric values that illustrate the magnitude of the spring deflection can be obtained through the electrical part of the hardware (Section 2.3). Based on these values, the displacement can then be solved to compute the collision information (see Chapter 3).

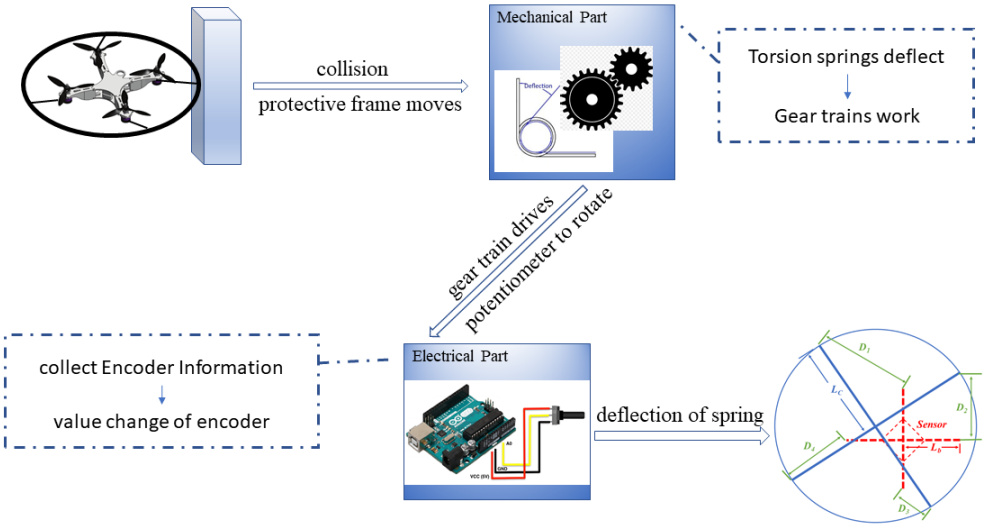


Figure 2·3: Interdependence of hardware (mechanical and electrical)

2.2 Mechanical design

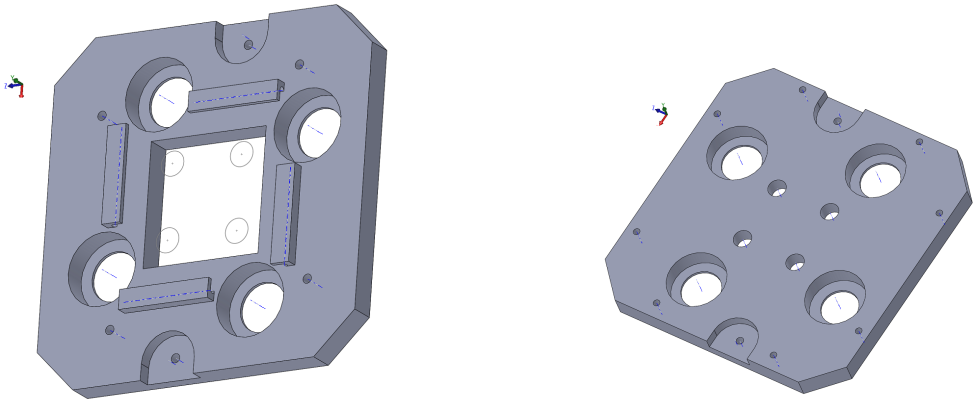
In order to prototype the collision sensor, a few components are designed using 3-D modeling software (Solidworks), and manufactured through 3D printing. The necessary components are shown in Figure 2.6, including two bases, four pairs of gear trains, connection joints, bearings, landing struts and torsion springs.

2.2.1 Sensor bases

The 3-D model for the two sensor bases are shown in Figure 2.5. Subfigure (a) shows the bottom base which is fixed on the base support part of the quadrotor. The

Figure 2·4: Sensor components

square hole helps set the gear train during assembly, and, at the same time, reduce the weight of the sensor to some extent. Four circular gaps are designed to house the ball bearings (see Figure 2.7), and attach the gear pair with the torsion springs.



(a) (b)

Figure 2·5: Two sensor bases: (a) bottom base (b) top base

As we can see in (b), the top base has similar layout with four larger circular gaps, which serves the same objective, housing the ball bearings (see Figure 2.6). The four relatively small holes are drilled to allow the potentiometer shafts to pass through

from the other side of the base (see Figure 2.7).

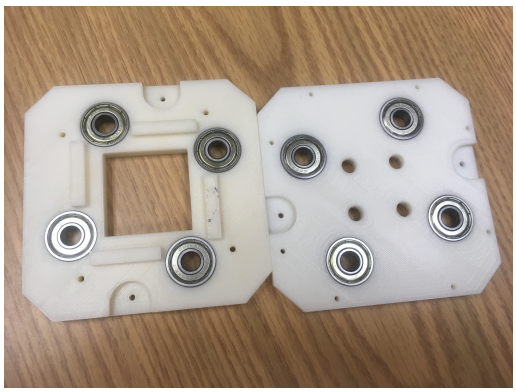
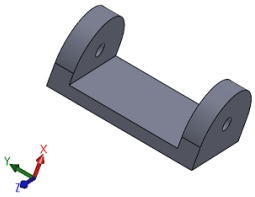
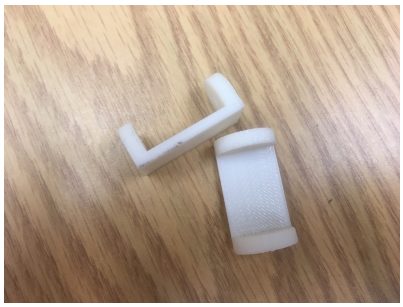


Figure 2·6: Two bases with bearing plugged-in

Figure 2·7: Top bases with perf board plugged in



(a)



(b)

Figure 2·8: Base connection joint: (a) 3D model (b) printed part

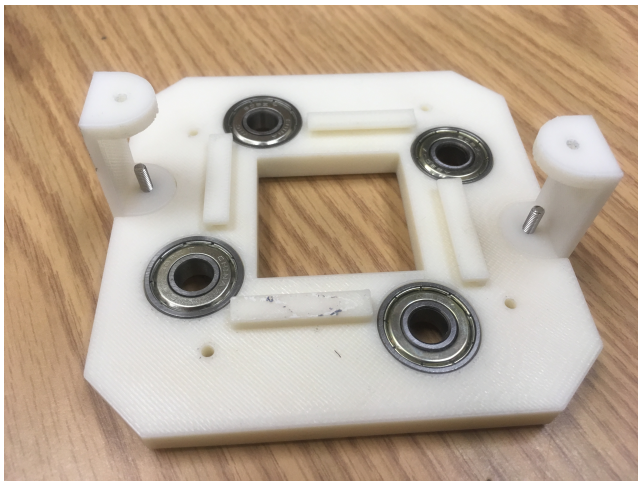


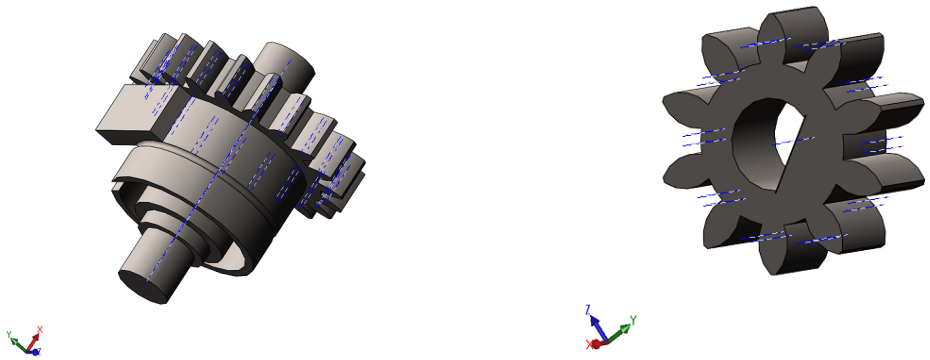
Figure 2·9: Connection parts assembled on the bottom base

Two connection parts are used to rigidly combine the two sensor bases. The 3-D model and the printing of such base connection parts are presented in Figure 2.8. Figure 2.9 illustrates the assembled parts on the bottom base.

2.2.2 Gear train

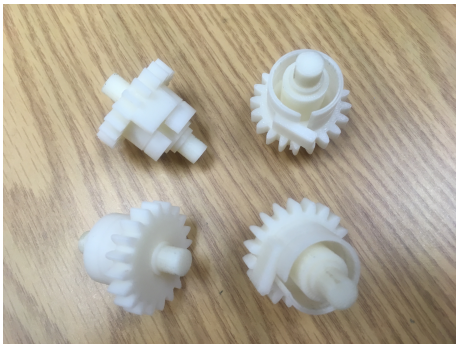
The function of the gear train is to drive the potentiometers in response to deflections of the springs. The gear train is needed to mechanically amplify the magnitude of the displacements due to typical inertial forces. If we were to use only one gear to connect to the potentiometer, a small deflection of the torsion spring would not make the value of potentiometer change significantly enough.

In order to gather the magnitude of spring deflection, we designed a gear train composed of a driving gear and a driven gear (see Figure 2.10 and Figure 2.11).



(a) (b)

Figure 2·10: Gear train: (a) driving gear (b) driven gear



(a)

(b)

Figure 2·11: Printed gear train: (a) driving gear (b) driven gear

As apparent from Figure 2.11, the ring-shaped gap on the driving gear allows to plug torsion spring in it (see Figure 2.12). For the driven gears, the potentiometer shaft can be press-fit in the central hole, as shown in Figure 2.13. As a result, the potentiometer shaft is rotated by driving gear that will rotate because the torsion spring deflects while the protective frame hits obstacles.

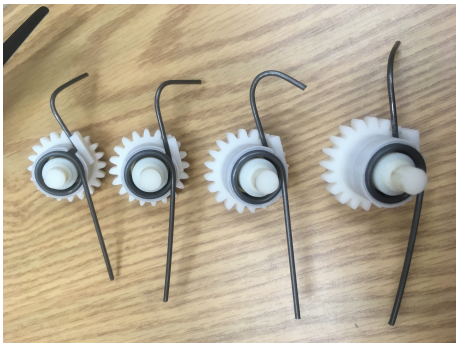


Figure 2·12: Driving gear with torsion spring plugged-in

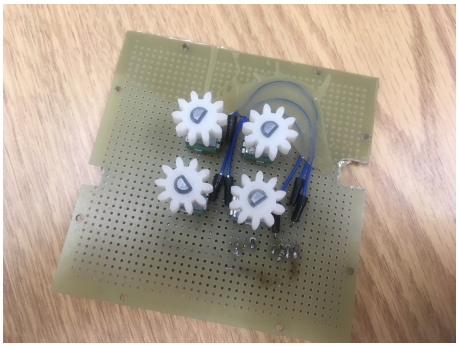


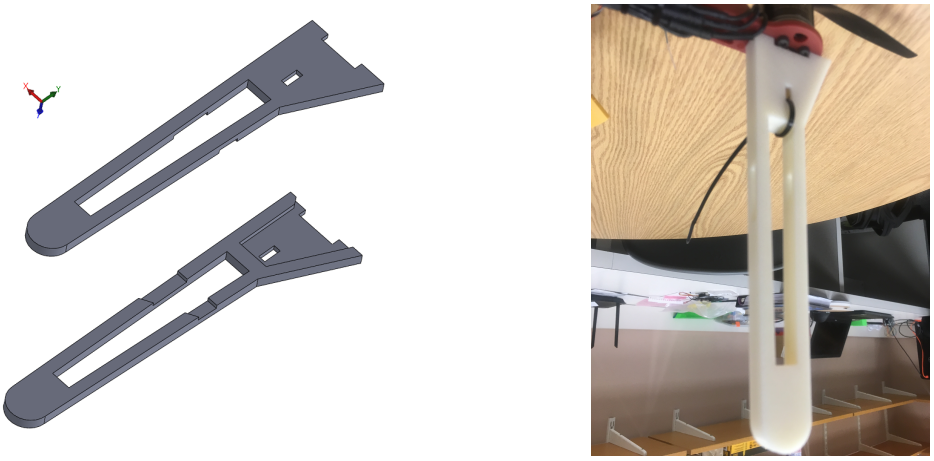
Figure 2·13: Driven gear with potentiometer plugged-in

2.2.3 Protective frame and landing struts

Figure 2.14 illustrates our design for the protective frame, which consists of 8 carbon fiber tubes that are connected using designed connection joint shown in (a).

(a) (b)

Figure 2·14: Protective frame: (a) connection joint (b) connected carbon fiber tubes



(a) (b)

Figure 2·15: Protective frame: (a) 3D model (b) printed part

The landing strut is sized with consideration of the distance that prevents the sensor from crashing on the ground while landing. Two pieces (see Figure 2.17) could be combined to form one landing strut. In the middle part of the landing strut, a reserved hole is serving as a routing space that allows the string pass through and connect the leg of torsion spring in the sensor and the protective frame.

2.3 Electrical circuit design

In order to solve the relative displacement due to the collision with obstacles,it is necessary to design an electrical component that maps the displacement (deflection of torsion springs) from the mechanical system into readable numeric data that could be quantitatively analyzed. To this end, we present below our circuit which realizes the connection between the Arduino and the potentiometers.

2.3.1 Circuit schematic

The circuit schematic is as shown in Figure 2.16. Components R_1 through R_4 refer to the four potentiometers. A first jumper (JP1) is used to wiring the ground and power interface on the Arduino board, while a second jumper (JP2) is allows the connection of the middle pins of potentiometers (floating contacts) to the analog-input interfaces A_0 through A_3 on the Arduino board.

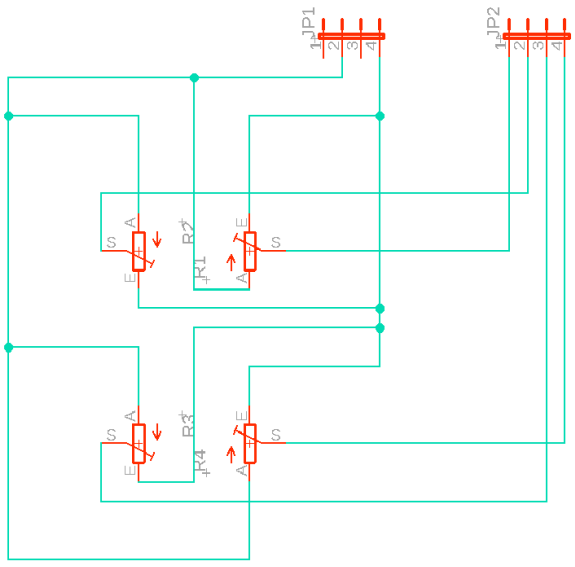
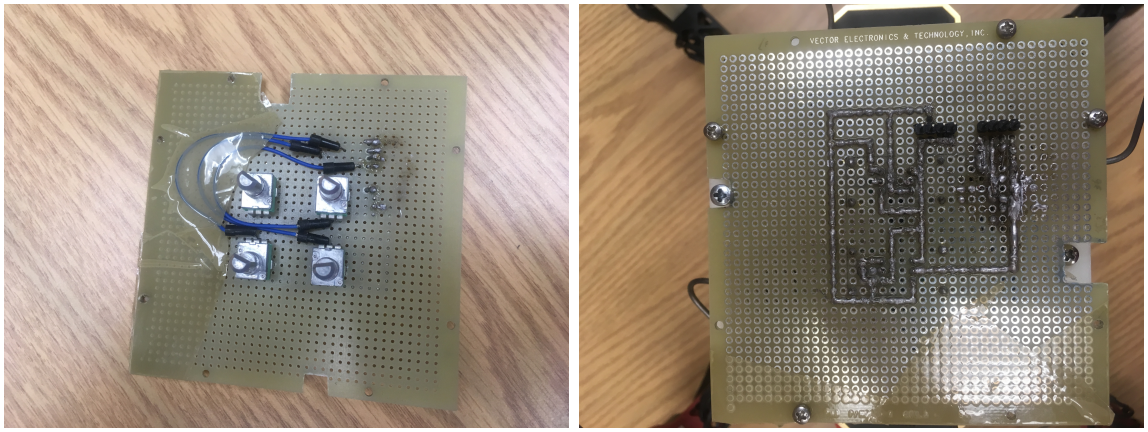


Figure 2·16: Circuit schematic

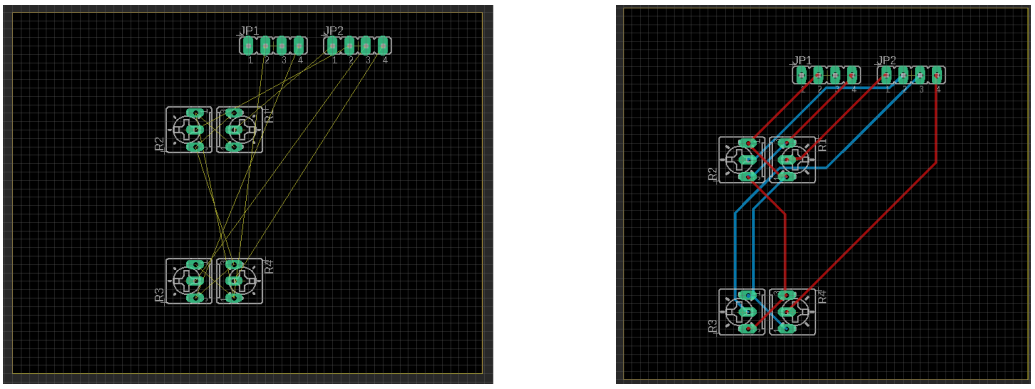
2.3.2 Soldered perf board and PCB layout

After designing the circuit schematic, an equivalent circuit, mapped from Figure 2.17, is prototyped on a soldered perf board. As we can see in (a) shows that four potentiometers are soldered on the bottom side for the mechanical stability, and four jumper wires are used to wire the analog inputs of the Arduino. From (b) a similar layouted circuit is soldered based on the previously presented circuit schematic.



(a) (b)

Figure 2·17: Two sides of soldered perf board: (a). bottom side connecting the encoders (b). top side connecting Arduino



(a) (b)

Figure 2·18: Printed circuit board: (a). under-routing PCB (b). auto-routed PCB

In Figure 2.18, the PCB layout only with signal routes and routed PCB layout are generated based on the circuit schematic in Figure 2.3. In the future, the PCB is expected to be manufactured and used for the sensor, which will help suppress numeric noise in the measurements due to engineering tolerance and system vibrations.

Chapter 3

Mathematical modeling and algorithms

3.1 Introduction

In order to describe the collision-caused relative motion between the protective frame and the sensor, it is necessary to compute the transformation matrices between a coordinate frame fixed in the protective frame and another coordinate frame fixed in the sensor base (reference frame). The 1-D rotation of the protective frame, attached to a certain coordinate reference with respect to the sensor base reference is given by a 2 by 2 rotation matrix. The homogeneous transformation matrix is then used to signify position vectors of the protective frame in a 2-D space along with the rotation matrix of the sensor coordinate.

Similar conventions for the definition of spatial geometric representations for manipulators was first employed (Denavit and Hartenberg, 1995), which was applicable to solve the forward-kinematic problem of a numerous different groups of manipulators.

The primary aim in this chapter is to determine the translation and rotation of the protective frame relative to the vehicle for a given set of torsion spring deflection, that is a typical inverse kinematics problem. In general cases, inverse kinematics problems can be solved by using two types of approaches: The first one is closed form solutions, typically more efficient but system-dependent. Closed-form solutions can be classified in two type of methods: analytical methods (Stifer, 1994) and geometric methods (Fischer, 1990). Analytical methods, also referred to as algebraic methods,

analytically invert the forward kinematics equations; the inverse kinematics problem is then summarized by working out a system of algebraic equations in which transcendental functions, such as trigonometric functions, are substituted with carefully selected replacements (typically polynomial constraints, as in the approach used in this chapter). Geometric methods, for researching manipulators, first identify the relation between the end-effector position, or both position and orientation, in terms of the arm joint parameters. Then such 3-D problem can be treated as separate 2-D problems and solved utilizing algebraic methods.

On the other hand, the second way solving the inverse kinematics problems is numerical approach. Numercial methods are applied to perform approximations to original systems, thus they are not system-dependent. They could deal with, theoretically, any kinematic arrangement; however, it is usually slower than the previously mentioned closed-form methods and very initialization-sensitive. In some cases, it might fail to compute a feasible solution even if there exists due to improper initial input. There are different methods classified as this approach, such as symbolic elimination (Raghavan and Roth, 1990), continuation (Tsai and Morgan, 1985) and iterative methods. Symbolic elimination method eliminates variables from the set of nonlinear equation to convert it into a smaller set of equations. Continuation methods are based on the idea that small changes in the system parameters influence the solution changes. Those methods track a known system to the solution of target system, and approximate the tracked solutions using scheduled mapping scheme (example is predictor-corrector method). Nowadays, the most diffused type of methods for solving the inverse kinematics problems are iterative; examples include Newton-Raphson (Pieper, 1968), optimization approach (Zhao and Badler, 1994) which applies gradient-based nonlinear programming algorithm to deal with equivalent minimization problem, cyclic coordinate descent (Wong and Hu, 2011), pseudoin-

verse (Whitney, 1969), Jacobian transpose (Wolovich and Elliot, 1984), Levenberg-Marquardt damped least squares (Wampler, 1986), quasi-newton and conjugate gradient (Walker and Chen, 1993), and artificial intelligence methods (Lendaris, Oyama, Ramdane and D'Souza 1999-2002).

3.2 Modeling

In order to explore the collision-caused displacement between the protective frame and the vehicle, in this chapter we use two reference frames to illustrate the coordinate transformation between the protective frame and the sensor. The relative displacement is formulated in terms of 3 degrees-of-freedom (2-D translation and 1-D rotation). Our goal is to recover this displacement information from the value of the string displacements recovered by the sensor base (see Chapter 2).

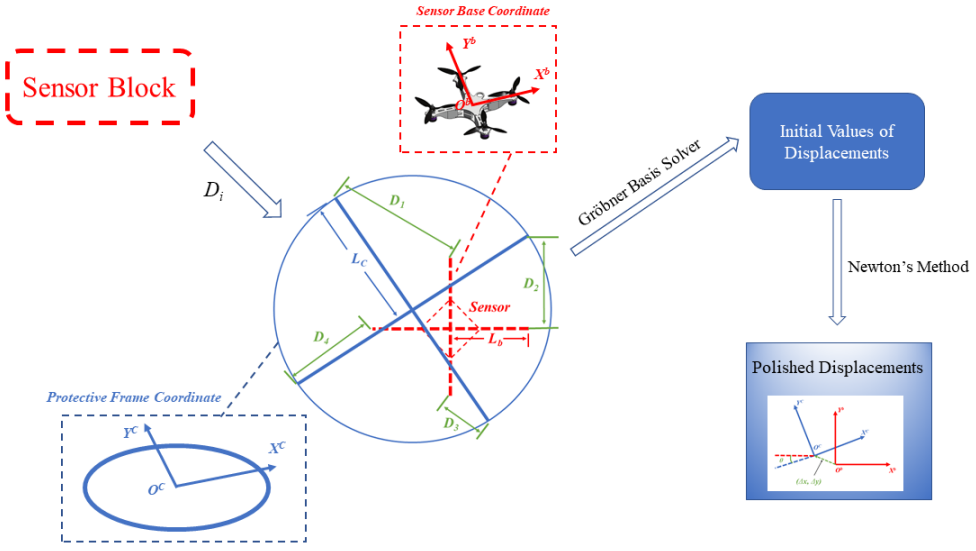


Figure 3·1: Working flow of mathematical model

The interdependence of this chapter is illustrated as Figure 3.1. Based on the input D_i from the sensor block by gathering angle encoders' information and computing the

deflections of four torsion springs, in the first place the displacement is solved using the Gröbner basis solver (Kukelova, Bujnak and Pajdla, 2008). The results are treated as the initial inputs for the following Newton's method to increase the precision of the result.

The mentioned two reference frames with relative displacements are illustrated in Figure 3.2:

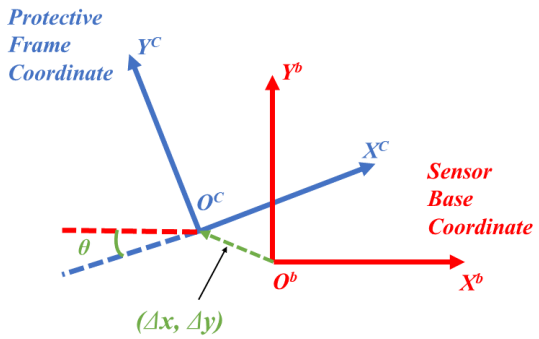


Figure 3·2: $X^bO^bY^b$ and $X^cO^cY^c$

1. The sensor frame ($X^bO^bY^b$) is attached to the base of the sensor, with the origin of the frame located at the geometric center.

2. The cage frame ($X^cO^cY^c$) is fixed on the protective cage, whose geometric center is treated as the origin of the frame.

In absence of any deflection, the origins of the two reference frames overlap, and the x,y axes are aligned.

The relative motion between the sensor frame and the cage frame can be represented by a composition of translation and rotation. The mapping relation between the two frames is as follows:

$$\begin{aligned}\begin{bmatrix} x^b \\ y^b \end{bmatrix} &= \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x^c \\ y^c \end{bmatrix} + \begin{bmatrix} \Delta x_0^c \\ \Delta y_0^c \end{bmatrix} \\ &= \begin{bmatrix} \cos\theta & -\sin\theta & \Delta x_0^c \\ \sin\theta & \cos\theta & \Delta y_0^c \end{bmatrix} \begin{bmatrix} x^c \\ y^c \\ 1 \end{bmatrix}\end{aligned} \tag{3.1}$$

In the matrix equation (3.1) above, θ denotes the relative rotational displacement between the two frames, Δx_o^c and Δy_o^c refer to the translational displacements of the collision-sensing frame with respect to the initial condition without any displacement.

The constant distances between the origin and each routing hole on the landing struts are represented by L^b. For the initial condition without any relative translation and rotation, the distance from each protective attaching point to the origin is expressed by L^c.

Four coordinates of routing holes are defined as follows:

$$\begin{bmatrix} x_1^b \\ y_1^b \end{bmatrix} = \begin{bmatrix} 0 \\ L^b \end{bmatrix} \qquad \begin{bmatrix} x_2^b \\ y_2^b \end{bmatrix} = \begin{bmatrix} L^b \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} x_3^b \\ y_3^b \end{bmatrix} = \begin{bmatrix} 0 \\ -L^b \end{bmatrix} \qquad \begin{bmatrix} x_4^b \\ y_4^b \end{bmatrix} = \begin{bmatrix} -L^b \\ 0 \end{bmatrix}$$

Four coordinates of string attaching point on the protective frame are:

$$\begin{bmatrix} x_1^c \\ y_1^c \end{bmatrix} = \begin{bmatrix} 0 \\ L^c \end{bmatrix} \qquad \begin{bmatrix} x_2^c \\ y_2^c \end{bmatrix} = \begin{bmatrix} L^c \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} x_3^c \\ y_3^c \end{bmatrix} = \begin{bmatrix} 0 \\ -L^c \end{bmatrix} \qquad \begin{bmatrix} x_4^c \\ y_4^c \end{bmatrix} = \begin{bmatrix} -L^c \\ 0 \end{bmatrix}$$

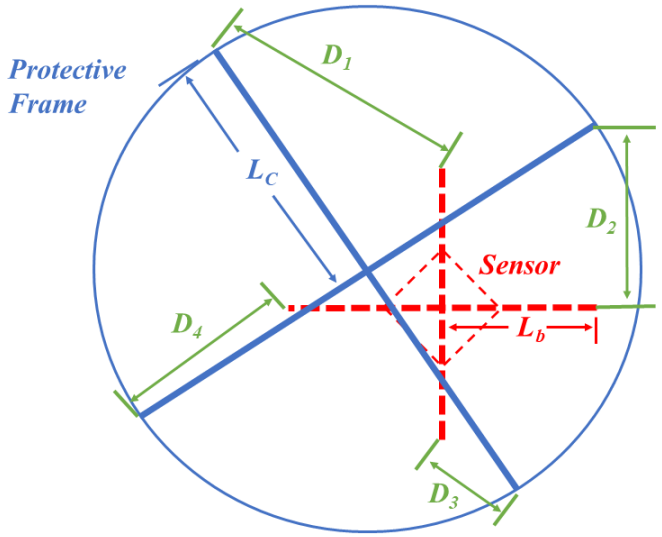


Figure 3·3: Top view of two reference frames

Considering both of translation and rotation, the displacements between the cage terminals and corresponding bearings yield:

$$\begin{cases} D_1^2 = (L^c \sin\theta - \Delta x_0^c)^2 + (L^b - L^c \cos\theta - \Delta y_0^c)^2 \\ D_2^2 = (L^b - L^c \cos\theta - \Delta x_0^c)^2 + (-L^c \sin\theta - \Delta y_0^c)^2 \\ D_3^2 = (-L^c \sin\theta - \Delta x_0^c)^2 + (-L^b + L^c \cos\theta - \Delta y_0^c)^2 \\ D_4^2 = (-L^b + L^c \cos\theta - \Delta x_0^c)^2 + (L^c \sin\theta - \Delta y_0^c)^2 \end{cases} \tag{3.2}$$

where D_i $(i = 1, 2, 3, 4)$ represents the distance from each string attaching point on the frame to the corresponding routing hole (see Figure 3.3).

There are three variables, which are Δx_0^c, Δy_0^c and θ, need solving in the above quadratic equations based on the D_i gathered by the Arduino; in fact the D_i's are functions of torsion springs' deflection that correspond to the value variation of potentiometers and transmission ratio of gear pair. Thus, equations (3.2) represents an overdetermined system of nonlinear equations with trigonometric functions.

For solving the nonlinear system, in this research the original system is modified

by performing linear composition to remove one redundant equation, and adding one more constraint that satisfies the feature of trigonometric functions:

$$\begin{cases} D_3^2 - D_1^2 - 4(L^b\Delta y_0^c - L^c\cos\theta\Delta y_0^c + L^c\Delta x_0^c\sin\theta) = 0 \\ 4[(L^b)^2 + (L^c)^2] - 8L^bL^c\cos\theta + 4[(\Delta x_0^c)^2 + (\Delta y_0^c)^2] - \sum_{i=1}^{4} D_i^2 = 0 \\ D_4^2 - D_2^2 - 4L^b\Delta x_0^c + 4L^c\Delta x_0^c\cos\theta + 4L^c\Delta y_0^c\sin\theta = 0 \\ \sin^2\theta + \cos^2\theta - 1 = 0 \end{cases} \tag{3.3}$$

Considering the values of $\sin\theta$ and $\cos\theta$ as separate variables, the system above represents a system of second-order polynomial equations. We will exploit this fact to find a solution in the next section.

3.3 Solving the initial values using Gröbner basis solver

In order to obtain the initial value for the iterative algorithm in the next section, a Gröbner basis solver, finding solutions to minimal problems (Kukelova, Bujnak and Pajdla, 2008), is applied to generate an action matrix (Kukelova, Bujnak and Pajdla, 2008). The action matrix method, common approach in computer vision, is used to transform, in general, those very difficult problems of nding the solutions to an equivalent eigenvalue/eigenvector problem which is numerically solvable. In this solver (see Appendix Solver), the variables that need solving are set as $\sin\theta, \cos\theta, \Delta x_0^c, \Delta y_0^c$. The eigenvalues solved of the action matrix are the value of term $\cos\theta$, which are two pairs of repeated roots. Only one repeated root that satisfies the bounds on trigonometric functions ($-1 \leq \sin\theta, \cos\theta \leq 1$) is selected as a valid solution. The action matrix M_{action} for equation (3.4) is obtained algebraically using the Matlab toolbox provided by the work of (Kukelova, Bujnak and Pajdla, 2008). As a result, the expression for

the action matrix is found to be as follows:

$$M_{action} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix}, \tag{3.4}$$

where the entries m_{ij}, $i, j = 1, 2, 3, 4$ are given by:

$$\begin{aligned}
m_{11} &= -\frac{t_4 t_6 t_{19}}{32}\left[\sum_{j=8}^{15} t_j - 16t_{16} + t_{18} - 4t_3\sum_{i=1}^{4} D_i^2 - 2(D_1^2 D_3^2 + D_2^2 D_4^2)\right] \\
m_{21} &= \frac{t_4 t_6 t_7 t_{22} t_{23}}{256}, m_{31} = \frac{t_4 t_6 t_{21} t_{22} t_{23}}{256}, m_{41} = 0 \\
m_{12} &= -\frac{t_4 t_6 t_7}{4}, m_{22} = \frac{t_4 t_6 t_{19}}{32}(t_{12} + t_{14} - t_{18} + t_{20} - 2D_1^2 D_3^2) \\
m_{32} &= t_{24}, m_{42} = \frac{t_{19} t_{21} t_{26}}{8} \\
m_{13} &= -\frac{t_4 t_6 t_{21}}{4}, m_{23} = t_{24} \\
m_{33} &= \frac{t_4 t_6 t_{19}}{32}(t_{13} + t_{15} - t_{18} + t_{20} - t_{25}), m_{43} = -\frac{t_7 t_{19} t_{26}}{8} \\
m_{14} &= 0, m_{24} = -\frac{t_{19} t_{21}}{8}, m_{34} = \frac{t_7 t_{19}}{8}, m_{44} = -\frac{t_4 t_{19}}{8}\left[\sum_{i=1}^{4} D_i^2 - 4(t_2 + t_3)\right]
\end{aligned}$$

The intermediate variables t's have expressions as follows:

$$\begin{aligned}
&t_2 = L_b^2, t_3 = L_c^2, t_4 = L_c^{-1}, t_5 = t_2 - t_3, t_6 = t_5^{-1}; \\
&t_7 = D_1^2 - D_3^2, t_8 = 4t_2 D_1^2, t_9 = 4t_2 D_2^2, t_{10} = 4t_2 D_3^2, t_{11} = 4t_2 D_4^2; \\
&t_{12} = D_1^4, t_{13} = D_2^4, t_{14} = D_3^4, t_{15} = D_4^4; \\
&t_{16} = t_2^2, t_{17} = t_3^2, t_{18} = 16t_{17}, t_{19} = L_b^{-1}, t_{20} = 16t_{16}; \\
&t_{21} = D_2^2 - D_4^2, t_{22} = L_b^{-2}; \\
&t_{23} = \sum_{j=8}^{15} t_j + t_{18} + t_{20} - t_{25} - 4t_3\sum_{i=1}^{4} D_i^2 - 32t_2 t_3 - 2D_1^2 D_3^2; \\
&t_{24} = \frac{t_4 t_6 t_7 t_{19} t_{21}}{32}, t_{25} = 2D_2^2 D_4^2, t_{26} = L_c^{-2}
\end{aligned}$$

The eigenvalues of the action matrix provide valid values for $\cos\theta$. The values

for the other unknowns can be obtained by back-substiution (see the next section for details).

3.4 Ferrari's method solving the quartic characteristic equation for the action matrix

Because general eigen-solvers do not work well on the Arduino platform due to resource limitation and insufficient built-in libraries, in this research a specilized method solving the eigenvalue of the 4D matrix is applied. The action matrix generated in the last section is 4 dimensional. In order to compute the eigenvalue of the action matrix, we are using Ferrari's method to solve the quartic characteristic polynomial of the action matrix.

The expression of the action matrix M_{action} has been given in the previous section. Define a general quartic characteristic equation for the action matrix in the following form

$$a\lambda^4 + b\lambda^3 + c\lambda^2 + d\lambda + e = 0 \tag{3.5}$$

The coefficients a, b, c and d in the above characteristic polynomial can be expressed in terms of the matrix entries from the relation

$$\begin{aligned} |M_{action} - \lambda I| &= \begin{vmatrix} m_{11}-\lambda & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22}-\lambda & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33}-\lambda & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44}-\lambda \end{vmatrix} \\ &= a\lambda^4 + b\lambda^3 + c\lambda^2 + d\lambda + e = 0 \end{aligned}$$

The roots of the polynomial (i.e., the four eigenvalues, and the four possible values

for $\cos\theta$), can be found using Ferrari's method (M. Spiegel, 2008):

$$\begin{cases} \lambda_{1,2} = \dfrac{-b + \sqrt{3b^2 - 8ac}}{4a} \\ \\ \lambda_{3,4} = \dfrac{b - \sqrt{3b^2 - 8ac}}{4a} \end{cases} \tag{3.6}$$

As mentioned before, due to the features of the derived equations, only one pair of the repeated eigenvalues has absolute value less than one, as prescribed by the bounds on the trigonometric functions. Denoting as λ_{valid} the value of the valid root, we can then back-substitute the value of $\cos\theta$ in the original system (3.3) to obtain the complete solution:

$$\begin{aligned} \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} &= \begin{bmatrix} \lambda_{valid} \\ \pm\sqrt{1 - \lambda_{valid}} \end{bmatrix} \\ \begin{bmatrix} \Delta x_0^c \\ \Delta y_0^c \end{bmatrix} &= \begin{bmatrix} -4L_c\sin\theta & -4L_b + 4L_c\cos\theta \\ -4L_b + 4L_c\cos\theta & 4L_c\sin\theta \end{bmatrix}^{-1} \begin{bmatrix} -D_3^2 + D_1^2 \\ -D_4^2 + D_2^2 \end{bmatrix} \end{aligned} \tag{3.7}$$

Note that, in general, there are two solutions obtained. To further make sure which one is the correct answer, in real practical application, hardware device IMU (Inertial Measurement Unit) will be required.

3.5 The Newton's method solving nonlinear equations

It is well known that algebraic solvers, such as the one employed in the previous section, might present numerical precision problems. Hence, we use Newton's method to approximate and polish the analytic solution iteratively.

The previously established model is a 4-D nonlinear systems. The general expression of nonlinear equations with four variables and closed-form solution is represented as the following matrix form:

$$\vec{F}(\vec{x}) = \begin{bmatrix} f_1(x_1, x_2, x_3, x_4) \\ f_2(x_1, x_2, x_3, x_4) \\ f_3(x_1, x_2, x_3, x_4) \\ f_4(x_1, x_2, x_3, x_4) \end{bmatrix} = \vec{0} \tag{3.8}$$

where $\vec{x} = [x_1, x_2, x_3, x_4]^T$ is the variable vector that has four entries.

In order to solve the nonlinear system above, here we are using linearization technique and assuming that the local area of the closed-form solution can be linearized. By taking Tayplor exansion for equation (3.8), we can get:

$$\begin{aligned} \vec{F}(\vec{x} + \Delta\vec{x}) &= \begin{bmatrix} f_1(x_1 + \Delta x_1, x_2 + \Delta x_2, x_3 + \Delta x_3, x_4 + \Delta x_4) \\ f_2(x_1 + \Delta x_1, x_2 + \Delta x_2, x_3 + \Delta x_3, x_4 + \Delta x_4) \\ f_3(x_1 + \Delta x_1, x_2 + \Delta x_2, x_3 + \Delta x_3, x_4 + \Delta x_4) \\ f_4(x_1 + \Delta x_1, x_2 + \Delta x_2, x_3 + \Delta x_3, x_4 + \Delta x_4) \end{bmatrix} \\ &\approx \begin{bmatrix} f_1(x_1, x_2, x_3, x_4) + \dfrac{\partial f_1}{\partial x_1}\Delta x_1 + \dfrac{\partial f_1}{\partial x_2}\Delta x_2 + \dfrac{\partial f_1}{\partial x_3}\Delta x_3 + \dfrac{\partial f_1}{\partial x_4}\Delta x_4 \\ f_2(x_1, x_2, \ldots, x_4) + \dfrac{\partial f_2}{\partial x_1}\Delta x_1 + \dfrac{\partial f_2}{\partial x_2}\Delta x_2 + \dfrac{\partial f_2}{\partial x_3}\Delta x_3 + \dfrac{\partial f_2}{\partial x_4}\Delta x_4 \\ f_3(x_1, x_2, x_3, x_4) + \dfrac{\partial f_3}{\partial x_1}\Delta x_1 + \dfrac{\partial f_3}{\partial x_2}\Delta x_2 + \dfrac{\partial f_3}{\partial x_3}\Delta x_3 + \dfrac{\partial f_3}{\partial x_4}\Delta x_4 \\ f_4(x_1, x_2, x_3, x_4) + \dfrac{\partial f_4}{\partial x_1}\Delta x_1 + \dfrac{\partial f_4}{\partial x_2}\Delta x_2 + \dfrac{\partial f_4}{\partial x_3}\Delta x_3 + \dfrac{\partial f_4}{\partial x_4}\Delta x_4 \end{bmatrix} \end{aligned} \tag{3.9}$$

where the vector $\Delta\vec{x}$ represents the small perturbation term. The second order terms have been ignored and the approximation of the nonlinear equations has second order accuracy.

The equation (3.9) could be further expressed as:

$$\vec{F}(\vec{x}+\Delta\vec{x}) \approx \begin{bmatrix} f_1(x_1,x_2,x_3,x_4) \\ f_2(x_1,x_2,x_3,x_4) \\ f_3(x_1,x_2,x_3,x_4) \\ f_4(x_1,x_2,x_3,x_4) \end{bmatrix} + \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} & \frac{\partial f_1}{\partial x_4} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} & \frac{\partial f_2}{\partial x_4} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} & \frac{\partial f_3}{\partial x_4} \\ \frac{\partial f_4}{\partial x_1} & \frac{\partial f_4}{\partial x_2} & \frac{\partial f_4}{\partial x_3} & \frac{\partial f_4}{\partial x_4} \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \\ \Delta x_4 \end{bmatrix} \tag{3.10}$$

$$= \vec{F}(\vec{x}) + J(\vec{x})\Delta\vec{x} = \vec{0}$$

where $J(\vec{x})$ is a Jacobian matrix whose entries are terms of 1st-order differentiation with respect to vector $\vec{x}$.

Then, the rule for iteration is obtained:

$$\begin{aligned} \Delta\vec{x}^{k+1} &= -J^{-1}(\vec{x}^k)\vec{F}(\vec{x}^k) \\ \vec{x}^{k+1} &= \vec{x}^k + \Delta\vec{x}^{k+1} \\ \vec{F}(\vec{x}^{k+1}) &= -J(\vec{x}^{k+1})\Delta\vec{x}^{k+1} \end{aligned} \tag{3.11}$$

where k refers to the level index of iteration time.

The procedure of iteration using Newton's method is illustrated in the following Table 3.1 (δ and ε are the specified threshold value as a convergence condition.):

Pseudo Code for the Iterative Algorithm:

START

1: Assume that it is the k^{th} iteration;

2: Compute Jacobian matrix $J(\vec{x}^k)$;

3: $\Delta\vec{x}^k = -J^{-1}(\vec{x}^k)\vec{F}(\vec{x}^k)$;

4: $\vec{x}^{k+1} = \vec{x}^k + \Delta\vec{x}^{k+1}$;

5: Compute $\vec{F}(\vec{x}^{k+1})$;

6: If $||\vec{F}(\vec{x}^{k+1})|| < \delta$ or $||\Delta\vec{x}^k|| < \varepsilon \Rightarrow$ goto : 8;
Otherwise $\Rightarrow$ goto : 7;

7: $k := k + 1$
$\vec{x}^k := \vec{x}^{k+1}$;
goto: 2;

8: Return the result and break.

END

Table 3.1: Pseudo code of Newton's method

Plug the solution $\begin{bmatrix}\cos\theta & \sin\theta & \Delta x_0^c & \Delta y_0^c\end{bmatrix}^T$ from Gröbner basis solver as the initial value $\vec{x}^0$ into the Newton's iterative algorithm, the nonlinear equations could be re-written as:

$$\begin{aligned}
&\vec{F}(\cos\theta, \sin\theta, \Delta x_0^c, \Delta y_0^c) \\
&= \begin{bmatrix}
D_3^2 - D_1^2 - 4(L^b\Delta y_0^c - L^c\cos\theta\Delta y_0^c + L^c\Delta x_0^c\sin\theta) \\
4[(L^b)^2 + (L^c)^2] - 8L^bL^c\cos\theta + 4[(\Delta x_0^c)^2 + (\Delta y_0^c)^2] - \sum_{i=1}^{4} D_i^2 \\
D_4^2 - D_2^2 - 4L^b\Delta x_0^c + 4L^c\Delta x_0^c\cos\theta + 4L^c\Delta y_0^c\sin\theta \\
\sin\theta^2 + \cos\theta^2 - 1
\end{bmatrix} \\
&= \vec{0}
\end{aligned} \tag{3.12}$$

The corresponding expression of Jacobian matrix is (recalling that $\sin\theta$ and $\cos\theta$

are treated as separate variables):

$$J(\cos\theta, \sin\theta, \Delta x_0^c, \Delta y_0^c) = \begin{bmatrix} 4L^c\Delta y_0^c & -4L^c\Delta x_0^c & -4L^c\sin\theta & 4(L^c\cos\theta - L^b) \\ -8L^bL^c & 0 & 8\Delta x_0^c & 8\Delta y_0^c \\ 4L^c\Delta x_0^c & 4L^c\Delta y_0^c & 4(L^c\cos\theta - L^b) & 4L^c\sin\theta \\ 2\cos\theta & 2\sin\theta & 0 & 0 \end{bmatrix} \tag{3.13}$$

Then, the equations (3.11) could be solved numerically.

3.6 Numerical simulation

In order to verify the fidelity of the built solver to the constructed mathematical model for solving displacement, four tests are carried out numerically in this section. The necessary input D_i^2 for each test case is listed in Table 3.2. As we can see in Table 3.3, the results show that the solver built in this thesis has the accuracy that satisfies the engineering tolerance.

Table 3.2: Simulation results against real values

Input	Test Case 1	Test Case 2	Test Case 3	Test Case 4
L^b	2.5	2.5	16.5	16.5
L^c	5	5	22.5	22.5
D_1^2	8.8349	2.4307	78.033	918.5676
D_2^2	5.2570	12.7345	15.6601	671.8290
D_3^2	7.4279	14.8693	100.1706	656.8869
D_4^2	11.0058	4.5656	162.5438	903.6254

Table 3.3: Simulation results against real values

	Test Case 1		Test Case 2		Test Case 3		Test Case 4	
Variables	Real	Simulation	Real	Simulation	Real	Simulation	Real	Simulation
$\cos\theta$	0.9359	0.935896	0.9888	0.988771	0.9553	0.955337	0.0008	0.000796322
$\sin\theta$	-0.3526	-0.352277	0.1474	0.149436	0.2955	0.29552	-1.0	-1.0
Δx_0^c	-0.32	-0.319993	1.12	1.12	-2.12	-2.12	3.12	3.12
Δy_0^c	0.42	0.42	-0.93	-0.930009	-3.93	-3.93	0.29	0.29

Chapter 4

Particle filter based localization

4.1 Introduction

State estimation problems are of great interest in numerous practical applications. For these kinds of problems, the measured information is put together with prior knowledge concerning the measuring devices and the physical transition in order to estimate the desired variables for dynamical systems, which is implemented by statistically minimizing the error (Maybeck, 1979). For instance, the position estimate of an aerial vehicle could be carried using the integration of its spatial velocity vector over time since its original position. However, this type of estimate is usually subject to drift, and it is necessary to fuse other sensor modalities as body-fixed cameras, indoor motion capture systems, or GPS. The goal of state estimation algorithms is then to handle the combination of model predictions and sensor measurements to obtain estimates of the system variables that have higher fidelity than those possible by using individual modalities alone.

Most modern state estimation approaches are based on Bayesian filtering (Kaipio and Somersalo, 2004). The Bayesian method aims to utilize all the observable information in order to reduce the extent of uncertainty in the current estimate. As updated information is gathered, it is combined with the previous state estimate in a statistically optimal way using Bayes' theorem (Winkler and Somersalo, 2004).

Self-localization is a well known state estimation problem in mobile robotics, and many effective solutions have been proposed. The presence of an initial position guess

strongly conditions the development of a localization algorithm. Whenever a position guess is available, a localization technique has to keep consistent the estimates of the system over time, but it has not to determine the robot location from scratch. This family of techniques goes under the name of position tracking.

The prototype of algorithms proposed to solve position tracking is the famous, widely used in state estimate, Kalman Filter localization (Leonard and Durrant-White, 1991). However, the application of the Kalman filter has limitation that it only deals with linear systems with Gaussian noises. Extensions of the Kalman filter (EKF) were developed (Bishop and Maskell et al., 2004) for handling nonlinear cases using linearization technique. Similarly, Monte Carlo methods have been developed representing the posterior density in terms of randomly generated samples with associated weights. Such Monte Carlo methods, usually denoted as particle filtering, are not limited to the restrictive hypotheses the Kalman filter has. Therefore, particle filtering can be applied to approximate nonlinear models with errors that satisfy non-Gaussian distribution (Doucet et al., 2004).

Localization based on particle filters has been previously proposed (Vlassis et al., 2002) for solving the vision based localization problem, together with a non-parametric estimate of the likelihood function. In their work, they stayed focus on vision based localization using a non-parametric estimate of the likelihood function

while global positioning encloses common frameworks like Multi Hypotheses Localization (Jensfelt and Kristensen, 2001), Histogram Filters (Burgard, Fox and Henning, 1996) and Particle Filters (Burgard, Dellaert and Thrun, 1999). The latters, also known as Monte Carlo Localization (MCL), became one of the most popular approaches for solving the self-localization problem. The framework has been developed for both feature based maps (Jensfelt, Austin, Wijk and Andersson, 2000) and grid based maps (Burgard, Fox and Thrun, 1999).

4.2 State estimation problem

In order to abstract the state estimation problem, a model that represents the evolution of system state vector $\vec{x}$ is considered in the following form:

$$\vec{x}_k = \vec{f}_k(\vec{x}_{k-1}, \vec{v}_{k-1}) \tag{4.1}$$

where $k = 1, 2, \ldots,$ represents a discrete time instant t_k in a dynamical system. The vector $\vec{x} \in R^{n_x}$ is called the system state vector containing the system variables to be estimated dynamically as time proceeds. The vector advances based on the system transition given by equation (4.1), where $\vec{f}$ is, generally, a nonlinear function of the state variables $\vec{x}$ and with a state noise vector $\vec{v} \in R^{n_v}$.

Consider that a measurement vector $\vec{z} \in R^{n_z}$ is available at time step t_k, where the measurements are related to the state vector $\vec{x}$ through the general function $\vec{h}$ represented in the following form:

$$\vec{z}_k = \vec{h}_k(\vec{x}_k, \vec{n}_k) \tag{4.2}$$

where $\vec{n} \in R^{n_z}$ is the noise of observation. Equation (4.2) is referred to as the measurement model.

The aim of the state estimation problem lies in obtaining the state vector $\vec{x}$ based on the state evolution equation (4.2) and on the observation $\vec{z}$ from the observation model (4.1, 4.2) (Doucet et al., 2004).

The evolution-observation model given by equation (4.1, 4.2) are based on the following assumptions (Kaipio and Somersalo, 2004):

1. The state sequence $\vec{x}_k$ for $k = 1, 2, \ldots,$ is a Markov process, which is,

$$P(\vec{x}_k|\vec{x}_0, \vec{x}_1, \ldots, \ldots x_{k-1}) = P(\vec{x}_k|\vec{x}_{k-1}) \tag{4.3}$$

2. The measurement sequence $\vec{z}_k$ for $k = 1, 2, \ldots,$ is a Markov process about the history of the state variables $\vec{x}_k$, that is

$$P(\vec{z}_k|\vec{x}_0, \vec{x}_1, \ldots, \vec{x}_k) = P(\vec{z}_k|\vec{x}_k) \tag{4.4}$$

3. The sequence $\vec{x}_k$ does not depend on the past observations, that is,

$$P(\vec{x}_k|\vec{x}_{k-1}, \vec{z}_1, \vec{z}_2, \ldots, \vec{z}_{k-1}) = P(\vec{x}_k|\vec{x}_{k-1}) \tag{4.5}$$

where $P(A|B)$ denotes the conditional probability of event A when event B is given.

In addition, for the evolution-observation model given by equations (4.1, 4.2), it is assumed that the noise vectors $\vec{v}_i$ and $\vec{v}_j$ as well as $\vec{n}_i$ and $\vec{n}_j$ $(i \neq j)$ are mutually independent, and also independent of the initial state $\vec{x}_0$. For all i and j, the vectors $\vec{v}_i$ and $\vec{n}_j$ are independent as well.

Assuming that $P(\vec{x}_0|\vec{z}_0) = P(\vec{x}_0)$ is available, the posterior probability density $P(\vec{x}_k|\vec{z}_1, \vec{z}_2, \ldots, \vec{z}_k)$ then could be obtained using Bayesian filters in two steps: prediction and update.

4.3 The particle filter

The particle filter method is a Monte Carlo method for solving the state estimation problem. The primary idea is to represent the required posterior density function using a set of random particles with associated weights, and compute the estimates based on these samples and their corresponding weights. The most likely particles, satisfying the current observed system state better, survive as the real system evolves. As the number of samples gets larger, the solution has higher fidelity and approaches the optimal Bayesian estimate. But at the same time, more computing time and computational resources are needed.

In this thesis, the Sequential Importance Sampling (SIS) algorithm for the particle filter is implemented, which includes a resampling step at each time step, as described in detail in reference (Arulampalam and Maskell et al., 2001). The SIS algorithm makes use of an importance density, which is a density proposed to represent another one that cannot be exactly computed, that is, in the localization case, the posterior probability density given previous states and previous measurements. Then, samples are drawn from the importance density instead of the actual density.

A common problem with the SIS particle filter method is the degeneracy phenomenon in which a few states all but one particle is going to have small weight that is negligible. This implies that a large computational effort is devoted to particle swarm updates whose contribution to the approximation of the posterior density function is almost zero. Increasing the number of particles or properly selecting the importance density could overcome such issue.

4.4 Particle filtering models in localization from collisions

In this section we clarify the assumptions we make and what model we use for the implementation of a particle filter for localization from collision information gathered with our cage sensor.

The dynamical system model (4.1) is assumed to be of the following form:

$$\begin{aligned} \vec{x}_k^{sys} &= \vec{f}_k^{sys}(\vec{x}_{k-1}^{sys}, \vec{v}_{k-1}) \\ &= \vec{x}_{k-1}^{sys} + \vec{v}_{k-1} + \vec{u}_{k-1} \end{aligned} \tag{4.6}$$

where vector $\vec{u}$ refers to the motion command given to the robot (which we assumed to be known).

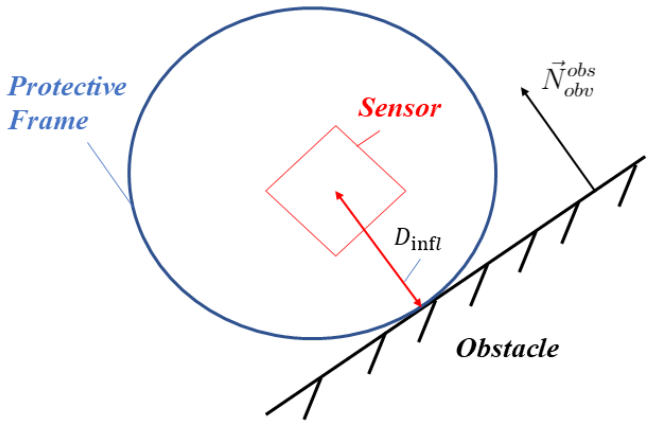


Figure 4·1: Collision description

We assume that we have available a full map of the environment, and that the environment contains polygonal obstacles and a polygonal boundary.

Our observation model in the final simulation includes three parts: Observed position with noise $\vec{z}_k$ at sampling time t_k, influencing distance D_{infl} and obstacle's normal vector $\vec{N}_{obv}^{obs}$ (see Figure 4.1):

$$\begin{cases} \vec{z}_k = \vec{h}_k(\vec{x}_k, \vec{n}_k) = \vec{x}_k^{sys} + \vec{n}_k \\ D_{infl}, \vec{N}_{obv}^{obs} \end{cases} \tag{4.7}$$

Define the particle swarm in the following form:

$$\vec{x}_{0:k}^{i}, i = 0, 1, \ldots, N \tag{4.8}$$

the associated weights are expressed as:

$$w_k^i, i = 0, 1, \ldots, N \tag{4.9}$$

Then, the update rule for particle weight at time t_k can be discretely approximated

by:

$$w_{k+1}^{i} = \delta(\vec{z}_{0:k} - \vec{x}_{0:k})\phi_k^i \psi_k^i \tag{4.10}$$

where $\delta(\cdot)$ is the Gaussian distribution function:

$$\delta_a(\vec{x}) = \frac{1}{a\sqrt{\pi}} e^{-\left(\frac{|\vec{x}|}{a}\right)^2} \tag{4.11}$$

and ϕ_k^i, ψ_k^i are weights relating the expected collision information from a particle $\vec{x}_k^i$ against the actual information recovered by the sensor.

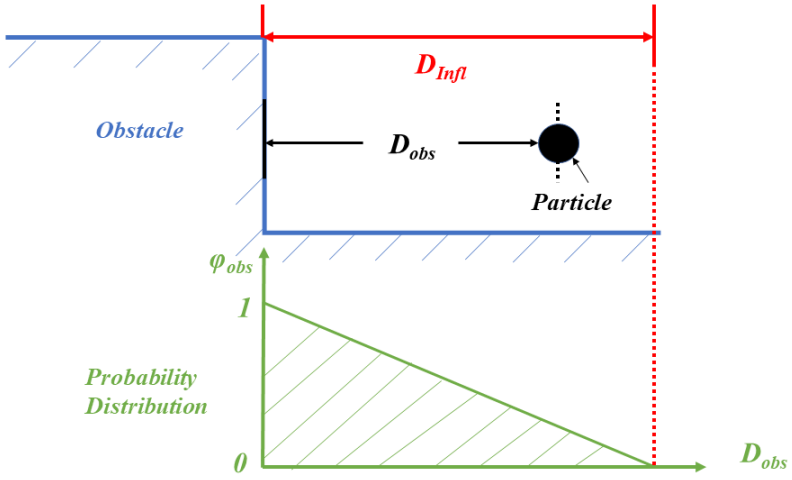


Figure 4·2: Probability distribution depending on distance to obstacle (vehicle with collisions)

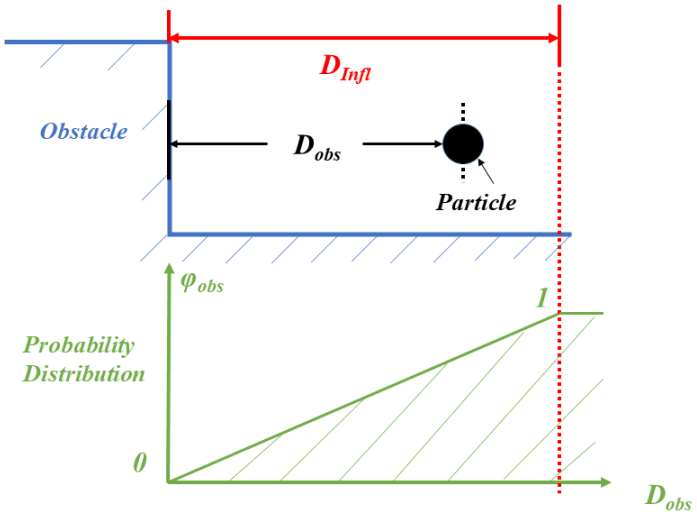


Figure 4·3: Probability distribution depending on distance to obstacle (vehicle without collisions)

More in detail, as shown in Figure 4.2 and 4.3, ϕ_k^i denotes the probability that depends on the collision state of k^{th} system state influences on the weight distribution for the i^{th} particle, that is:

$$\phi(D_{obs}, D_{infl}) = \begin{cases} \max[0, \min(1, \dfrac{D_{obs}}{D_{infl}})], & in\ collision \\ \max[0, \min(1, 1 - \dfrac{D_{obs}}{D_{infl}})], & out\ of\ collision \end{cases} \tag{4.12}$$

where d_{obs} refers to distance of the nearest obstacle edge, d_{infl} is a influencing distance equal the distance between the vehicle and nearest obstacle. radius of protective frame. While the system is not in collision with obstacles, in Figure 4.3, for those particles that represent likely position of the real system state, the closer those particles are to obstacles, the less associated weight they have; for the case with collision in Figure 4.2, the associated weight is inversely proportional to the distance from

particle to the nearest obstacle.

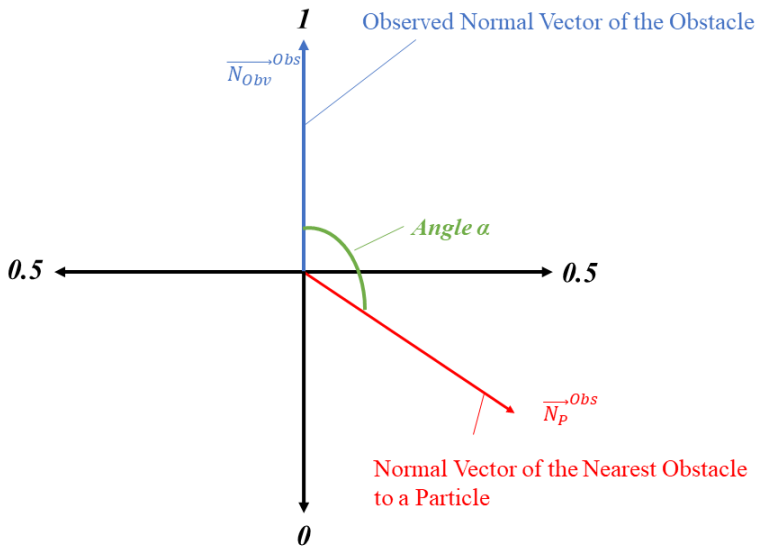


Figure 4·4: Probability distribution depending on obstacle normal vector

The weight ψ_k^i refers to the coefficient that depends on the angle between normal vector of the nearest obstacle edge (see Figure 4.4). The expression is:

$$\psi(\vec{N}_p^{obs}, \vec{N}_{obv}^{obs}) = \frac{\vec{N}_p^{obs} \cdot \vec{N}_{obv}^{obs} + 1}{2} \tag{4.13}$$

where $\vec{N}_p^{obs}$ and $\vec{N}_{obv}^{obs}$ refer to the normal vectors of the nearest obstacle edges to the particle, and to the observation state respectively. From Figure 4.4, we can tell that those particles, whose nearest obstacles' normal vectors have similar direction as observed normal vector, possess higher weight.

Equations (4.13) and (4.14) provide an empirical model to calculate particle weight set from collision information. Therefore, other similar models for updating the weight probably would offer similar simulation results.

In order to avoid the degeneracy phenomenon, the resampling technique is applied.

Resampling involves a mapping of the random measure $\{x_k^i, \vec{w}_k^i\}$ onto a random measure $\{\vec{x_k^{i*}}, N^{-1}\}$ with uniform weights. It is able to be performed if the number of effective particles with large weights become less than a certain convergence value.

The procedure for simulating and evaluating the particle filter in this thesis is as follows:

1. Load information of environment including the shape of obstacles, start point, goal point;

2. Find a feasible path from start point to goal point using A* pathfinding algorithm and create the motion command vector $\vec{u}_{1:k}$;

3. Randomly generate N particles in the free space and initialize the particle weights using Gaussian distribution;

4. Collision check is performed based on the observed information, such as collision status and the normal vector of the nearest obstacle edge, and update each particle weight using equation 4.11 based on the collision status in each sampling time;

5. Resample the particle swarm based on the weights. Effectively, particles that have small weights are eliminated, while new particles will be created around the particles that survived.

6. Advance the time index k used for estimation, and repeat from step 4 until the end of the trajectory.

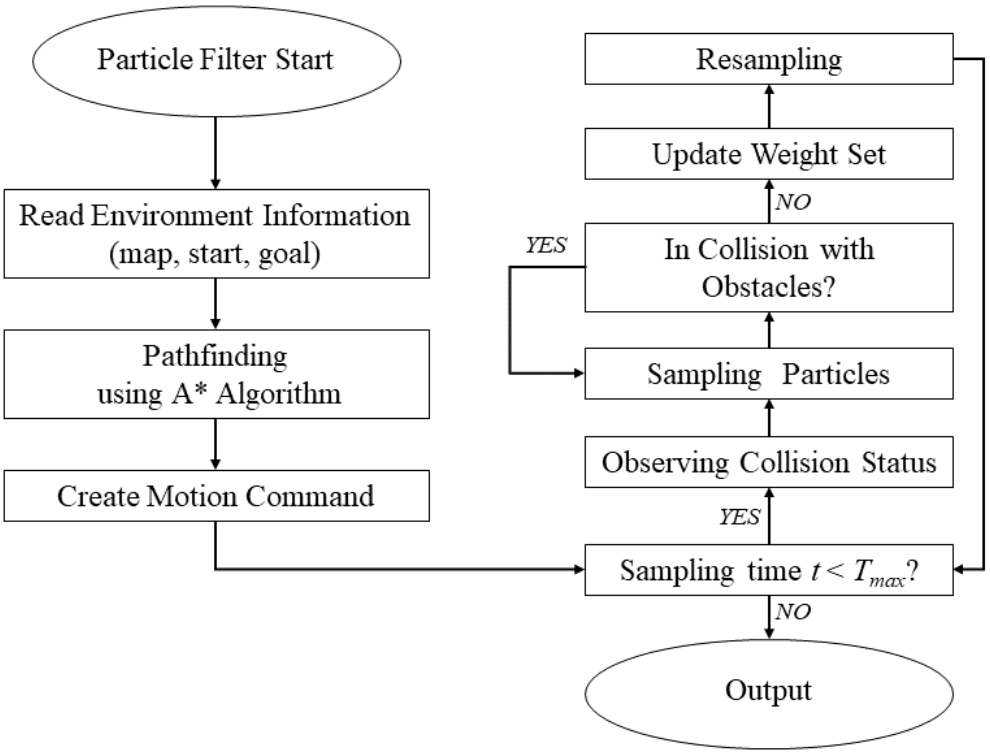


Figure 4·5: Flowchart of particle filter algorithm

Pseudo Code for the Particle Filter Algorithm:

START

$Input(map, start, goal)$;

$Path = Pathfinding(map, start, goal)$;//using $A*$ algorithm

$\vec{u}_{1:k}(Path)$;//create motion command vector

$\vec{x}_0, \vec{w}_0 = particleInit(map, N)$;//initialization

for i **in** $\vec{u}_{1:k}$:

    $collsionFlag, \vec{N}_{obv}^{obs} = Observation()$;//collision check

    for j **in** $\vec{x}_{1:N}$:

        $\vec{x}_j, \vec{w}_j = Prediction(collisionFlag, \vec{N}_{obv}^{obs}, map, \vec{u}_i)$;

        $\vec{x}_j, \vec{w}_j := Resampling(\vec{x}_j, \vec{w}_j)$;

    end

end

END

Resampling Step:

$c_i = c_{i-1} + w_k^i$ where $i = 1, \ldots, N$;//cumulative sum of weights

Let $i = 1$ and randomly select a start point $p_1 \sim U(0, \frac{1}{N})$;

for $j = 1, 2, \ldots, N$:

    $p_j = p_1 + N^{-1}(j-1)$;

    if $u_j > p_i :\ i+ = 1$;

    $x_j = x_i$;//copy samples

    $w_j = N^{-1}$;//weight assignment

end

Table 4.1: Pseudo code for the particle filter and resampling procedure

The pseudo code (Table 4.1) and the flowchart (Figure 4.5) for implementing the particle filter method are shown above.

4.5 Simulation results

Figure 4.6 shows the result after initializing the particle swarm for the first calculated feasible path (blue line). The grey areas refer to obstacles in which particles are not allowed to pass through. The small cross shows the initial position of each particle in the free space. The blue point tells where the geometric center of generated particle swarm is. The red point represents the real system position.

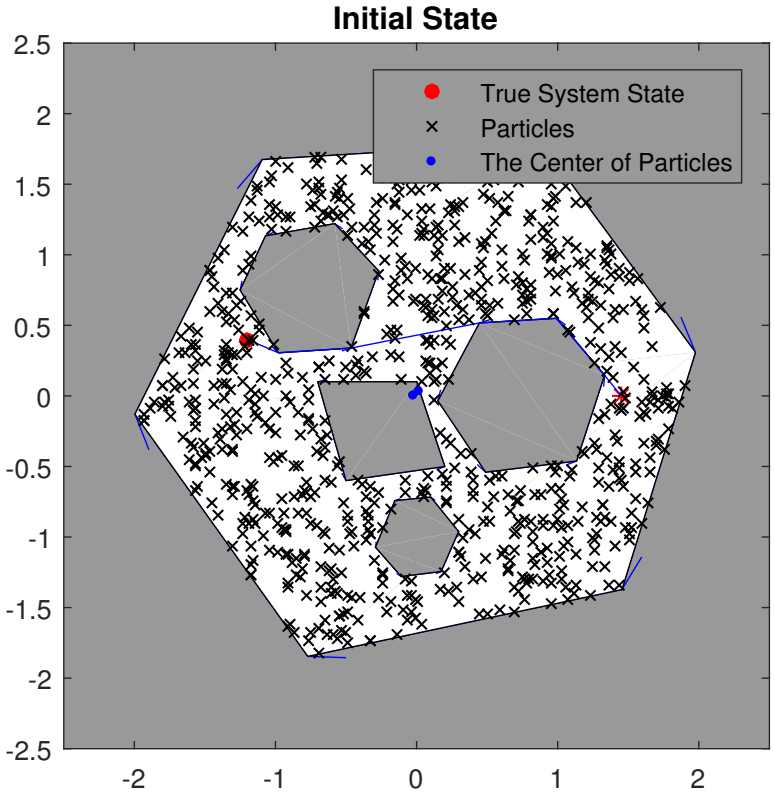


Figure 4·6: Initialization for the 1^{st} path

The process that particle swarm evolves is shown in Figure 4.9. Compared with the initialized result (Figure 4.6), the geometric center of particles (blue points) is approaching the true system position as system evolves, which means the particles start to converge to the system state with a Gaussian noise (green star).

For each sampling time (t_i, $i = 1, 2, \ldots, 6$), it is possible to see the particle swarm can track the real system position in the given known environment gathering the collision information. During the propagation process, it keeps generating new particles based on the weight (probability) in the free space.

Interestingly, looking at sub-figure (b) in Figure 4.9, the geometric center of the has almost overlapped with the real system state. However, there is a small group of particles staying close to the quadrilateral obstacle. The reason is that that edge is almost parallel to the polygon edge near the real system is, thus it cannot be disambiguated until additional measurements are collected.

Figure 4.7 shows the localization error, i.e., the distance between the geometric center of particle swarm and the real system position. The result shows that the magnitude of error is following a decreasing trend, which means particle filter is able to track the real system with an acceptable accuracy as time proceeds.

Figure 4.8 the change of trajectories that belong to the real system state and particle swarm's center. Overall, this section shows that utilizing collision information to perform particle-filter-based localization is feasible.

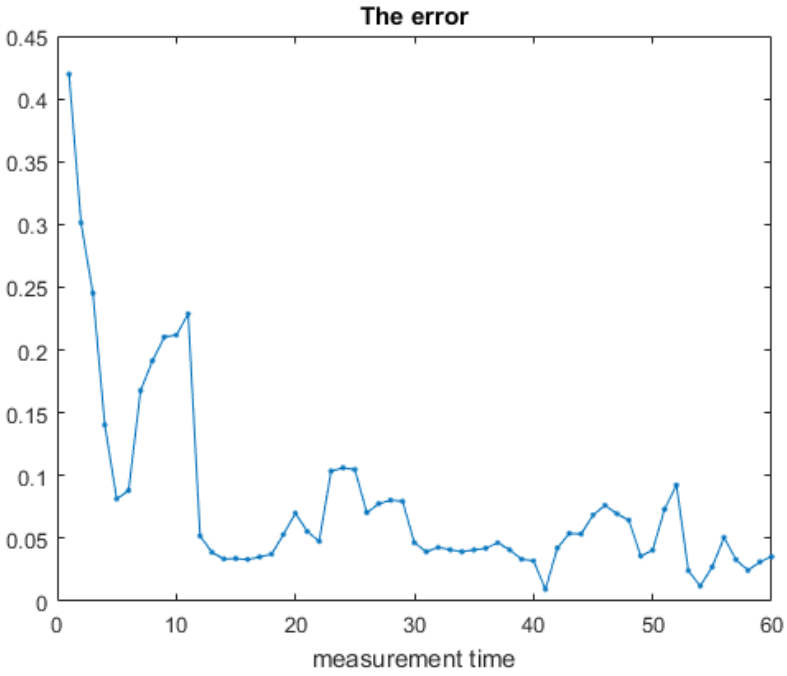


Figure 4·7: Error for path 1

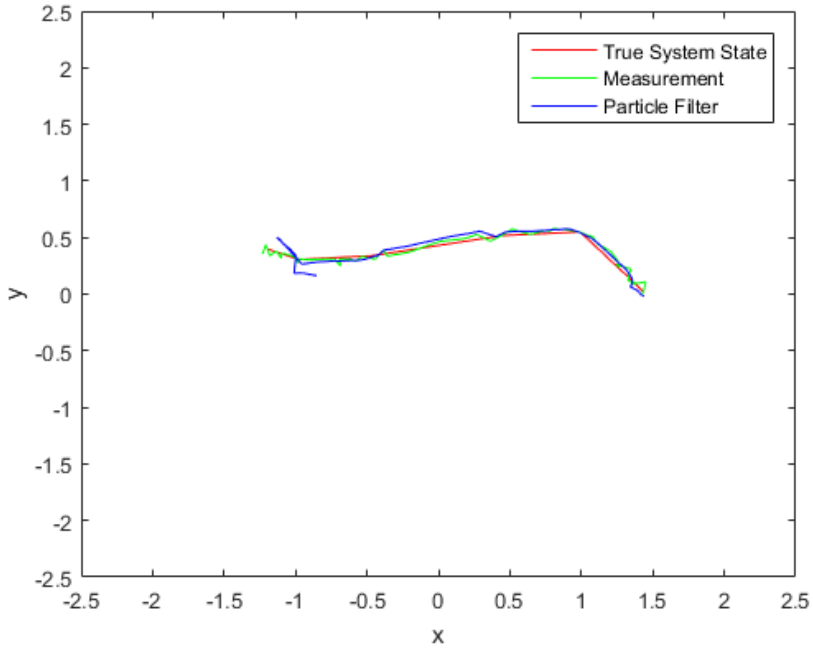
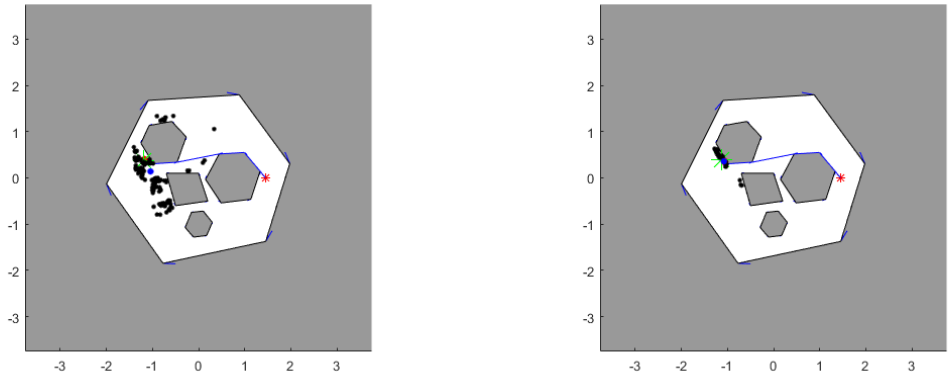
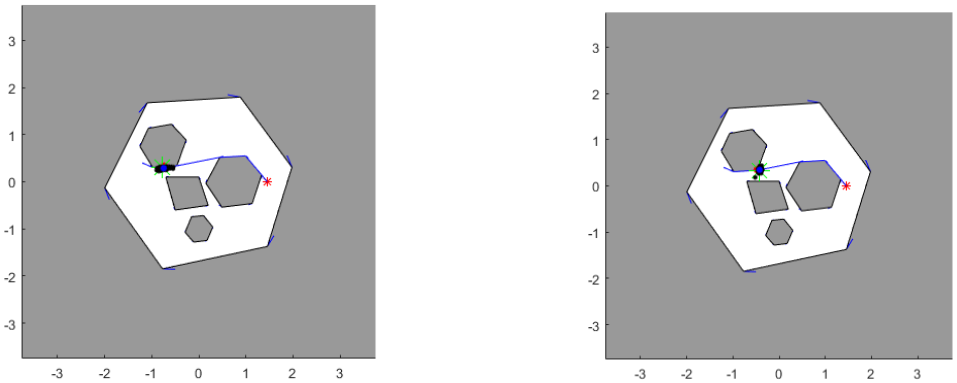


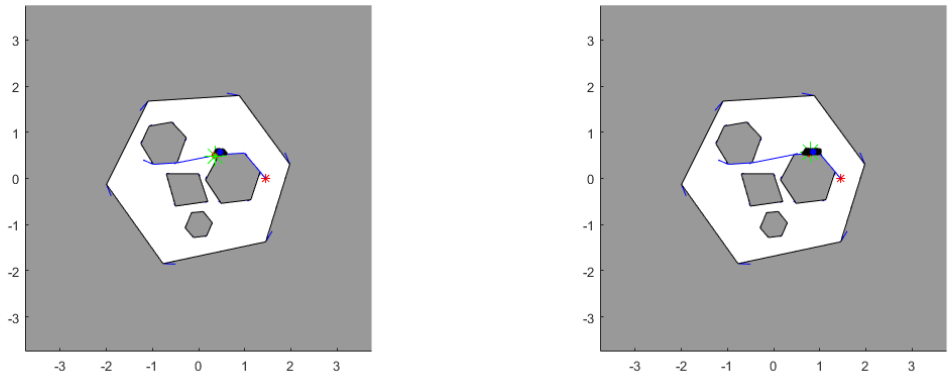
Figure 4·8: Tracking result for path 1



(a) t_1 (b) t_2



(c) t_3 (d) t_4



(e) t_5 (f) t_6

Figure 4·9: Particle propagation for path 1 from time step t_1 to t_6

Chapter 5

Conclusions

5.1 Summary of the thesis

In this thesis, a touch-only sensor is designed and manufactured including mechanical and electrical components for the objective of collision-sensing and relative motion-measuring between the aerial vehicle and the protective frame. The numerical approach to analyze the geometric relation in terms of relative planar translational and 1-D rotational displacement is developed. A numerical experiment is performed for simulating the particle filter-based localization calculating posterior probability in terms of distance to and relative orientation to polygonal obstacles. The test verifies that the application of collision sensor observing the collision direction and distance to the obstacle is theoretically feasible for localization problem in a known 2-D environment utilizing Monte Carlo method.

5.2 Future work

In this research, the theoretical feasibility applying a distance-sensing collision sensor to perform localization in a known 2-D environment has been investigated numerically. The further step is expected to validate the proposed particle filter algorithm and the effectiveness of the designed sensor in practical experiment.

Considering the predictable leading cause of measurement reading issues in flight test, several modifications could be made to improve the collision sensor design. For

the electrical part, currently a soldered perf board is used to sense the torsional springs' deflection resulting from the quadrotor's inertial force while being in collision with obstacles. The PCB with equivalent circuit is expected to be an alternative in the coming real experiment so that the measuring stability could be guaranteed. As to the mechanical design, the transmission ratio of the gear train would be modified based on the magnitude of angle variation as potentiometer shaft rotates; the material of 3D printed part and the mechanical layout are expected to be optimized aiming at minimizing the sensor weight since the sensor is mounted on an unmanned aerial vehicle of which the take-off weight is a crucial parameter.

Last but not least, a sensor-combination could be designed in order to sense displacement of six degrees of freedom (DOFs), which is expected to support positioning and localization in 3-D environments.

Appendix A

Simulation Results

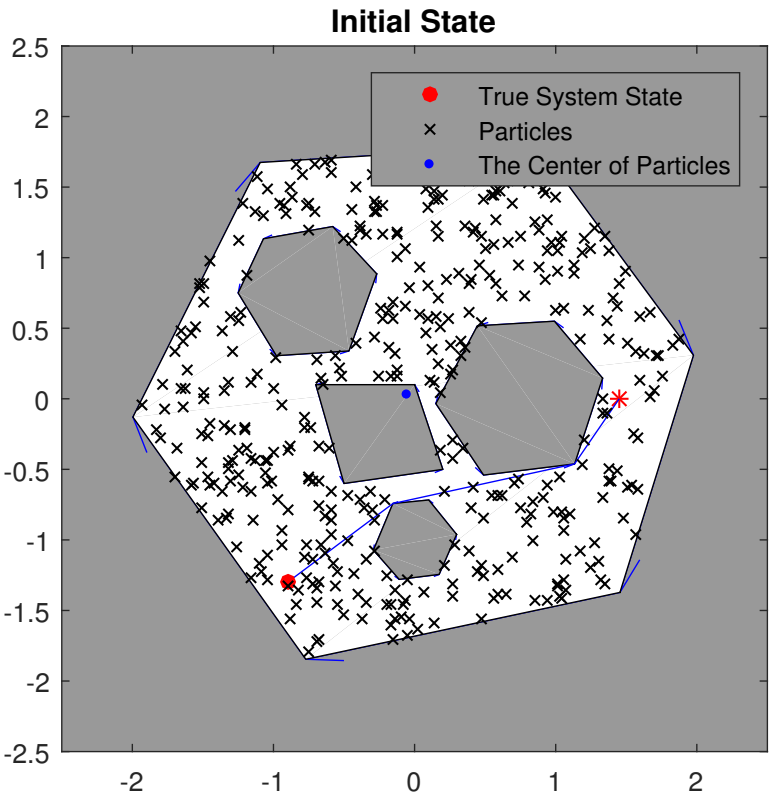
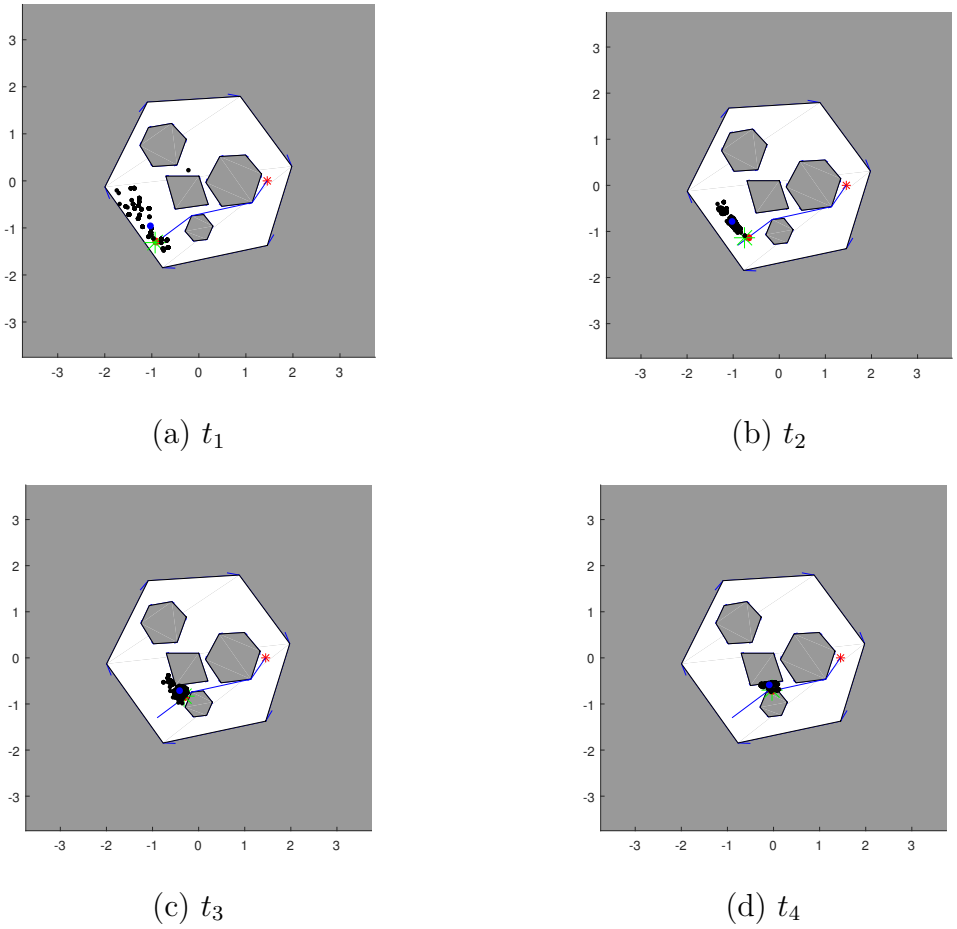


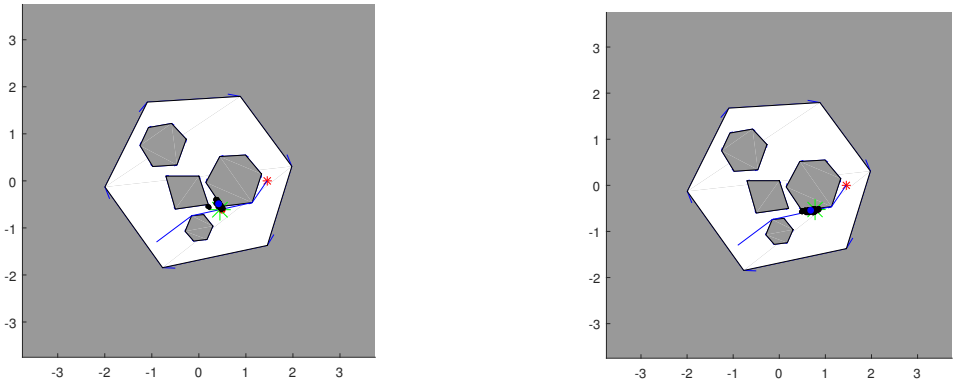
Figure A·1: Initialization for path 2



(a) t_1 (b) t_2

(c) t_3 (d) t_4

Figure A·2: Particle propagation for path 2 from time step t_1 to t_4



(a) t_5 (b) t_6

(c) t_7 (d) t_8

Figure A·3: Particle propagation for path 2 from time step t_5 to t_8

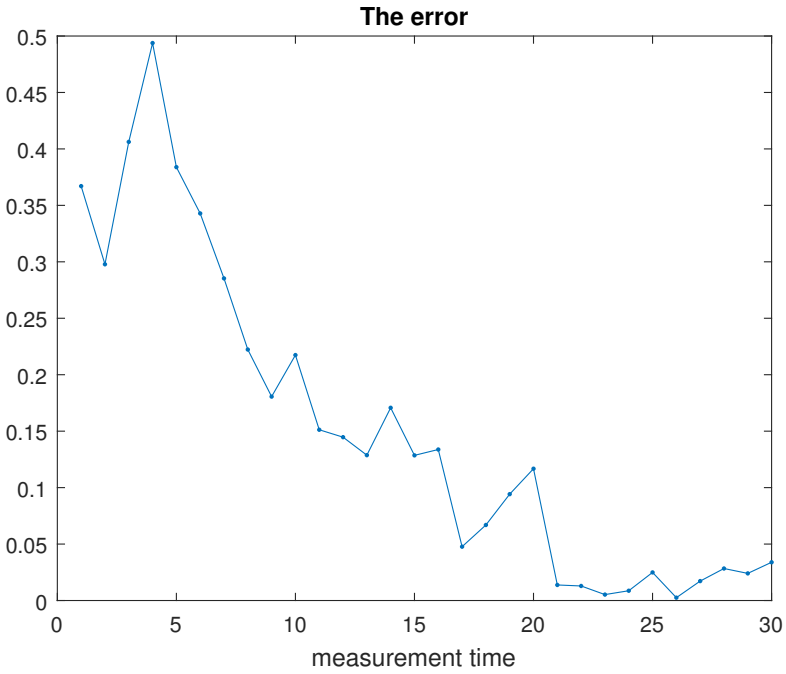


Figure A·4: Error for path 2

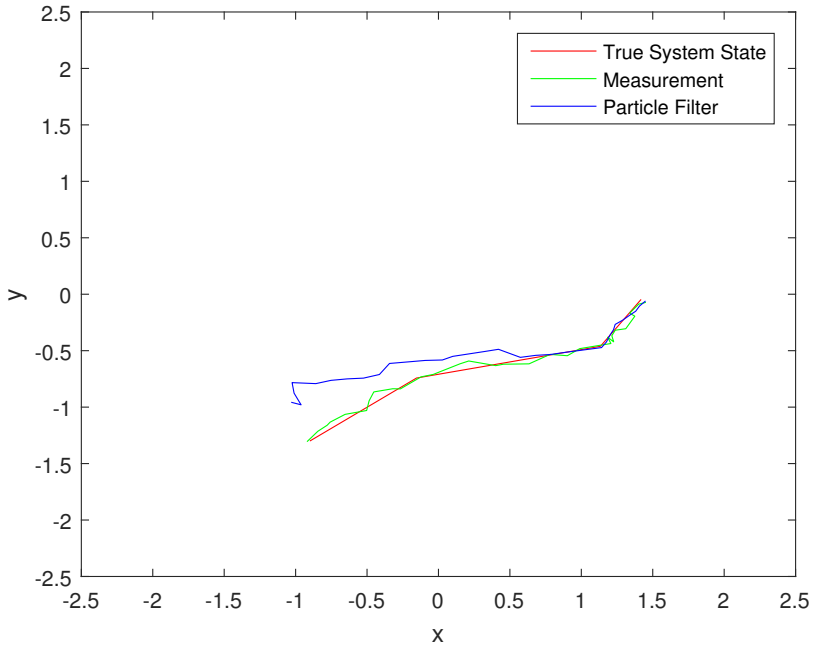


Figure A·5: Tracking result for path 2

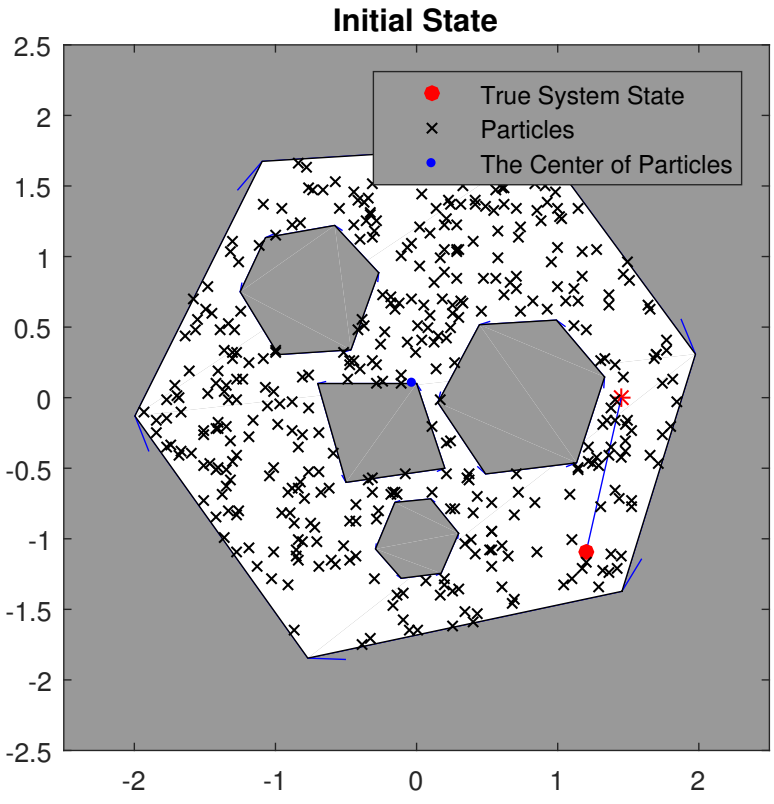
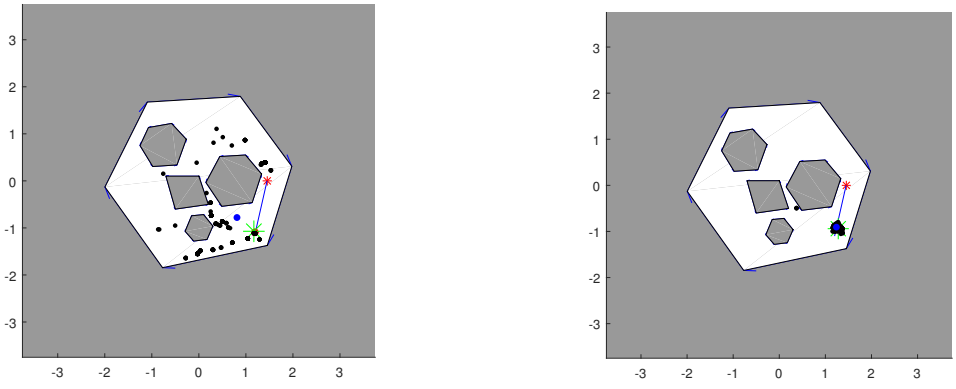


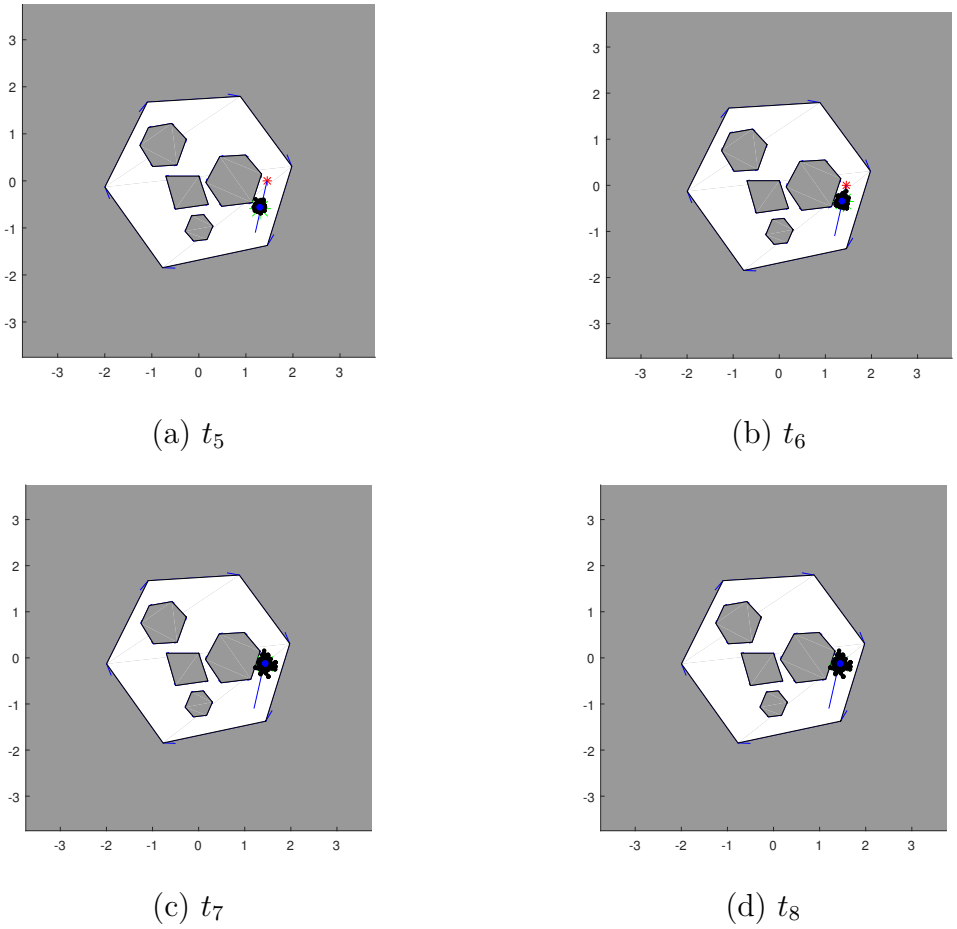
Figure A·6: Initialization for path 3



(a) t_1 (b) t_2

(c) t_3 (d) t_4

Figure A·7: Particle propagation for path 3 from time step t_1 to t_4



(a) t_5 (b) t_6

(c) t_7 (d) t_8

Figure A·8: Particle propagation for path 3 from time step t_5 to t_8

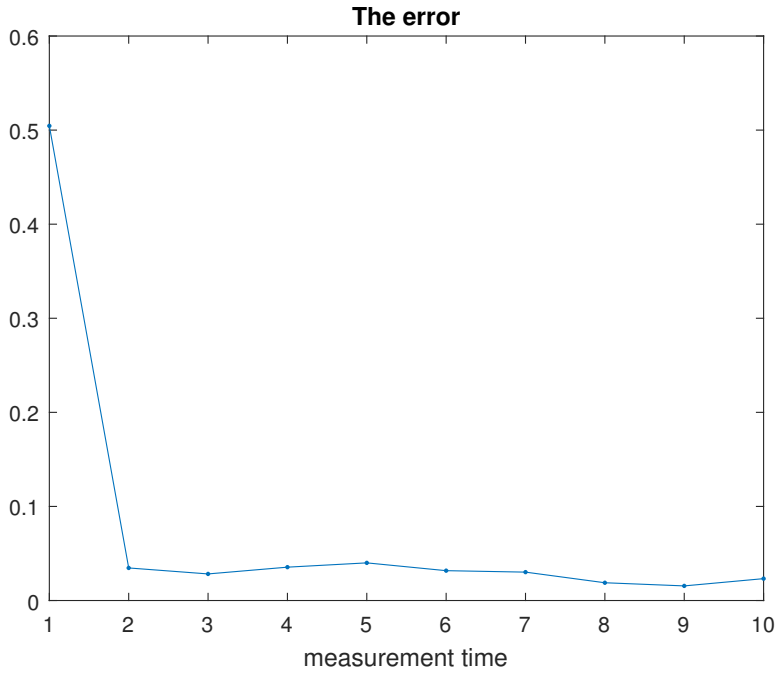


Figure A·9: Error for path 3

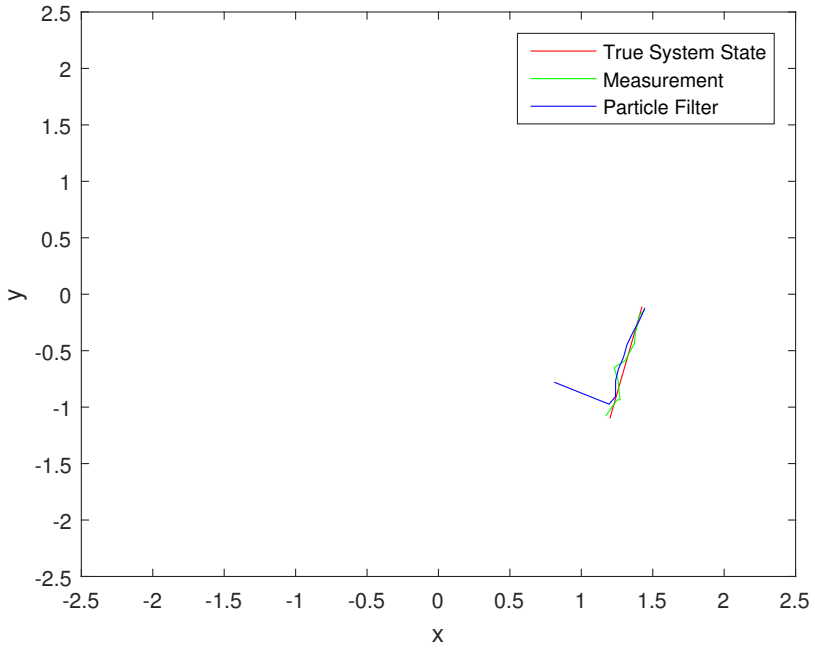


Figure A·10: Tracking result for path 3

Appendix B

Simulation code

B.1 Solver for the Overdetermined System

```
%Build symbolic expressions for computing offsets
function [offsetComputed,symvars]=basefloating_makeModel()
syms Lc Lb real
xc = [0.0 Lc 0.0 -Lc;Lc 0.0 -Lc 0.0];
xb = [0.0 Lb 0.0 -Lb;Lb 0.0 -Lb 0.0];

syms ctheta stheta dx dy real;
R=[ctheta -stheta;stheta ctheta];
T=[dx;dy];
offsetComputed = R*xc + T*ones(1,4) - xb ;

symvars.Lc=Lc;
symvars.Lb=Lb;
symvars.ctheta=ctheta;
symvars.stheta=stheta;
symvars.dx=dx;
symvars.dy=dy;

function [eq,symvars]=basefloating_makeEquations()

%expressions for computing offsets
[offsetComputed,symvars]=basefloating_makeModel();

%sym vars for measurements
Dsquared = gbs_Matrix('Dsquared%d%d',1,4);
symvars.Dsquared=Dsquared;

% build equations
DsquaredComputed=sum(offsetComputed.^2);
eqFull = DsquaredComputed-Dsquared;
eqSimplifiedAuto=simplify(expand(eqFull.'));
eqSimplifiedTrigIdentity=eqSimplifiedAuto-(symvars.Lc^2*symvars.ctheta^2...
+symvars.Lc^2*symvars.stheta^2)+symvars.Lc^2;

%[1 -1 1 -1]'*eqTrigIdentity gives a constant (zero), so we can remove that
%equation
%ASelectIndependent=[1 0 -1 0; 0 1 0 -1; 1 1 1 1];
ASelectIndependent=[1 0 -1 0; 1 1 1 1; 0 1 0 -1];
%ASelectIndependent=[1 1 1 1; 0 1 0 -1; 1 0 -1 0];
%ASelectIndependent=[1 0 -1 0; 0 1 0 -1; 1 1 1 1];
eqIndependent=ASelectIndependent*eqSimplifiedTrigIdentity;
eq=[eqIndependent; symvars.ctheta^2+symvars.stheta^2-1];

function solver_basefloating_test
Lb=2.5;
Lc=5;
%Lc=6.5;

dx=-0.32;
dy=0.42;
%theta=rand;
%dx=0.0;
```

```
%dy=0.0;
theta=-0.36;

ctheta=cos(theta);
stheta=sin(theta);

offsetEq=basefloating_makeModel();
constraintsEq=basefloating_makeEquations();

offset=double(subs(offsetEq,{'Lc','Lb','ctheta','stheta','dx','dy'},...
    {Lc,Lb,ctheta,stheta,dx,dy}));
Dsquared=sum(offset.^2)
constraints=double(subs(constraintsEq,{'Lc','Lb','ctheta','stheta','dx','dy',...
    'Dsquared11','Dsquared12','Dsquared13','Dsquared14'},...
    {Lc,Lb,ctheta,stheta,dx,dy,Dsquared(1),Dsquared(2),Dsquared(3),Dsquared(4)}));
disp('Constraint_residuals_from_real_values')
disp(constraints')

Dsquared(1), Dsquared(2), Dsquared(3), Dsquared(4));
cthetaEstimated = solver_basefloating(Lc, Lb, Dsquared(1), Dsquared(2),...
    Dsquared(3), Dsquared(4));
tol=1e-6;
flagValid=cthetaEstimated>-tol & cthetaEstimated<(1+tol) & (1-cthetaEstimated.^2)>0;
cthetaEstimated=cthetaEstimated(flagValid);

sthetaEstimated=sqrt(1-cthetaEstimated.^2);

%consider ambiguity of sign angle
cthetaEstimated=[cthetaEstimated cthetaEstimated];
sthetaEstimated=[sthetaEstimated -sthetaEstimated];

for iSolution=1:numel(cthetaEstimated)
    sthetaThis=sthetaEstimated(iSolution);
    cthetaThis=cthetaEstimated(iSolution);
    %Linear system from first and third equation in constraintEq
    Adxdy=[-4*Lc*sthetaThis -4*Lb+4*Lc*cthetaThis; -4*Lb+4*Lc*cthetaThis 4*Lc*sthetaThis];
    bdxdy=[-Dsquared(3)+Dsquared(1); -Dsquared(4)+Dsquared(2)];
    %pause;
    dxdy=Adxdy\bdxdy;
    dxEstimated(iSolution)=dxdy(1);
    dyEstimated(iSolution)=dxdy(2);
end

disp('Constraint_residuals_from_solutions')
for iSolution=1:numel(cthetaEstimated)
    constraintsEstimated=double(subs(constraintsEq,{'Lc','Lb','ctheta','stheta','dx','dy',...
    'Dsquared11','Dsquared12','Dsquared13','Dsquared14'},
    {Lc,Lb,cthetaEstimated(iSolution),sthetaEstimated(iSolution),...
    dxEstimated(iSolution),dyEstimated(iSolution),...
    Dsquared(1),Dsquared(2),Dsquared(3),Dsquared(4)}));
    disp(constraintsEstimated')
end

disp('Real_Values')
disp([ctheta; stheta; dx; dy])
disp('Solutions')
disp([cthetaEstimated; sthetaEstimated; dxEstimated; dyEstimated])
```

B.2 Solver for the Overdetermined System

The entire project files including source code of simulation, 3-D models, pictures and documents are listed using Github: https://github.com/cLiu713/master-projectfiles.

References

Andrieu, C., Doucet, A., and Robert, C. (2004a). Computational advances for and from bayesian analysis. *Statistical Science*, 19:118–127.

Andrieu, C., Doucet, P., and Sumeetpal, S. (2004b). Particle methods for charge detection, system identification and control. *Proceedings of the IEEE*, 92:423–438.

Arulampalam, S., Maskell, S., Gordon, N., and Clapp, T. (2001). A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50:174–188.

Burgard, W., Fox, D., and Schmidt, T. (1996). Estimating the absolute position of a mobile robot using position probability grids. *In AAAI'96: Proceedings of 13th National Conference on Artificial Intelligence*, 2:896–901.

Burgard, W., Fox, D., and Thrun, S. (2001). Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128(1):99–141.

Caroti, G., Piemonte, A., Zaragoza, I. M.-E., and Brambilla, G. (2018). Indoor photogrammetry using uavs with protective structures: issues and precision tests. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-3/W4.

Carpenter, J., Clifford, P., and Fearnhead, P. (1999). An improved particle filter for non-linear problems. *IEE Proceedings. Radar, Sonar and Navigation*, 146(1):2–7.

Denavit, J. and Hartenberg, R. (1995). A kinematic notation for lower-pair mechanisms based on matrices. *Journal of Applied Mechanics*, 22:215–221.

Deo, A. S. and Walker, I. D. (1993). Adaptive nonlinear least squares for inverse kinematics. *IEEE International Conference on Robotics and Automation*, pages 193–196.

Doucet, A., Freitas, N., and Gordon, N. (2001). *Sequential Monte Carlo Methods in Practice*. Springer.

Doucet, A., Godsill, S., and Andrieu, C. (2000). On sequential monte carlo sampling methods for bayesian filtering. *Statistics and Computing*, 10:197–208.

D'Souza, A., Vijayakumar, S., and Schaal, S. (2001). Learning inverse kinematics. *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 298–303.

Fischer, I. S. (1990). A geometric method for determining joint rotations in the inverse kinematics of robotic manipulators. *Journal of Robotic Systems*, 17(2):107–117.

Fonseca, H., Orlande, H., Silva, W., and Dulikravish, G. (2010). Kalman filtering for transient source term mapping from infrared images. *Inverse Problems, Design and Optimization Symposium*.

Fox, D., Burgard, W., Dellaert, F., and Thrun, S. (1999). Monte carlo localization: efficient position estimation for mobile robots. *In AAAI'99/IAAI'99: Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference*, pages 343–349.

Gutmann, J. and Fox, D. (2002). An experimental comparison of localization methods continued. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems(IORS)*.

Jensfelt, P., Austin, D., Wijk, O., and Andersson, M. (2000). Feature based condensation for mobile robot localization. *IEEE International Conference on Robotics and Automation*.

Jensfelt, P. and Kristensen, S. (2001). Active global localization for a mobile robot usingmultiple hypothesis tracking. *IEEE Transactions on Robotics and Automation*, 17:748–760.

Jensfelt, P. and Kristensen, S. (2003). An experimental comparison of localization method. *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Kaipio, J., Duncan, S., Seppanen, A., Somersalo, E., and Voutilainen, A. (2005). *Handbook of process imaging for automatic control*. CRC Press.

Kaipio, J. and Somersalo, E. (2004). *Statistical and computational inverse problems*. Springer-Verlag.

Kalantari, A. and Spenko, M. (2013). Design and experimental validation of hytaq, a hybrid terrestrial and aerial quadrotor. *IEEE International Conference on Robotics and Automation (ICRA)*.

Kalman, R. (1960). A new approach to linear filtering and prediction problems. *ASME J. Basic Engineering*, 82:35–45.

Kukelova, Z., Bujnak, M., and Pajdla, T. (2008). Automatic generator of minimal problem solvers. *European Conference on Computer Vision, Part III, Lecture Notes in Computer Science*, 5304:302–315.

Lendaris, G. G., Mathia, K., and Sacks, R. (1999). Linear hopfield networks and constrained optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, 29(1):114–118.

Leonard, J. and Durrant-White, H. (1991). Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7:376–382.

Liu, J. and Chen, R. (1988). Sequential monte carlo methods for dynamical systems. *Journal of the American Statistical Association*, 93:1032–1044.

Maybeck, P. (1979). *Stochastic models, estimation and control*. Academic Press.

Moral, P. D., Doucet, A., and Jasra, A. (2006). Sequential monte carlo samplers. *Journal of the Royal Statistical Society. Series B (Statistical Methdology)*, 68(3):411–436.

Moral, P. D., Doucet, A., and Jasra, A. (2007). Sequential monte carlo for bayesian computation. *Bayesian Statistics*, 8:1–34.

Oyama, E., Chong, N. Y., Agah, A., Maeda, T., and Tachi, S. (2001). Inverse kinematics learning by modular architecture neural networks with performance prediction networks. *IEEE International Conference on Robotics and Automation*, pages 1009–1012.

Pieper, D. (1968). The kinematics of manipulator under computer control. *Ph.D Dissertation, Stanford*.

Raghavan, M. and Roth, B. (1990). Kinematics analysis of the 6r manipulator of general geometry. *5th International Symposium on Robotics Research*.

Ramdane-Cherif, A., Daachi, B., Benallegue, A., and Levy, N. (2002). Kinematic inversion. *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1904–1909.

Song, W. and Hu, G. (2011). A fast inverse kinematics algorithm for joint animation. *International Conference on Advances in Engineering*, pages 350–354.

Sorenson, H. (1970). Least-squares estimation: from gauss to kalman. *IEEE Spectrum*, 7:63–68.

Spiegel, M., Lipschutz, S., and Liu, J. (2008). Mathematical handbook of formulas and tables.

Stifter, A. (1994). Algebraic methods for computing inverse kinematics. *Journal of Intellegent and Robotic Systems*, 11:79–89.

Tsai, L. and Morgan, A. (1985). Solving the kinematics of the most general six and five dof manipulator by continuation methods. *ASME Journal of Mechanisms, Transmissions and Automation in Design*, 107(2):189–200.

Uicker, J., Denavit, J., and Hartenberg, R. (1964). An interactive method for the displacement analysis of spatial mechanisms. *Journal of Applied Mechanics*, 86:309–314.

Vlassis, N., Terwijn, B., and Krose, B. (2002). Auxiliary particle filter robot localization from high-dimensional sensor observations. *IEEE International Conference on Robotics and Automation*.

Wampler, C. W. (1986). Manipulator inverse kinematic solutions based on vector formulations and damped least squares methods. *IEEE Transactions on Systems, Man and Cybernetics*, 16(1):93–101.

Wang, L.-C. T. and Chen, C. C. (1991). A combined optimization method for solving the inverse kinematics problem of mechanical manipulators. *IEEE Transactions on Robotics and Automation*, 7(4):489–499.

Welch, G. and Bishop, G. (2006). *An introduction to the Kalman Filter*. UNC-Chapel Hill.

Whitney, D. E. (1969). Resolved motion rate control of manipulators and human prostheses. *IEEE Transactions on Man-Machine Systems*, 10(2):47–53.

Winkler, R. and Somersalo, E. (2003). *An introduction to Bayesian inference and decision*. Probabilistic Publishing.

Wolovich, W. A. and Elliot, H. (1984). A computational technique for inverse kinematics. *23rd IEEE Conference on Decision and Control*, pages 1359–1363.

Zhao, J. and Badler, N. (1994). Inverse kinematics positioning using nonlinear programming for highly articulated figures. *ACM Transactions on Graphic*, 13(4):313–336.

CURRICULUM VITAE