Title

Montiel Abello

Chapter 1

Introduction

1.1 Literature Review

1.1.1 infinite-dimensional observers

linear:
-observer theory for linear infinite-dimensional systems widely studied
-techniques used typically extensions of luenberger observers used for finite-dimensional systems.
-simplified approach: use spatial discretisation such as finite different/finite element to reduce infinite-dimensional to finite-dimensional observer. [1, 2]
-better to design infinite-dimensional observer and only discretise for numerical implementation. [3, 4, 5] **TODO:** *describe these design methods.*
nonlinear:
-no universal approach for observer design for infinite-dimensional nonlinear systems
-some methods for special case - infinite dimensional bilinear systems. [6, 7]
-for finite-dimensional nonlinear systems, common design methods are: linearisation (ie EKF), lyapunov method, sliding mode, high gain

1.1.2 symmetry preserving observers

1.1.2.1 early work

1.1.2.2 bonnabel et al

1.1.2.3 trumpf, mahony et al

1.1.2.4 juan's work - in detail

1.2 Theoretical Background

1.2.1 Rigid Body Kinematics

A rigid body is a model of a solid object whose deformation is assumed to be negligible. The distance between every pair of points on the body remains constant. Because such a body does not deform, knowledge of the orientation and position of a single point constitutes knowledge of the position and orientation of all points. The position of the rigid body is thus defined as the position of a single point in the body, most commonly its centre of mass. The orientation can be defined using a set of coordinate axes fixed to the body. The theory of Lie groups will be used to describe the kinematics of rigid bodies on this report.

1.2.1.1 Lie Groups

A Lie group $\mathbf{G}$ is a group that is also a differentiable manifold. As a group, $\mathbf{G}$ is a set of elements and a group operation. This group operation is a binary operation that combines two elements and is denoted by multiplication: AB for $A, B \in \mathbf{G}$. Because it is a group, $\mathbf{G}$ satisfies the 4 group axioms:

- **Closure:** The group operation $\mathbf{G} \times \mathbf{G} \mapsto \mathbf{G}$ is a function that maps elements of $\mathbf{G}$ onto itself; $\forall A, B \in \mathbf{G}$, $AB \in \mathbf{G}$.
- **Associativity:** Elements of G are associative under the group operation; $\forall A, B, C \in \mathbf{G}$, $(AB)C = A(BC)$.
- **Identity:** There exists an identity element $I \in \mathbf{G}$ such that $\forall A \in \mathbf{G}$, $IA = AI = A$.
- **Inverse:** For all $A \in \mathbf{G}$ there exists an inverse element $A^{-1} \in \mathbf{G}$ such that $AA^{-1} = A^{-1}A = I$.

Because the Lie group $\mathbf{G}$ is a differentiable manifold, it is locally Euclidean. This means that the neighbourhood around every element of $\mathbf{G}$ can be approximated with a tangent plane. This property allows calculus to be performed on elements of $\mathbf{G}$.

Matrix Lie groups
A matrix Lie group $\mathbf{G} \in \mathbf{GL}(n)$ is made up of group elements which are $n \times n$ matrices. This work will focus on matrix Lie groups because the exponential map and Lie bracket functions given below only apply to such Lie groups.

Lie algebra
The tangent space at the identity element of a Lie group is called the Lie

algebra $\mathfrak{g}$. It is called the Lie *algebra* because it has a binary operation, known as the Lie bracket $[X, Y]$. For matrix Lie groups the Lie bracket is

$$[A, B] \triangleq AB - BA \tag{1.1}$$

The exponential map and logarithm map
The mapping from the Lie algebra $\mathfrak{g}$ to the Lie group $\mathbf{G}$ is called the exponential map:

$$\exp : \mathfrak{g} \to \mathbf{G} \tag{1.2}$$

Similarly, the logarithm map maps elements from $\mathbf{G}$ to $\mathfrak{g}$:

$$\log : \mathbf{G} \to \mathfrak{g} \tag{1.3}$$

For matrix Lie groups, the exponential map and logarithm map correspond to matrix exponential and matrix logarithm respectively.

Infinitesimal generators
For an n-dimensional matrix Lie group, the Lie algebra $\mathfrak{g}$ is a vector space isomorphic to $\mathbb{R}^n$. The hat operator $\hat{}$ maps vectors $x \in \mathbb{R}^3$ to elements of $\mathfrak{g}$.

$$\hat{} : x \in \mathbb{R}^n \to \hat{x} \in \mathfrak{g} \tag{1.4}$$

For a matrix Lie group $\mathbf{G}$ whose elements are $n \times n$ matrices, the elements of $\mathfrak{g}$ will also be $n \times n$ matrices. The hat operator is defined

$$\hat{x} = \sum_{i=1}^{n} x_i G^i \tag{1.5}$$

where G^i are $n \times n$ matrices known as the infinitesimal generators of $\mathbf{G}$.

Lie bracket and group operation
For Lie groups endowned with the commutative property ($\forall A, B \in \mathbf{G}, AB = BA$), vector addition in the Lie algebra maps to a group operation in the Lie group. For $C = A + B$ where $A, B, C \in \mathfrak{g}$,

$$e^C = e^{A+B} = e^A e^B \tag{1.6}$$

For non-commutative Lie groups, the relationship between the Lie bracket and group operation does not hold. Instead, for $C = \log e^A e^B$, C is calculated with the Baker-Campbell-Hausdorff formula:

$$C = A + B + \frac{1}{2}[A, B] + \frac{1}{12}[A - B, [A, B]] + \frac{1}{24}[B, [A, [A, B]]] + \dots \tag{1.7}$$

Actions
When a group action for a Lie group G acting on a manifold M is a differentiable map, this is known as a Lie group action. For example, 3D rotations

act on 3D points so the Lie group $\mathbf{SO}(3)$ acts on $\mathbb{R}^3$. A left action of $\mathbf{G}$ on M is defined as a differentiable map

$$\Phi : \mathbf{G} \times M \mapsto M \tag{1.8}$$

where

- the identity element I maps M onto itself *(is that the right wording?)

$$\Phi(I, m) = m, \forall m \in M \tag{1.9}$$

- Group actions compose according to

$$\Phi(m, \Phi(n, o)) = \Phi(mn, o) \tag{1.10}$$

Adjoint map
EXPLANATION???
For $A \in \mathbf{G}$ and $B \in$? define a function Ψ, known as the adjoint map of $\mathbf{G}$:

$$\Psi_A : \mathbf{G} \to \mathbf{G},\ \Psi_A(B) \triangleq ABA^{-1} \tag{1.11}$$

Taking the derivative:

$$\frac{\partial}{\partial t}\Psi_A(B(t))|_{t=0} = AVA^{-1},\ V \triangleq \frac{\partial}{\partial t}B(t)|_{t=0} \tag{1.12}$$

The adjoint representation of $\mathbf{G}$ is given by the mapping

$$\mathbf{Adj}_A : \mathfrak{g} \to \mathfrak{g},\ \mathbf{Adj}_A(V) \triangleq AVA^{-1} \tag{1.13}$$

1.2.1.2 SO(3)

A rotation represents the motion of a point about the origin of a Euclidean space. In $\mathbb{R}^3$ this is a proper isometry: a transformation that preserves distances between any pair of points and has a determinant of $+1$. The set of all rotations about the origin of $\mathbb{R}^3$ is known as the *special orthogonal group* $\mathbf{SO}(3)$. Group elements of $\mathbf{SO}(3)$ can be represented using a special subset of 3×3 invertible matrices and in this case, forms a matrix Lie group. Several rotation representations are described in REF, but the theory presented below only applies to matrix Lie groups which rely on the rotation matrix representation for group elements.

Lie algebra
The Lie algebra $\mathfrak{so}(3)$ is vector space whose elements correspond to an angular velocity. These elements can be represented with 3×3 skew-symmetric matrices $[\boldsymbol{\omega}]_\times$, where $\boldsymbol{\omega}$ is a 3-vector representing an angular

velocity. Elements of $\mathfrak{so}(3)$ are mapped to $\mathbf{SO}(3)$ according to the exponential map:

$$\begin{aligned} \exp : \mathfrak{so}(3) &\rightarrow \mathbf{SO}(3) \\ [\boldsymbol{\omega}]_\times &\rightarrow \mathbf{R}_{3x3} \end{aligned} \tag{1.14}$$

i.e. $\forall \omega \in \mathfrak{so}(3),\ \exp([\omega]_\times) \in \mathbf{SO}(3)$

Conversely, the logarithm map maps 3×3 rotation matrices of $\mathbf{SO}(3)$ to elements of $\mathfrak{so}(3)$:

$$\begin{aligned} \log : \mathbf{SO}(3) &\rightarrow \mathfrak{so}(3) \\ \mathbf{R}_{3x3} &\rightarrow [\boldsymbol{\omega}]_\times \end{aligned} \tag{1.15}$$

i.e. $\forall \mathbf{R} \in \mathbf{SO}(3),\ \log(\mathbf{R}) \in \mathfrak{so}(3)$

Actions
By the group action, elements of $\mathbf{SO}(3)$ rotate points in $\mathbb{R}^3$ about the origin.

$$\Phi : \mathbf{SO}(3) \times \mathbf{R}^3 \mapsto \mathbf{R}^3 \tag{1.16}$$

Adjoint map
EXPLANATION???? Hard to explain practical application without discussing reference frames: ie if position defined in body fixed frame but some other transformation defined in inertial frame. First undo rotation to get pose in inertial frame, apply transformation, then re-apply rotation.

$$\Psi_R : \mathbf{SO}(3) \rightarrow \mathbf{SO}(3),\ \Psi_R(A) \triangleq RAR^{-1} \tag{1.17}$$

Taking the derivative:

$$\frac{\partial}{\partial t}\Psi_R(A(t))|_{t=0} = RBR^{-1},\ B \triangleq \frac{\partial}{\partial t}A(t)|_{t=0} \tag{1.18}$$

The adjoint representation of $\mathbf{SO}(3)$ is given by the mapping

$$\mathbf{Adj}_R : \mathfrak{so}(3) \rightarrow \mathfrak{so}(3),\ \mathbf{Adj}_R(B) \triangleq RBR^{-1} \tag{1.19}$$

Rotation representations
There are many conventions by which elements of $\mathbf{SO}(3)$ can be represented. The representations that will be used in this report are described below.

Rotation matrices
A 3D rotation matrix $\mathbf{R}$ is an orthogonal 3×3 matrix with a determinant of $+1$. Since $\mathbf{R}$ is orthogonal, its columns and rows are respectively sets of orthogonal unit vectors and

$$\mathbf{R}^{-1} = \mathbf{R}^T \tag{1.20}$$

The group operation using rotation matrices is simply a matrix multiplication which concatenates the two rotations. The product of two rotation matrices $\mathbf{R}_3 = \mathbf{R}_2\mathbf{R}_1$ is a rotation matrix corresponding to left multiplication by $\mathbf{R}_1$ followed by $\mathbf{R}_2$.

The left action of a rotation matrix $\mathbf{R}$ on a point $\mathbf{p} \in \mathbb{R}^3$ is a left matrix multiplication that rotates $\mathbf{p}$ about the origin.

Scaled-axis representation
An orientation in $\mathbf{R}^3$ can also be represented by a 3-vector $\boldsymbol{\theta}$ whose direction $\mathbf{r}$ represents the axis of rotation and magnitude θ represents the angle of rotation.

$$\boldsymbol{\theta} = \theta\mathbf{r} \tag{1.21}$$

Though scaled-axis vectors are not typically used to perform rotations, Rodrigues' rotation formula efficiently converts scaled-axis vectors to rotation matrices:

$$\mathbf{R}_{\boldsymbol{\theta}} = \mathbf{I} + [\mathbf{r}]_\times \sin\theta + ([\mathbf{r}]_\times)^2(1 - \cos\theta) \tag{1.22}$$

Elements of $\mathfrak{so}(3)$ are typically represented with a scaled-axis vector $\boldsymbol{\omega}$ where the magnitude $|\boldsymbol{\omega}|$ corresponds to the angular velocity about the axis $\boldsymbol{\omega}/|\boldsymbol{\omega}|$.

Rotation quaternions
Quaternions are an extension of complex numbers. The set of unit quaternions can be used to represent $\mathbf{SO}(3)$, and will be referred to as rotation quaternions. A rotation quaternion $\mathbf{q}$ is a 4-tuple of real numbers that encode the same information as axis angle. $\mathbf{q}$ is often described in terms of its first element w - the scalar part, and the remaining elements x, y and z - the vector part. Given an axis of rotation $\mathbf{r}$ and an angle of rotation θ:

$$\mathbf{q} = \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} w \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} \cos(\theta/2) \\ \sin(\theta/2)\mathbf{r} \end{bmatrix} \tag{1.23}$$

In general, the quaternion inverse is given by

$$\mathbf{q}^{-1} = \frac{1}{w^2 + x^2 + y^2 + z^2} \begin{bmatrix} w \\ -x \\ -y \\ -z \end{bmatrix} \tag{1.24}$$

For unit magnitude rotation quaternions the inverse represents a rotation

by $-\theta$ and is given by

$$\mathbf{q}^{-1} = \begin{bmatrix} \cos(\theta/2) \\ -\sin(\theta/2)\mathbf{r} \end{bmatrix} = \begin{bmatrix} w \\ -x \\ -y \\ -z \end{bmatrix} \tag{1.25}$$

The group operation is performed with quaternion multiplication which is defined:

$$\mathbf{q}_1\mathbf{q}_2 = \begin{bmatrix} w_1 \\ \mathbf{v}_1 \end{bmatrix} \begin{bmatrix} w_2 \\ \mathbf{v}_2 \end{bmatrix} = \begin{bmatrix} w_1 w_2 - \mathbf{v}_1 \cdot \mathbf{v}_2 \\ w_1\mathbf{v}_2 + w_2\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2 \end{bmatrix} \tag{1.26}$$

As with rotation matrices, quaternion multiplication is associative but not commutative.

The group action rotates a point $\mathbf{p} \in \mathbb{R}^3$ to $\mathbf{p}'$ by embedding it as the vector part of a quaternion and using a conjugation operation with $\mathbf{q}$: formula for rotating vector:

$$\begin{bmatrix} 0 \\ \mathbf{p}' \end{bmatrix} = \mathbf{q} \begin{bmatrix} 0 \\ \mathbf{p} \end{bmatrix} \mathbf{q}^{-1} \tag{1.27}$$

1.2.1.3 SE(3)

The special Euclidean group $\mathbf{SE}(3)$ represents rigid transformation in $\mathbb{R}^3$. This is a matrix Lie group whose elements are the set of all rigid transformations in $\mathbb{R}^3$ and can be represented with 4×4 matrices of the form

$$\mathbf{X} = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{1.28}$$

where $\mathbf{R} \in \mathbf{SO}(3)$ and $\mathbf{p} = \begin{bmatrix} p_x & p_y & p_z \end{bmatrix}^\top \in \mathbb{R}^3$.

$\mathbf{SE}(3)$ is a semidirect product of $\mathbf{SO}(3)$ and $\mathbb{R}^3$. As its group elements contain a rotation matrix and translation vector, $\mathbf{SE}(3)$ has 6 degrees of freedom and is a 6-dimensional manifold.

Lie algebra
The Lie algebra $\mathfrak{se}(3)$ is a vector space whose elements are 4×4 matrices of the form

$$\begin{bmatrix} [\omega]_\times & \mathbf{v} \\ \mathbf{0}_{1\times 3} & 0 \end{bmatrix} \tag{1.29}$$

where $\omega = \begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^\top \in \mathbf{so}(3)$, representing an angular velocity in scaled axis representation, and $\mathbf{v} = \begin{bmatrix} v_x & v_y & v_z \end{bmatrix}^\top \in T_{\mathbf{p}}\mathbb{R}^3$, representing a linear velocity vector.

Elements of $\mathfrak{se}(3)$ are mapped to $\mathbf{SE}(3)$ according to the exponential map:

$$\begin{gathered} \exp : \mathfrak{se}(3) \to \mathbf{SE}(3) \\ \begin{bmatrix} [\omega]_\times & \mathbf{v} \\ \mathbf{0}_{1\times 3} & 0 \end{bmatrix} \to \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \end{gathered} \tag{1.30}$$

i.e. $\forall \mathbf{T} \in \mathfrak{se}(3),\ \exp(\mathbf{T}) \in \mathbf{SE}(3)$

Conversely, the logarithm map maps elements of $\mathbf{SE}(3)$ to elements of $\mathfrak{se}(3)$:

$$\begin{gathered} \log : \mathbf{SE}(3) \to \mathfrak{se}(3) \\ \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \to \begin{bmatrix} [\omega]_\times & \mathbf{v} \\ \mathbf{0}_{1\times 3} & 0 \end{bmatrix} \end{gathered} \tag{1.31}$$

i.e. $\forall \mathbf{S} \in \mathbf{SE}(3),\ \log(\mathbf{S}) \in \mathfrak{se}(3)$

Actions
$\mathbf{SE}(3)$ group elements acts to perform a rigid transformation on points in $\mathbb{R}^3$. This corresponds to a rotation about the origin and a translation. To apply a transformation using the 4×4 matrix elements of $\mathbf{SE}(3)$ to a point $\mathbf{p} = (x, y, z)$ in $\mathbb{R}^3$, the point must be represented with homogeneous coordinates: (is p' okay for homogeneous points? $\hat{}$ is already used for skew-symmetric matrix)

$$\mathbf{p}' = \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{1.32}$$

The left group action of $\mathbf{SE}(3)$ is now simply a left matrix multiplication of $\mathbf{p}$:

$$\mathbf{p'}_1 = \mathbf{S}\mathbf{p'}_0 = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{p}_0 \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}\mathbf{p}_0 + \mathbf{p} \\ 1 \end{bmatrix} \tag{1.33}$$

Adjoint Map
EXPLANATION
The adjoint map of $\mathbf{SE}(3)$ is

$$\Psi_S : \mathbf{SE}(3) \to \mathbf{SE}(3),\ \Psi_S(A) \triangleq SAS^{-1} \tag{1.34}$$

Taking the derivative:

$$\frac{\partial}{\partial t}\Psi_S(A(t))|_{t=0} = SBS^{-1},\ B \triangleq \frac{\partial}{\partial t}A(t)|_{t=0} \tag{1.35}$$

The adjoint representation of $\mathbf{SE}(3)$ is given by the mapping

$$\mathbf{Adj}_S : \mathfrak{se}(3) \to \mathfrak{se}(3),\ \mathbf{Adj}_S(B) \triangleq SBS^{-1} \tag{1.36}$$

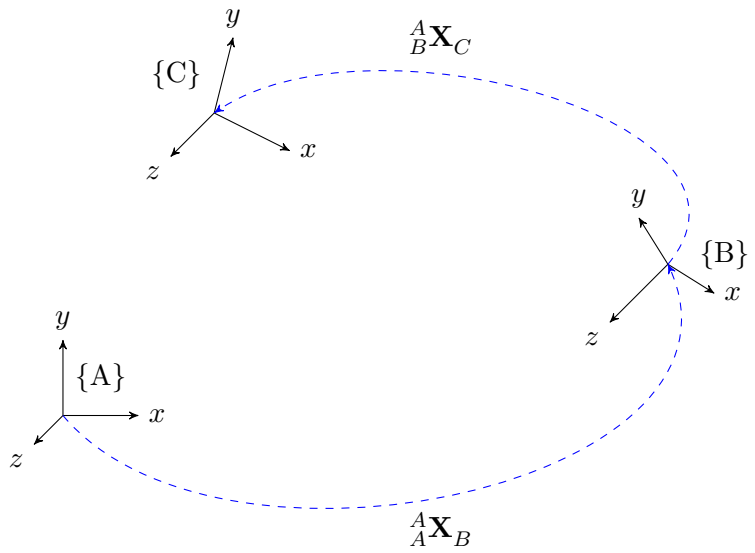


Figure 1.1: what frame should transformations be defined in?

1.2.1.4 Reference Frames

A reference frame is a system of coordinates that is used to uniquely identify points on a manifold. This report will deal with reference frames on $\mathbb{R}^3$, that are used both to define the position of a point and the pose of a rigid body in 3D space. Such a reference frame is represented by an element of $\mathbf{SE}(3)$.

Consider three different reference frames, denoted {A},{B} and {C}. The notation ${}^{A}_{B}\mathbf{X}_{C}$ defines the transformation in $\mathbf{X}$ of the reference frame {C} with respect to the frame {B}, defined in the frame {A}.

For example, ${}^{A}_{B}\mathbf{R}_{C}$ defines the rotation of {C} with respect to {B}, defined in {A}.

The notion of an inertial reference frame is introduced here. This will be defined as a reference frame that is stationary for the purpose of the problem being described.

Figure 1.1 shows...

Pose:
The pose of a rigid body in a given reference frame is defined by its relative position and orientation with respect to the given reference frame and is represented by an element of $\mathbf{SE}(3)$. If a rigid body has orientation aligned with a reference frame {C} and position at the origin of {C}, then the pose

of the rigid body with respect to {B} and defined in {A} is:

$$ {}^{A}_{B}\mathbf{S}_{C} = \begin{bmatrix} {}^{A}_{B}\mathbf{R}_{C} & {}^{A}_{B}\mathbf{p}_{C} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{1.37} $$

Point:
A point $\mathbf{p} \in \mathbb{R}^3$ in the frame {A} is denoted ${}^{A}\mathbf{p}$ and is expressed as a 3-vector of the weights used to compose it from the basis vectors of {A}.

$$ {}^{A}\mathbf{p} = \begin{bmatrix} {}^{A}x \\ {}^{A}y \\ {}^{A}z \end{bmatrix} \tag{1.38} $$

Homogeneous coordinates:
To be acted on by an element of $\mathbf{SE}(3)$, a point must be expressed in homogeneous coordinates:

$$ {}^{A}\mathbf{p}' = \begin{bmatrix} {}^{A}\mathbf{p} \\ 1 \end{bmatrix} \tag{1.39} $$

Defining a point in terms of another reference frame:
Consider a point in $\mathbb{R}^3$ defined as the position of the frame {B} with respect to the frame {A}, defined in terms of the frame {B}. To redefine the point in terms of {A}, the left action of ${}^{A}_{A}\mathbf{S}_{B} \in \mathbf{SE}(3)$ is used:

$$ {}^{A}\mathbf{p}' = {}^{A}_{A}\mathbf{S}_{B}\ {}^{B}\mathbf{p}' \tag{1.40} $$

Concatenating poses:
Multiply relative poses.

$$ {}^{A}_{A}\mathbf{X}_{C} = {}^{A}_{A}\mathbf{X}_{B}\ {}^{B}_{B}\mathbf{X}_{C} \tag{1.41} $$

Defining a pose in terms of another reference frame:
To define a pose transformation matrix in terms of a different reference frame, a matrix conjugation is used:

$$ {}^{B}_{C}\mathbf{X}_{D} = ({}^{B}_{B}\mathbf{X}_{A})\ {}^{A}_{C}\mathbf{X}_{D}\ ({}^{B}_{B}\mathbf{X}_{A})^{-1} \tag{1.42} $$

Inverse:
Taking the inverse of a pose transformation matrix has the effect of reversing the transformation, but does not alter the frame that the transformation is defined in terms of.

$$ ({}^{A}_{B}\mathbf{X}_{C})^{-1} = {}^{A}_{C}\mathbf{X}_{B} \tag{1.43} $$

1.2.1.5 Rigid Body State Representation

The state of a rigid body moving through 3D space can be represented by its linear and angular position, velocity and acceleration. Higher derivatives could be taken but will be ignored for simplicity. The inertial frame is denoted {F} and a frame {A} is fixed to the pose of the moving body.

The pose of the body with respect to the inertial frame at time t, defined in the inertial frame is represented by the screw matrix ${}^{F}_{F}\mathbf{S}_{A}(t) \in \mathbf{SE}(3)$,

$$ {}^{F}_{F}\mathbf{S}_{A}(t) = \begin{bmatrix} {}^{F}_{F}\mathbf{R}_{A}(t) & {}^{F}_{F}\mathbf{p}_{A}(t) \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{1.44} $$

where ${}^{F}_{F}\mathbf{R}_{A}(t) \in \mathbf{SO}(3)$ is a rotation matrix, and the position ${}^{F}_{F}\mathbf{p}_{A}(t) \in \mathbb{R}^3$.

The linear and angular velocity of the body at time t with respect to the inertial frame, defined in the body-fixed frame, is represented by the twist matrix ${}^{A}_{F}\mathbf{T}_{A}(t) \in \mathfrak{se}(3)$,

$$ {}^{A}_{F}\mathbf{T}_{A}(t) = \begin{bmatrix} [{}^{A}_{F}\omega_{A}(t)]_{\times} & {}^{A}_{F}\mathbf{v}_{A}(t) \\ \mathbf{0}_{1\times 3} & 0 \end{bmatrix} \tag{1.45} $$

where ${}^{A}_{F}\omega_{A}(t) \in \mathfrak{so}(3)$ is an angular velocity in the scaled-axis representation, and the linear velocity is ${}^{A}_{F}\mathbf{v}_{A}(t) \in T\mathbb{R}^3$.

The linear and angular acceleration of the body at time t with respect to the inertial frame, defined in the body-fixed frame, is represented by the wrench matrix ${}^{A}_{F}\mathbf{W}_{A}(t) \in T\mathfrak{se}(3)$,

$$ {}^{A}_{F}\mathbf{W}_{A}(t) = \begin{bmatrix} [{}^{A}_{F}\alpha_{A}(t)]_{\times} & {}^{A}_{F}\mathbf{a}_{A}(t) \\ \mathbf{0}_{1\times 3} & 0 \end{bmatrix} \tag{1.46} $$

where ${}^{A}_{F}\alpha_{A}(t) \in T\mathfrak{so}(3)$ is an angular acceleration in the scaled-axis representation, and the linear acceleration is ${}^{A}_{F}\mathbf{a}_{A}(t) \in T^2\mathbb{R}^3$.

*From now on, will not show frames in notation. This is how S,T,W will be defined unless explicitly stated otherwise.

1.2.1.6 Rigid Body Kinematics

The dynamics of the screw, twist and wrench matrices as they are defined in 1.2.1.5 is governed by the following ODEs,

$$ \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{S}(t) = \mathbf{S}(t)\mathbf{T}(t) \tag{1.47} $$

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{T}(t) = \mathbf{W}(t) \tag{1.48}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{W}(t) = \mathbf{f}(t) \tag{1.49}$$

where the function $\mathbf{f}(t)$ is known.

1.2.1.7 Scanning Laser Rangefinder Dynamic Model

A scanning laser rangefinder fixed to a moving rigid body. State is same as moving rigid body defined above (S,T,W)
+
Unit vector defined in the body fixed frame - ${}^{A}\mathbf{n}(t) \in T\mathbb{R}^3$.
+
Range $r(t) \in \mathbb{R}^{0+}$, defining range from ${}^{F}_{F}\mathbf{p}_{A}(t)$ to nearest object in environment in direction ${}^{F}\mathbf{n}(t) = {}^{F}_{F}\mathbf{R}_{A}(t)\ {}^{A}\mathbf{n}(t)$

Scan direction in sensor frame is vector rotating at constant speed about z-axis, with unit size inside sensor's field-of-view and zero size outside it. Measurements returned at regular, discrete times - when t is an integer multiple of $\delta\tau$:

$${}^{A}\mathbf{n}(t) = \begin{cases} \begin{bmatrix} \cos(-\theta + 2\pi t') \\ -\sin(-\theta + 2\pi t') \\ 0 \end{bmatrix} & \text{if } t' \leq \theta/pi,\ t' = k\delta\tau \text{ where } k \in \mathbb{N} \\ \mathbf{0} & \text{if } t' > \theta/pi,\ t' \neq k\delta\tau \text{ where } k \in \mathbb{N} \end{cases} \tag{1.50}$$

where

$$t' = \mod(t, 1/d\theta)\, d\theta \tag{1.51}$$

*θ_0 is start of FOV, $t' = X$ at end of FOV

1.2.2 Symmetry Preserving Observers

1.2.2.1 definitions?

1.2.2.2 construction, ie moving frame method etc

1.2.3 Infinite Dimensional Observers

1.2.4 Discretisation Methods?

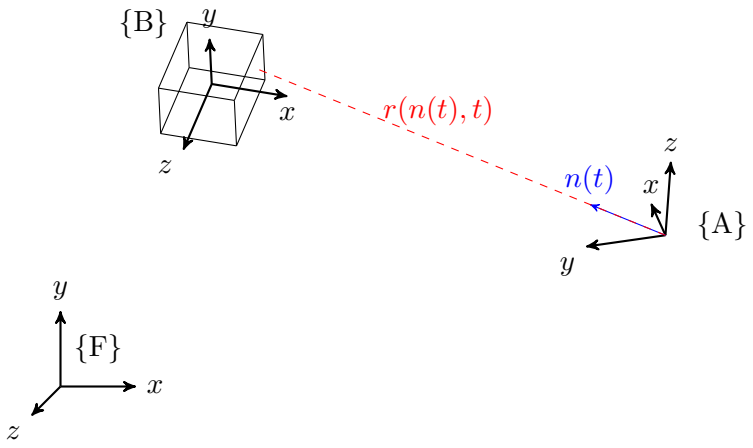


Figure 1.2: caption

1.3 Problem Statement

context etc here...

estimation problem - cube pose & size:
This report will solve a particular estimation problem before generalising the results to a wider class of systems. A situation in which an infinite dimensional observer would be useful is in the estimation of the pose of an object of size moving in an environment of unknown state. For example, consider an autonomous robot deployed in an agricultural survey, which must determine the position and size of a certain crop.

The problem to be investigated is shown in Figure 1.2. A 2D scanning range sensor moves through an environment consisting of a target object of known shape, in this case a rigid cube, and an unknown background which may consist of one or more rigid bodies. The state of the sensor is known, but the states of the cube and background environment are unknown. The goal is to use the state of the sensor and the range measurements it provides to estimate the state of the cube.

The frames used to describe the motion of the rigid bodies in this problem are:

- $\{F\}$ - the inertial (fixed) frame. For the purposes of this problem, the inertial frame is a frame whose motion is negligible. For the practical experiment this frame will be fixed to the ground.
- $\{A\}$ - the frame fixed to the sensor. The origin of this frame is the centre of rotation of the sensor's scan direction. The axes of $\{A\}$ are fixed to the sensor according to figure **TODO: FIGURE**. The

transformation from $\{F\}$ to $\{A\}$ at time t is defined by the screw matrix of the sensor $\mathbf{S}_s(t)$.

- $\{B\}$ - the frame fixed to the cube. The origin of $\{B\}$ coincides with the centre of the cube and is aligned so that each axis intersects with the centre of a face of the cube. The transformation from $\{F\}$ to $\{B\}$ at time t is defined by the screw matrix of the cube $\mathbf{S}_c(t)$.

The sensor provides measurements of the range r to the nearest object from the sensor (either the cube or the background) in the direction $\mathbf{n}(t)$. The state of the sensor $\mathbf{X}_s(t)$ is defined as:

$$\mathbf{X}_s(t) = \{{}^F_F\mathbf{S}_A(t), {}^A_F\mathbf{T}_A(t), {}^A_F\mathbf{W}_A(t), {}^A\mathbf{n}(t)\} \tag{1.52}$$

The screw matrix represents the transformation from $\{A\}$ to $\{F\}$, defined in $\{F\}$. The twist and wrench matrices, as well as the scan direction $\mathbf{n}(t)$ are defined in terms of$\{A\}$. For simplicity, this will be denoted

$$\mathbf{X}_s(t) = \{\mathbf{S}_s(t), \mathbf{T}_s(t), \mathbf{W}_s(t), {}^A\mathbf{n}(t)\} \tag{1.53}$$

The direction of measurement ${}^A\mathbf{n}(t)$ varies as a rotation about the z-axis of $\{A\}$. This 2D scanning motion depends on the model of the sensor used and is described in more detail in ADD REFERENCE. For simplification, the motion of the sensor itself with respect to $\{F\}$ will be limited to rotation about the y-axis of $\{F\}$.

TODO: DIAGRAM SHOWING RESTRICTED MOTION

The state of cube $\mathbf{X}_c(t)$ is defined as

$$\mathbf{X}_c(t) = \{{}^F_F\mathbf{S}_B(t), {}^B_F\mathbf{T}_B(t), {}^B_F\mathbf{W}_B(t), s\} \tag{1.54}$$

For simplicity, this will be denoted

$$\mathbf{X}_c(t) = \{\mathbf{S}_c(t), \mathbf{T}_c(t), \mathbf{W}_c(t), s\} \tag{1.55}$$

The range measurements do not indicate whether the object detected is the cube or a background object. Though the pose of the cube and environment remain unknown, for simplification, it is assumed that either:

- the cube is within a distance d from the sensor and background objects are at least a distance d away
- these target and background objects do not touch or overlap and their surfaces are continuous functions on $\mathbb{R}^3$

For simulated data, only the first assumption is necessary. For experimental data sets the environment is more complex so the second assumption is required to identify range measurements corresponding to the cube.

The aim is to design an observer which estimates the state of the cube from the pose of the sensor, as well as $\mathbf{n}$ and r. ie observer function f such that:

$$\hat{\mathbf{X}}_c = f(\mathbf{X}_s, r(t) - \hat{r}(t)) \tag{1.56}$$

*convergence - how to quantify performance?

deliverables:

Chapter 2

Simulation

2.1 Implementation

A simulation toolbox was implemented in Matlab to model scanning laser range-finder measurements and test observer schemes. The main components of the simulation are:

- rigid body trajectory computation;
- solid object modelling;
- range measurement simulation;
- noise modelling;
- observer implementation.

Algorithm 1 provides a high level description of the simulation.
Load settings - saved data can be loaded here
Initialise sensor, computes pose of sensor, scan directions over time + creates sensor class instance with sensor settings for noise generation
Initialise environment, creates environment class instance - points in body frame, triangles, computes position of points over time
Initialise observer - creates observer class instance
For each time step, get state of sensor and environment & compute range - this is done with a parallel for loop
Add noise to ranges
Observer simulation: each time step, update estimate based on estimated state (kinematics - numerical integration), predict range measurement, check if observing object, if yes update estimate based on measured and predicted range measurement.

Algorithm 1: Range measurement and state observer simulation

simulationLength - no. steps in simulation
sensorState - pose and scan direction
environmentState - points for cube and background, triangles
stateEstimate - estimate of pose and size of cube
observingObject - true/false, current range measurement of cube
rangesTrue - geometric range
incidenceAngles - angle of incidence for each measurement
triangleIndexes - index of triangle measured
rangesMeasured - ground truth + noise
rangesPredicted - expected range from state estimate
scanAngles - scan angle in sensor frame
$fieldOfView$ - set of scan angles corresponding to range measurement
begin
    $settings =$`loadsettings`
    $sensorState \leftarrow$ `initialisesensor`$(settings)$
    $environmentState \leftarrow$ `initialiseenvironment`$(settings)$
    `initialiseobserver`
    for $ii \leftarrow 1$ **to** $simulationLength$ **do**
        if $scanAngles[ii] \in fieldOfView$ **then**
            $[rangesTrue[ii], incidenceAngles[ii], triangleIndexes[ii]] \leftarrow$ `computerange`$(sensorState[ii], environmentState[ii])$
        end
    end
    $rangesMeasured =$ `addnoise`$(rangesTrue, incidenceAngles, triangleIndexes, settings)$
    for $ii \leftarrow 1$ **to** $simulationLength$ **do**
        $stateEstimate[ii+1] \leftarrow$ `estimatestate`$(stateEstimate[ii])$
        if $scanAngles[ii] \in fieldOfView$ **then**
            $rangesPredicted[ii] \leftarrow$ `computerange`$(sensorState[ii], stateEstimate[ii])$
            $observingObject \leftarrow$ `identifyobject`$(observingObject, rangesMeasured)$
            if $observingObject$ **then**
                $stateEstimate[ii+1] \leftarrow$ `updatestate`$(stateEstimate[ii+1])$
            end
        end
    end
end

2.1.1 Rigid Body Motion

To simulate range measurements the pose of the sensor and the objects comprising the environment must be computed at each time step. The computations required to do so can be reduced by taking into account the kinds of motion that must be simulated.

The observer actually computes the *relative* position between the sensor and cube and simply uses knowledge of the sensor pose to determine the pose of the cube in the inertial frame. There is no need to simulate complex sensor motions because the motion of the cube can be adjusted to achieve the same result. The only requirement of the sensor motion is that a large field of view is acquired so that the entire target object can be viewed. The scanning behaviour of the sensor is to rotate back and forth about the z-axis of the body fixed frame $\{A\}$. To provide a rectangular field of view, the motion of the sensor is therefore limited to constant velocity rotation about y-axis of inertial frame $\{F\}$.

The environment is modelled with two rigid bodies: a cube to be observed as the target object, and a stationary rectangular prism enclosing the sensor and cube acting as the background. The various cube motions that will be simulated to test the observer's performance can be classed in terms of the wrench matrix of the cube as either

1. $\mathbf{W}_c = \mathbf{0}$
2. $\mathbf{W}_c \neq \mathbf{0}$

For case 1. the wrench and screw are constant so only the initial value is required. It is more efficient to represent the pose of a rigid body with just position and orientation in this case. The pose can be quickly computed by interpolating between an initial and final pose. For case 2. the screw, twist and wrench must be integrated numerically.

2.1.1.1 Interpolation

To compute a trajectory of k poses beginning at $\{\mathbf{p}_i, \mathbf{q}_i\}$ and ending at $\{\mathbf{p}_f, \mathbf{q}_f\}$:

Poses for each time $\mathbf{t} = \begin{bmatrix} t_1 & t_2 & t_3 & \dots & t_k \end{bmatrix}$

Linear interpolation for position $\mathbf{P} = \begin{bmatrix} \mathbf{p}_1 & \mathbf{p}_2 & \mathbf{p}_3 & \dots & \mathbf{p}_k \end{bmatrix}$:

$$\mathbf{P} = \mathbf{p}_{1[1\times k]} + (\mathbf{p}_k - \mathbf{p}_1)\frac{\mathbf{t} - \mathbf{t}_{1[1\times k]}}{t_k - t_1} \tag{2.1}$$

OR?

$$\mathbf{P} = \mathbf{P}_{1[1\times k]} + (\mathbf{p}_k - \mathbf{p}_1)\frac{\mathbf{t} - \mathbf{t}_{1[1\times k]}}{t_k - t_1} \tag{2.2}$$

Spherical linear interpolation for orientation quaternion $\mathbf{q} = \begin{bmatrix}\mathbf{q}_1 & \mathbf{q}_2 & \mathbf{q}_3 & \dots & \mathbf{q}_k\end{bmatrix}$:

$$\mathbf{Q} = \frac{\mathbf{q}_1 \sin((\mathbf{1}_{[1\times k]} - \mathbf{t})\theta) + \mathbf{q}_k \sin(\mathbf{t}\theta)}{\sin(\theta)} \tag{2.3}$$

where

$$\theta = \cos^{-1}(\mathbf{q}_1 \cdot \mathbf{q}_k) \tag{2.4}$$

*MAKE THESE CLEARER

SCANNING:
Require multiple views of target object,reverse trajectory, concatenate and replicate
concatenate:

$$\mathbf{P}_{loop} = \begin{bmatrix}\mathbf{p}_1 & \mathbf{p}_2 & \mathbf{p}_3 & \dots & \mathbf{p}_k & \mathbf{p}_k & \mathbf{p}_{k-1} & \mathbf{p}_{k-2} & \dots & \mathbf{p}_1\end{bmatrix} \tag{2.5}$$

$$\mathbf{Q}_{loop} = \begin{bmatrix}\mathbf{q}_1 & \mathbf{q}_2 & \mathbf{q}_3 & \dots & \mathbf{q}_k & \mathbf{q}_k & \mathbf{q}_{k-1} & \mathbf{q}_{k-2} & \dots & \mathbf{q}_1\end{bmatrix} \tag{2.6}$$

replicate:

$$\mathbf{P} = \mathbf{P}_{loop_{[1\times k]}} \tag{2.7}$$

$$\mathbf{Q} = \mathbf{Q}_{loop_{[1\times k]}} \tag{2.8}$$

2.1.1.2 Numerical Integration

The time evolution of the screw, twist and wrench is computed iteratively from initial conditions by numerically integrating the ODEs in section 1.2.1.6. For a rigid body with an associated reference frame $\{X\}$:

$$\mathbf{S}_X(t + \delta t) = \mathbf{S}_X(t)\exp(\delta t \mathbf{T}_X(t)) \tag{2.9}$$

$$\mathbf{T}_X(t + \delta t) = \mathbf{T}_X(t) + \delta t \mathbf{W}_X(t) \tag{2.10}$$

Assuming constant acceleration

$$\mathbf{W}_X(t + \delta t) = \mathbf{W}_X(t) \tag{2.11}$$

*will add runge-kutta method to simulation. allow choice of numerical method in config.
*add equations for RK4 *not essential - ground truth is ground truth, experimental data won't be nearly as smooth anyway

2.1.2 Sensor modelling

The state of the sensor $\mathbf{X}_s(t)$ consists of terms corresponding to its motion and scanning operation.

motion:
Since motion restricted, state of sensor actually implemented as

$$\mathbf{X}_s(t) = \{\mathbf{p}_s(t), \mathbf{q}_s(t), {}^A\mathbf{n}(t)\} \tag{2.12}$$

Stationary position:

$$\mathbf{p}_1 = \mathbf{p}_2 = \mathbf{p}_3 = \cdots = \mathbf{p}_k = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.13}$$

Rotating from $-\phi$ to ϕ about y-axis of inertial frame ie interpolate between $\mathbf{q}_1$ and $\mathbf{q}_k$ with equation 2.3.

$$\mathbf{q}_1 = \begin{bmatrix} \cos(-\phi/2) \\ \sin(-\phi/2) \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \cos(\phi/2) \\ 0 \\ -\sin(\phi/2) \\ 0 \end{bmatrix} \tag{2.14}$$

$$\mathbf{q}_k = \begin{bmatrix} \cos(\phi/2) \\ \sin(\phi/2) \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \cos(\phi/2) \\ 0 \\ \sin(\phi/2) \\ 0 \end{bmatrix} \tag{2.15}$$

scanning:
*THIS SECTION IS NOT CLEAR AT ALL

The scanning behaviour of the sensor is determined by the vector ${}^A\mathbf{n}(t)$. To simulate a 2D scanning sensor the following parameters are used:

- field of view ($-\theta$ to θ, about -z-axis of sensor frame ie anticlockwise about z-axis). in practice, this is represented by start angle, scan direction and field of view angle
- angular resolution $d\theta$ (no. steps in 1 rev - 1024, $d\theta = 2\pi/1024$)
- time per revolution (1/24s = 24Hz). @nsteps/rev, this gives each time step as dtau = dt/nsteps

From these parameters, create vector $\mathbf{n}(t)$. At each time, n is either a unit vector in direction of scan, in body fixed frame, or returns no value for when sensor is not returning a measurement (outside FOV)

scan direction: ${}^{A}\mathbf{n}(t)$

$$ {}^{A}\mathbf{n}(t) = \begin{cases} \begin{bmatrix} \cos(-\theta + 2\pi t') \\ -\sin(-\theta + 2\pi t') \\ 0 \end{bmatrix} & \text{if } t' \leq \theta/pi,\ t' = k\delta\tau \text{ where } k \in \mathbb{N} \\ \mathbf{0} & \text{if } t' > \theta/pi,\ t' \neq k\delta\tau \text{ where } k \in \mathbb{N} \end{cases} \tag{2.16} $$

where

$$ t' = \mod(t, 1/d\theta)\, d\theta \tag{2.17} $$

*θ_0 is start of FOV, $t' = X$ at end of FOV

Scan direction in inertial frame required for measurement simulation:

$$ {}^{F}\mathbf{n}'(t) = \mathbf{X}_s(t)\, {}^{A}\mathbf{n}'(t) \tag{2.18} $$

2.1.3 Environment Modelling

motion:
Pose of object is actually pose of centre of mass of object. Each object modelled with S. from initial conditions, numerically integrate S,T,W. If constant velocity motion, faster to use waypoints like sensor. From pose of object at each time, compute pose of all points that make up object.

rigid objects:
Environment composed of rectangular prisms. These objects are modelled as an ordered set of 8 points in the inertial reference frame, and 12 triangles. Each triangle is a set of 3 integers, indicating the index of the three points that make up its vertexes. The position (in the inertial frame) of each of these points at each time is determined using the screw matrix and position in body frame (side lengths of the object & default cube points).

points in body frame: initialPoints = s*0.5*[];

$$ Pts = \frac{1}{2}s \begin{bmatrix} -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \end{bmatrix} \tag{2.19} $$

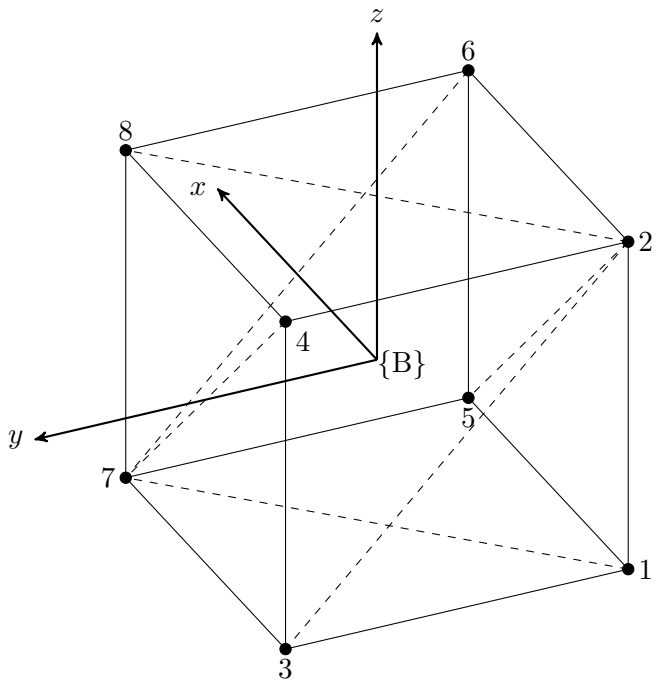


Figure 2.1: caption

triangles:

$$Tri = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 3 \\ 4 & 3 & 7 \\ 4 & 8 & 7 \\ 5 & 6 & 7 \\ 8 & 6 & 7 \\ 2 & 6 & 5 \\ 2 & 1 & 5 \\ 2 & 6 & 8 \\ 2 & 4 & 8 \\ 1 & 5 & 7 \\ 1 & 3 & 7 \end{bmatrix} \tag{2.20}$$

See figure 1.2

2.1.4 Measurement Modelling

2.1.4.1 Range Computation

Given the screw matrix (in the inertial frame) and scan direction (in the body fixed frame) of the sensor, the position and scan direction in the inertial frame are determined. The distance to the nearest environment object from this point, along the scan direction is determined with the Möller-Trumbore ray-triangle intersection algorithm.

Algorithm 2: Möller-Trumbore ray-triangle intersection algorithm

input : o - ray origin
d - ray direction vector
Pts - points
Tri - triangle matrix
output: *intersect* - True/False - whether object measured
range - distance to object in m
angle - incidence angle in rad
closest - index of triangle hit

begin
  `/* initialise outputs */`
  $intersect \longleftarrow 0$
  $range \longleftarrow NaN$
  $angle \longleftarrow NaN$
  $closest \longleftarrow NaN$
  `/* initialise other variables??? */`
  `/* triangle vertexes and edges */`
  $\mathbf{v}_1 \longleftarrow Pts[Tri[:,1]]$
  $\mathbf{v}_2 \longleftarrow Pts[Tri[:,2]]$
  $\mathbf{v}_3 \longleftarrow Pts[Tri[:,3]]$
  $e1 \longleftarrow e2 - e1$
  $e2 \longleftarrow e3 - e1$
  `/* determinant */`
  $T \longleftarrow$
  `/* barycentric coordinates */`
  `/* intersection vector */`
  if *any intersection* **then**
    min range, angle, triangle ID
  end
end

2.1.4.2 Object Surface

random walk

2.1.4.3 Sensor Noise

Noise model depends on sensor used.

$$\hat{r}(t) = f_s(r(t), \theta(t), \phi(k)) \tag{2.21}$$

where $\theta(t)$ is incidence angle of measurement, ϕ is surface properties of object k that was measured, f is some function for noise model of particular sensor.

For Hokuyo UBG-04LX-F01 used, noise model was measured experimentally:

$$f_{UBG} = \tag{2.22}$$

2.1.5 Observer implementation

2.1.5.1 Estimate: internal model

The state of the cube is estimated at each time $t \in \mathbf{t}$ using the numerical integration method described in section 2.1.1.2.

2.1.5.2 Identifying object/background

The variable *observingObject* indicates whether the range measurement is of the target object or the background. It is assumed that initially the sensor will be observing background, so $observingObject_0 = FALSE$

Range assumption: Assuming that the object is within X, background outside PSEUDOCODE if range $\leq$ X observingObject = 1 else observingObject = 0 end

Continuity assumption: if $abs(range_{i+1} - range_i) > X$ observingObject = mod(observingObject+1,2) end

2.1.5.3 Update

Input ranges:
use time ii, ii-1, ii-nSteps, ii-1-nSteps

Algorithm 3: Target/background object separation

input : *rangeAssumption* - true/false
continuityAssumption - true/false
rangeLimit
continuityLimit
observingObject - true/false
$range_{i+1}$ - distance to object in m at $t = i + 1$
$range_i$ - distance to object in m at $t = i$

output: *observingObject* - true/false

```
1  begin
2      if continuityAssumption then
3          if |range_{i+1} − range_i| > continuityLimit then
4              observingObject =
                 mod (observingObject + 1, 2)
5          end
6      end
7      if rangeAssumption then
8          if range_{i+1} > rangeLimit then
9              observingObject = 0
10         end
11     end
12 end
```

Orientation update:
at least 3 points, find planes and normal, rotate about cross of normals
→ gives rotation axis
EQUATIONS
DIAGRAM
figure 2.2

Position update:
find mean of intersection points of predicted, measured, vector subtraction
→ translation vector
EQUATIONS
DIAGRAM
figure 2.3

Size update:
if different pattern, delta s = dot(translation vector, scan direction)
if same pattern, delta s = vector(mean measured points - mean predicted points)
EQUATIONS
DIAGRAM
figure 2.4

Update scheme:
Can update S,T,W of state (or perhaps some combination)
Scale rotation axis, translation vector, delta s depending on if update is via S,T,W
EQUATIONS

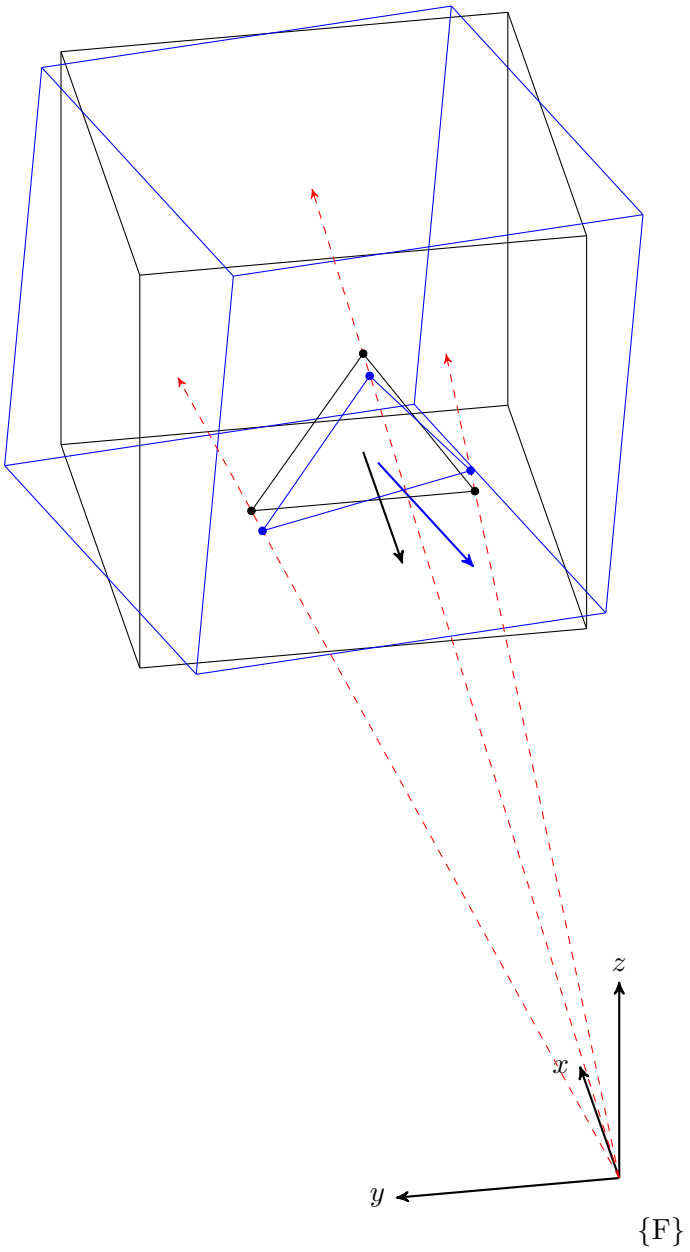
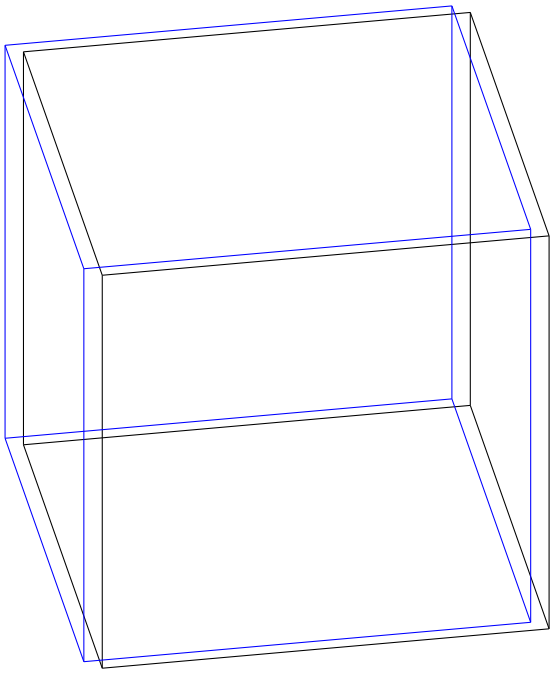


Figure 2.2: caption



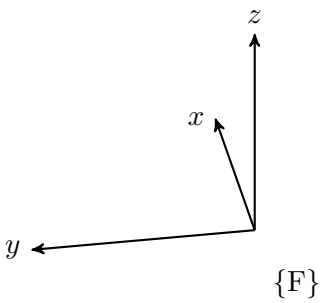


Figure 2.3: caption

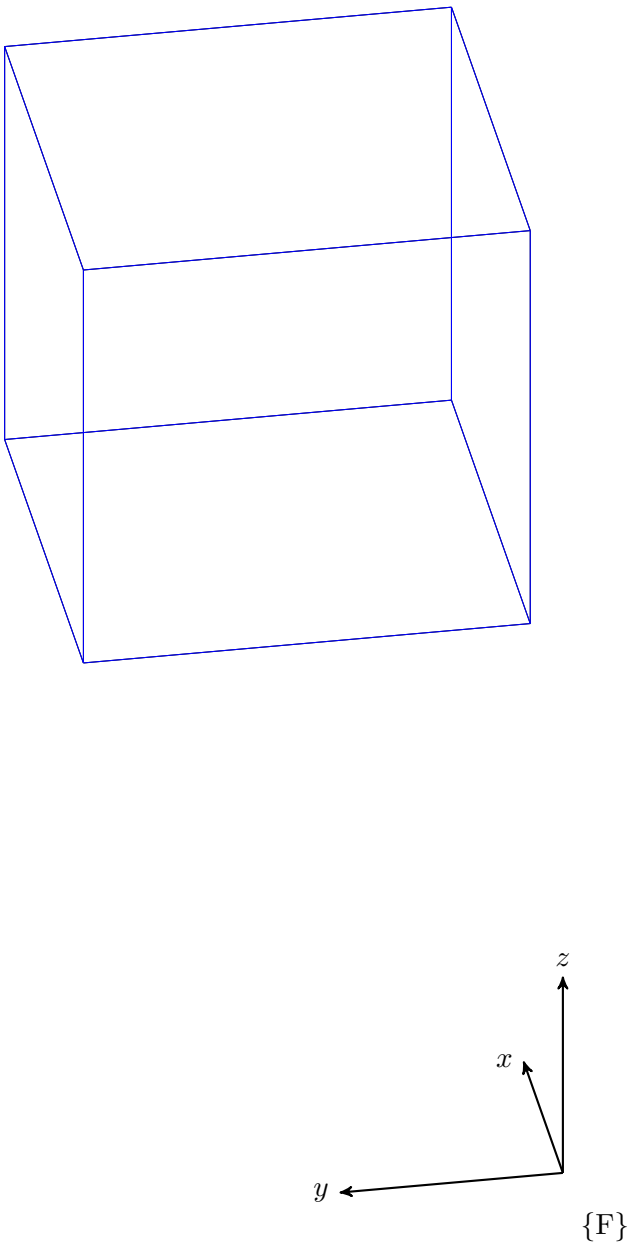


Figure 2.4: caption

2.2 Results

plots for different initial conditions & kinds of motion

Chapter 3

Experiment

intro: high level description of what was done, experimental setup
components: arm, cube, sensor
cube motion: different motions used
sensor calibration:
sensor motion:
experimental results: observer performance

Chapter 4

Conclusion

Bibliography

[1] C. Harkort and J. Deutscher, "Finite-dimensional observer-based control of linear distributed parameter systems using cascaded output observers," *International Journal of Control*, vol. 84, no. 1, pp. 107–122, 2011.

[2] L. Meirovitch and H. Baruh, "On the problem of observation spillover in self-adjoint distributed-parameter systems," *Journal of Optimization Theory and Applications*, vol. 39, no. 2, pp. 269–291, 1983.

[3] G. Haine, "Recovering the observable part of the initial data of an infinite-dimensional linear system with skew-adjoint generator," *Mathematics of Control, Signals, and Systems*, vol. 26, no. 3, pp. 435–462, 2014.

[4] J. W. Helton, "Systems with infinite-dimensional state space: the hilbert space approach," *Proceedings of the IEEE*, vol. 64, no. 1, pp. 145–160, 1976.

[5] K. Ramdani, M. Tucsnak, and G. Weiss, "Recovering the initial state of an infinite-dimensional system using observers," *Automatica*, vol. 46, no. 10, pp. 1616–1625, 2010.

[6] C.-Z. Xu, P. Ligarius, and J.-P. Gauthier, "An observer for infinite-dimensional dissipative bilinear systems," *Computers & Mathematics with Applications*, vol. 29, no. 7, pp. 13–21, 1995.

[7] H. Bounit and H. Hammouri, "Observers for infinite dimensional bilin ear, systems," *European journal of control*, vol. 3, no. 4, pp. 325–339, 1997.