Title

Montiel Abello

Chapter 1

Introduction

1.1 Notation

*INTERNAL - FOR KEEPING TRACK

NOTATION	DESCRIPTION
n_{steps}	no. time steps in simulation
n_{scans}	no. directions in rev.
$\hat{x}$	denotes prediction
$\tilde{x}$	denotes measurement
μ_x	mean of x
x'	gives x in homogeneous coords

1.2 Literature Review

1.2.1 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

1.2.2 symmetry preserving observers

1.2.2.1 early work

1.2.2.2 bonnabel et al

1.2.2.3 trumpf, mahony et al

1.2.2.4 juan's work - in detail

1.3 Theoretical Background

1.3.1 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 reference frame fixed to rigid body constitutes knowledge of the position 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.

1.3.1.1 Lie Groups

A Lie group $\mathbf{G}$ is a group whose elements form a differentiable manifold and group operation and inverse operation are differentiable. 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 $\mathbf{G} \subset \mathbf{GL}(n)$ is made up of group elements which are $n \times n$ matrices. This work will focus on matrix Lie groups because the form of the exponential map and Lie bracket functions given below only apply to such Lie groups. Generalised concepts for these functions exist, but a more detailed and relevant description can be provided by focusing on matrix 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 canonical 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 its domain $\mathbf{D} \subset \mathbf{G}$ to $\mathfrak{g}$

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

such that for a group element A,

$$\exp(\log(A)) = A \tag{1.4}$$

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

Infinitesimal generators
The *hat* operator $(\cdot)^\wedge$ can be used to map an n-vector to an $m \times m$ matrix representation, when $\mathbb{R}^{m\times m}$ is isomorphic to $\mathbb{R}^n$. (*NOTE - haven't actually used hat since it is used to denote prediction. wedge is a better counterpart to vee anyway. is this okay???)

$$\begin{aligned} (\cdot)^\wedge &: \mathbb{R}^n \to \mathbb{R}^{m\times m} \\ x &\mapsto x^\wedge = \sum_{i=1}^{n} x_i G_i \end{aligned} \tag{1.5}$$

where the set of elements G_i form a basis for $\mathbb{R}^{m\times m}$.

Conversely, the *vee* operator $(\cdot)^\vee$ maps matrices in $\mathbb{R}^{m\times m}$ to vectors in $\mathbb{R}^n$ such that $(x^\wedge)^\vee = x$

$$\begin{aligned} (\cdot)^\vee &: \mathbb{R}^{m\times m} \to \mathbb{R}^n \\ x^\wedge &\mapsto x \end{aligned} \tag{1.6}$$

For an n-dimensional matrix Lie group, the Lie algebra $\mathfrak{g}$ is a vector space isomorphic to $\mathbb{R}^n$. The hat operator $(\cdot)^\wedge$ maps vectors $x \in \mathbb{R}^n$ to elements of $\mathfrak{g}$. For a matrix Lie group $\mathbf{G}$ whose elements are $m \times m$ matrices, the elements of $\mathfrak{g}$ will also be $m \times m$ matrices.

$$\begin{aligned} (\cdot)^\wedge &: \mathbb{R}^n \to \mathfrak{g} \\ x &\mapsto x^\wedge = \sum_{i=1}^{n} x_i G_i \end{aligned} \tag{1.7}$$

The basis elements G_i are $m \times m$ 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 $A, B, C \in \mathfrak{g}$,

$$e^C = e^{A+B} = e^A e^B \tag{1.8}$$

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

$$C = A + B + \frac{1}{2}[A, B] + \frac{1}{12}[A - B, [A, B]] + \frac{1}{24}[B, [A, [A, B]]] + \dots \tag{1.9}$$

Actions
When a group action for a Lie group $\mathbf{G}$ 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

$$\Phi : \mathbf{G} \times M \mapsto M \tag{1.10}$$

where

- the identity element I acts as the identity on M

$$\Phi(I, m) = m \qquad \forall m \in M \tag{1.11}$$

- Group actions compose according to

$$\Phi(m, \Phi(n, o)) = \Phi(mn, o) \qquad \forall m, n, o \in M \tag{1.12}$$

1.3.1.2 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 later in this section, but the theory presented below only applies to matrix Lie groups which rely on the rotation matrix representation for group elements.

A rotation matrix $\mathbf{R}$ is a 3×3 matrix that performs a rotation operation when it acts on an element of $\mathbb{R}^3$. The properties of $\mathbf{R}$ are described in more detail below.

Lie algebra
The Lie algebra $\mathfrak{so}(3)$ is a vector space whose elements represent angular velocities. These elements can be represented with 3×3 skew-symmetric matrices $\boldsymbol{\omega}^\wedge$, where $\boldsymbol{\omega} \in \mathbb{R}^3$ is a 3-vector representing an angular velocity. For $\boldsymbol{\omega} = \begin{bmatrix}\omega_1 & \omega_2 & \omega_3\end{bmatrix}^T$, the skew symmetric representation is given by taking the hat representation of $\boldsymbol{\omega}$

$$\boldsymbol{\omega}^\wedge = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \tag{1.13}$$

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

$$\begin{aligned} \exp : \mathfrak{so}(3) &\to \mathbf{SO}(3) \\ \boldsymbol{\omega}^\wedge &\mapsto \exp(\boldsymbol{\omega}^\wedge) \end{aligned} \tag{1.14}$$

where the matrix $\exp(\boldsymbol{\omega}^\wedge) \in \mathbf{SO}(3)$ is a rotation matrix $\mathbf{R}$.

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

$$\begin{aligned} \log : \mathbf{SO}(3) &\to \mathfrak{so}(3) \\ \exp(\boldsymbol{\omega}^\wedge) &\mapsto \boldsymbol{\omega}^\wedge \end{aligned} \tag{1.15}$$

This means that for a rotation matrix $\mathbf{R}$, $\log(\mathbf{R}) \in \mathfrak{so}(3)$ and represents an angular velocity.

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

$$\begin{aligned} \Phi : \mathbf{SO}(3) \times \mathbb{R}^3 &\to \mathbb{R}^3 \\ (\mathbf{R}, \mathbf{p}) &\mapsto \mathbf{R}\mathbf{p} \end{aligned} \tag{1.16}$$

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

$$\mathbf{R}^{-1} = \mathbf{R}^T \tag{1.17}$$

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.

$$\boldsymbol{\theta} = \theta\mathbf{r} \tag{1.18}$$

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

$$\mathbf{R}_{\boldsymbol{\theta}} = \mathbf{I} + [\mathbf{r}]_\times \sin\theta + ([\mathbf{r}]_\times)^2(1 - \cos\theta) \tag{1.19}$$

Elements of $\mathfrak{so}(3)$ are typically represented with the hat representation $\boldsymbol{\omega}^\wedge$ of 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:

$$\mathbf{q} = \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} w \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} \cos(\theta/2) \\ \sin(\theta/2)\mathbf{r} \end{bmatrix} \tag{1.20}$$

In general, the quaternion inverse is given by

$$\mathbf{q}^{-1} = \frac{1}{w^2 + x^2 + y^2 + z^2} \begin{bmatrix} w \\ -x \\ -y \\ -z \end{bmatrix} \tag{1.21}$$

For unit magnitude rotation quaternions the inverse represents a rotation by $-\theta$ and is given by

$$\mathbf{q}^{-1} = \begin{bmatrix} \cos(\theta/2) \\ -\sin(\theta/2)\mathbf{r} \end{bmatrix} = \begin{bmatrix} w \\ -x \\ -y \\ -z \end{bmatrix} \tag{1.22}$$

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

$$\mathbf{q}_1\mathbf{q}_2 = \begin{bmatrix} w_1 \\ \mathbf{v}_1 \end{bmatrix} \begin{bmatrix} w_2 \\ \mathbf{v}_2 \end{bmatrix} = \begin{bmatrix} w_1 w_2 - \mathbf{v}_1 \cdot \mathbf{v}_2 \\ w_1\mathbf{v}_2 + w_2\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2 \end{bmatrix} \tag{1.23}$$

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:

$$\begin{bmatrix} 0 \\ \mathbf{p}' \end{bmatrix} = \mathbf{q} \begin{bmatrix} 0 \\ \mathbf{p} \end{bmatrix} \mathbf{q}^{-1} \tag{1.24}$$

1.3.1.3 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

$$\mathbf{X} = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{1.25}$$

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

$$\begin{bmatrix} \boldsymbol{\omega}^\wedge & \mathbf{v} \\ \mathbf{0}_{1\times 3} & 0 \end{bmatrix} \tag{1.26}$$

where $\boldsymbol{\omega} = \begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^\top \in \mathbf{so}(3)$, 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:

$$\begin{gathered} \exp : \mathfrak{se}(3) \to \mathbf{SE}(3) \\ \begin{bmatrix} \boldsymbol{\omega}^\wedge & \mathbf{v} \\ \mathbf{0}_{1\times 3} & 0 \end{bmatrix} \mapsto \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \end{gathered} \tag{1.27}$$

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 $\mathfrak{se}(3)$:

$$\begin{gathered} \log : \mathbf{SE}(3) \to \mathfrak{se}(3) \\ \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \mapsto \begin{bmatrix} \boldsymbol{\omega}^\wedge & \mathbf{v} \\ \mathbf{0}_{1\times 3} & 0 \end{bmatrix} \end{gathered} \tag{1.28}$$

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)

$$\mathbf{p}' = \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{1.29}$$

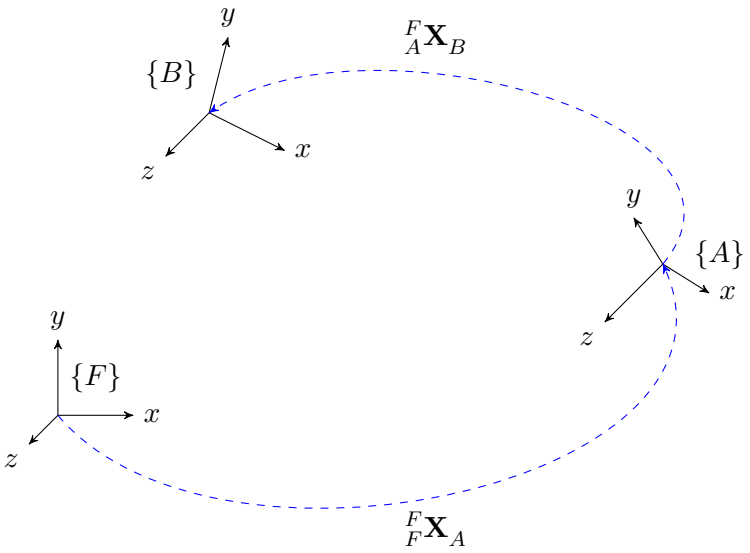


Figure 1.1: what frame should transformations be defined in?

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

$$\mathbf{p'}_1 = \mathbf{S}\mathbf{p'}_0 = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{p}_0 \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}\mathbf{p}_0 + \mathbf{p} \\ 1 \end{bmatrix} \tag{1.30}$$

1.3.1.4 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 $\mathbf{SE}(3)$.

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. The convention used will be to denote the inertial reference frame as $\{F\}$.

Consider the three reference frames shown in Figure 1.1, denoted $\{F\}$ (the inertial frame),$\{A\}$ and $\{B\}$. The notation ${}^F_A\mathbf{X}_B$ defines the transformation in $\mathbf{X}$ of the reference frame $\{B\}$ with respect to the frame $\{A\}$, defined in the frame $\{F\}$.

For example, ${}^F_A\mathbf{R}_B$ defines the rotation of $\{B\}$ with respect to $\{A\}$, defined in the inertial frame $\{F\}$.

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 $\{B\}$ and position at the origin of $\{B\}$, then the pose of the rigid body with respect to $\{A\}$ and defined in $\{F\}$ is:

$${}^F_A\mathbf{S}_B = \begin{bmatrix} {}^F_A\mathbf{R}_B & {}^F_A\mathbf{p}_B \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{1.31}$$

Point:
A point $\mathbf{p} \in \mathbb{R}^3$ in the frame $\{F\}$ is denoted ${}^F\mathbf{p}$ and is expressed as a 3-vector of the weights

used to compose it from the basis vectors of $\{F\}$.

$$ {}^{F}\mathbf{p} = \begin{bmatrix} {}^{F}x \\ {}^{F}y \\ {}^{F}z \end{bmatrix} \tag{1.32} $$

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

$$ {}^{F}\mathbf{p}' = \begin{bmatrix} {}^{F}\mathbf{p} \\ 1 \end{bmatrix} \tag{1.33} $$

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

$$ {}^{F}\mathbf{p}' = {}^{F}_{F}\mathbf{S}_{A}\ {}^{A}\mathbf{p}' \tag{1.34} $$

Concatenating poses:
Poses are concatenated by multiplying relative poses.

$$ {}^{F}_{F}\mathbf{X}_{B} = {}^{F}_{F}\mathbf{X}_{A}\ {}^{A}_{A}\mathbf{X}_{B} \tag{1.35} $$

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:

$$ {}^{A}_{B}\mathbf{X}_{C} = ({}^{A}_{A}\mathbf{X}_{F})\ {}^{F}_{B}\mathbf{X}_{C}\ ({}^{A}_{A}\mathbf{X}_{F})^{-1} \tag{1.36} $$

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.

$$ ({}^{F}_{A}\mathbf{X}_{B})^{-1} = {}^{F}_{B}\mathbf{X}_{A} \tag{1.37} $$

1.3.1.5 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 $\{F\}$ is represented by the **screw** matrix ${}^{F}_{F}\mathbf{S}_{A}(t) \in \mathbf{SE}(3)$,

$$ {}^{F}_{F}\mathbf{S}_{A}(t) = \begin{bmatrix} {}^{F}_{F}\mathbf{R}_{A}(t) & {}^{F}_{F}\mathbf{p}_{A}(t) \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{1.38} $$

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 ${}^{A}_{F}\mathbf{T}_{A}(t) \in \mathfrak{se}(3)$,

$$ {}^{A}_{F}\mathbf{T}_{A}(t) = \begin{bmatrix} {}^{A}_{F}\boldsymbol{\omega}^{\wedge}_{A}(t) & {}^{A}_{F}\mathbf{v}_{A}(t) \\ \mathbf{0}_{1\times 3} & 0 \end{bmatrix} \tag{1.39} $$

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

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 ${}^{A}_{F}\mathbf{W}_{A}(t) \in T\mathfrak{se}(3) \equiv \mathfrak{se}(3)$,

$$ {}^{A}_{F}\mathbf{W}_{A}(t) = \begin{bmatrix} {}^{A}_{F}\boldsymbol{\alpha}^{\wedge}_{A}(t) & {}^{A}_{F}\mathbf{a}_{A}(t) \\ \mathbf{0}_{1\times 3} & 0 \end{bmatrix} \tag{1.40} $$

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

From this point on, frames will dropped in the notation. For a body labelled x fixed to a frame $\{A\}$, ${}^{F}_{F}\mathbf{S}_{A}$, ${}^{A}_{F}\mathbf{T}_{A}$ and ${}^{A}_{F}\mathbf{W}_{A}$ will be denoted $\mathbf{S}_x$, $\mathbf{T}_x$ and $\mathbf{W}_x$.

1.3.1.6 Rigid Body Kinematics

The dynamics of the screw, twist and wrench matrices as they are defined in 1.3.1.5 is governed by the following ordinary differential equations (ODEs),

$$ \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{S}(t) = \mathbf{S}(t)\mathbf{T}(t) \tag{1.41} $$

$$ \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{T}(t) = \mathbf{W}(t) \tag{1.42} $$

$$ \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{W}(t) = \mathbf{f}(t) \tag{1.43} $$

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

1.3.1.7 Scanning Laser Rangefinder Dynamic Model

SHOULD THIS BE HERE? DESCRIBED IN MORE DETAIL IN SIMULATION CHAPTER

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 - ${}^{A}\mathbf{d}(t) \in T\mathbb{R}^3$.

+

Range $r(t) \in \mathbb{R}^{0+}$, defining range from ${}^{F}_{F}\mathbf{p}_{A}(t)$ to nearest object in environment in direction ${}^{F}\mathbf{d}(t) = {}^{F}_{F}\mathbf{R}_{A}(t)\ {}^{A}\mathbf{d}(t)$

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 $\delta\tau$:

$$ {}^{A}\mathbf{d}(t) = \begin{cases} \begin{bmatrix} \cos(-\theta + 2\pi t') \\ -\sin(-\theta + 2\pi t') \\ 0 \end{bmatrix} & \text{if } t' \leq \theta/\pi,\ t' = k\delta\tau \text{ where } k \in \mathbb{N} \\ \mathbf{0} & \text{if } t' > \theta/\pi,\ t' \neq k\delta\tau \text{ where } k \in \mathbb{N} \end{cases} \tag{1.44} $$

where

$$ t' = \mod(t, 1/d\theta)\, d\theta \tag{1.45} $$

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

1.3.2 Symmetry Preserving Observers

1.3.2.1 definitions?

1.3.2.2 construction, ie moving frame method etc

1.3.3 Infinite Dimensional Observers

1.3.4 Discretisation Methods?

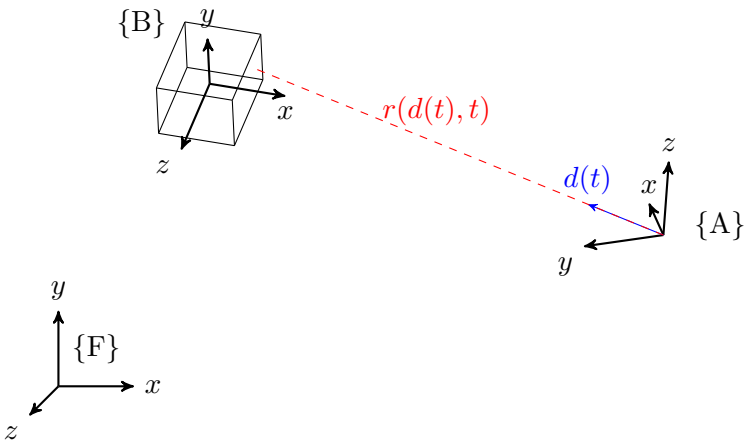


Figure 1.2: caption

1.4 Problem Statement

context etc here...

estimation problem - cube pose & size:

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. Using a geometric model for the general shape of the crop, an aerial vehicle that could routinely detect and characterise the position and size of specimens would be useful in monitoring growth and during harvesting.

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 2.1 in chapter 2. 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 $\mathbf{d}(t)$. The state of the sensor $\mathbf{X}_s(t)$ is defined as:

$$\mathbf{X}_s(t) = \{{}^F_F\mathbf{S}_A(t), {}^A_F\mathbf{T}_A(t), {}^A_F\mathbf{W}_A(t), {}^A\mathbf{d}(t)\} \tag{1.46}$$

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

this will be denoted

$$\mathbf{X}_s(t) = \{\mathbf{S}_s(t), \mathbf{T}_s(t), \mathbf{W}_s(t), {}^A\mathbf{d}(t)\} \tag{1.47}$$

The direction of measurement ${}^A\mathbf{d}(t)$ 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

$$\mathbf{X}_c(t) = \{{}^F_F\mathbf{S}_B(t), {}^B_F\mathbf{T}_B(t), {}^B_F\mathbf{W}_B(t), s\} \tag{1.48}$$

For simplicity, this will be denoted

$$\mathbf{X}_c(t) = \{\mathbf{S}_c(t), \mathbf{T}_c(t), \mathbf{W}_c(t), s\} \tag{1.49}$$

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 r_{max} from the sensor and background objects are at least a distance r_{max} 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:

$$\hat{\mathbf{X}}_c(k+1) = f(\mathbf{X}_s(t), \hat{\mathbf{X}}_c(k), r(t), \hat{r}(t)) \tag{1.50}$$

*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.

NOTE: PUT THIS BIT IN INTRODUCTION??? The notation employed here and throughout the rest of the report represents variables according to:

- *plain text*: single values;
- *bold lowercase*: vectors;
- *bold uppercase*: matrices

Additionally, an array formed by replicating a variable $\mathbf{a}$ in an $n \times m$ block array is denoted $\mathbf{a}_{n\times m}$.

In many cases, a variable such as the sensor position $\mathbf{p}_s(t)$ represents a set of elements, each corresponding to the value at a certain time t. These will be referred to by the form of the value at a single time. For example, a matrix where each column represents a different position vector is used to represent $\mathbf{p}_s(t)$, but it will be described as a vector as this is the form of a single position.

A high level description of the simulation is provided in Algorithm 1. First, a settings file is loaded. The most important settings determined here are the trajectories of the sensor and environment objects, the scanning behaviour of the sensor and the observer update function. Next, the sensor class instance is initialised with `initialisesensor`. This requires computation of the pose and scanning directions of the sensor over time. Similarly, initialisation of the environment through `initialiseenvironment` requires computation of the pose of each rigid body comprising it. The surfaces of the bodies are then represented with a set of points and corresponding triangles. The position of each point with respect to the inertial frame $\{F\}$ is computed at each time step. The settings file provides the initial conditions with which the observer is initialised in `initialiseobserver`. Beginning on line 6, the state of the sensor and

Algorithm 1: Scanning range-finder and state observer simulation

Data:
n_{steps} - no. steps in simulation
$\mathbf{X}_s$ - sensor pose and scan direction
$\mathbf{X}_e$ - cube and background pose, points and triangles
$\hat{\mathbf{X}}_c$ - estimate of the pose and size of the cube
c - (`true`/`false`) current range measurement is of cube or not
$\mathbf{r}$ - ground truth range
$\tilde{\mathbf{r}}$ - measured range
$\hat{\mathbf{r}}$ - predicted range from state estimate
$\boldsymbol{\alpha}$ - angle of incidence for each range measurement
$\mathbf{m}$ - index of triangle measured
$\boldsymbol{\theta}$ - scan angle in sensor frame
$\boldsymbol{\Theta}$ - set of scan angles that return range measurement

begin
  $settings \leftarrow$`loadsettings`
  $\mathbf{X}_s \leftarrow$ `initialisesensor`$(settings)$
  $\mathbf{X}_e \leftarrow$ `initialiseenvironment`$(settings)$
  `initialiseobserver`
  for $ii \leftarrow 1$ **to** n_{steps} **do**
    if $\boldsymbol{\theta}[ii] \in \boldsymbol{\Theta}$ **then**
      $[\mathbf{r}[ii], \boldsymbol{\alpha}[ii], \mathbf{m}[ii]] \leftarrow$ `computerange`$(\mathbf{X}_s[ii], \mathbf{X}_e[ii])$
    end
  end
  $\tilde{\mathbf{r}} =$ `addnoise`$(\mathbf{r}, \boldsymbol{\alpha}, \mathbf{m}, settings)$
  for $ii \leftarrow 1$ **to** n_{steps} **do**
    $\hat{\mathbf{X}}_c[ii+1] \leftarrow$ `estimatestate`$(\hat{\mathbf{X}}_c[ii])$
    if $\boldsymbol{\theta}[ii] \in \boldsymbol{\Theta}$ **then**
      $\hat{\mathbf{r}}[ii] \leftarrow$ `computerange`$(\mathbf{X}_s[ii], \hat{\mathbf{X}}_c[ii])$
      $c \leftarrow$ `identifyobject`$(c, \tilde{\mathbf{r}})$
      if c **then**
        $\hat{\mathbf{X}}_c[ii+1] \leftarrow$ `updatestate`$(\hat{\mathbf{X}}_c[ii+1], \mathbf{X}_s[ii], \hat{\mathbf{r}}, \tilde{\mathbf{r}})$
      end
    end
  end
end

environment are used to compute the ground truth range measurements $\mathbf{r}$ at each time step. The incidence angle between the scan direction and object, as well as the index of the triangle hit are also stored as they will be required for sensor noise modelling. This is performed with a parallel `for` loop to speed up computation. Line 7 ensures ranges are only computed when the current scan direction is within the sensor's field of view. In line 11, noise is simulated and added to the ground truth ranges to produce the measured ranges $\tilde{\mathbf{r}}$. The `for` loop beginning on line 12 begins the observer simulation. At each time step, `estimatestate` estimates the state of the cube $\hat{\mathbf{X}}_c$ from the previous state with the kinematics model in 1.3.1.6. From the sensor state $\mathbf{X}_s$ and the estimated state of the cube $\hat{\mathbf{X}}_c$, `computerange` is used to determined the predicted range measurement $\hat{\mathbf{r}}$. The variable c indicates whether the current range measurement corresponds to the cube or the background. On line 16 the measured ranges and previous value of c are used to determine whether the current measurement corresponds to the cube. If it does, the cube state estimate $\hat{\mathbf{X}}_c$ is updated using the previous state estimate, current sensor state, and the predicted and measured ranges.

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. This section details how the pose of the sensor and objects is represented and computed, which is used in the functions `initialisesensor`, `initialiseenvironment` and `estimatestate` referenced in Algorithm 1.

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 measurements of a large range of the environment are acquired to ensure 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

For the case of zero wrench, the pose of the body can be represented with a position vector $\mathbf{p}_i$ and an orientation quaternion $\mathbf{q}_i$. A trajectory of k poses at times $\mathbf{t} = \begin{bmatrix} t_1 & t_2 & t_3 & \dots & t_k \end{bmatrix}$, is computed by interpolating from an initial pose $\{\mathbf{p}_0, \mathbf{q}_0\}$ to a final pose $\{\mathbf{p}_k, \mathbf{q}_k\}$.

The array of position vectors $\mathbf{P} = \begin{bmatrix}\mathbf{p}_1 & \mathbf{p}_2 & \mathbf{p}_3 & \dots & \mathbf{p}_k\end{bmatrix}$ are computed with:

$$\mathbf{P} = (\mathbf{1}_{3\times k})\mathbf{p}_1 + (\mathbf{p}_k - \mathbf{p}_1)\frac{\mathbf{t} - t_1(\mathbf{1}_{1\times k})}{t_k - t_1} \tag{2.1}$$

Spherical linear interpolation is used to compute the array of orientation quaternions $\mathbf{Q} = \begin{bmatrix}\mathbf{q}_1 & \mathbf{q}_2 & \mathbf{q}_3 & \dots & \mathbf{q}_k\end{bmatrix}$ at each time step:

$$\mathbf{Q} = \frac{\mathbf{q}_1 \sin((\mathbf{1}_{[1\times k]} - \mathbf{t})\theta) + \mathbf{q}_k \sin(\mathbf{t}\theta)}{\sin(\theta)} \tag{2.2}$$

where

$$\theta = \cos^{-1}(\mathbf{q}_1 \cdot \mathbf{q}_k) \tag{2.3}$$

This interpolation method is used to compute the trajectory of the sensor. To acquire multiple views of the entire cube, the sensor must pan back and forth several times. This is achieved by first reversing the trajectory and concatenating with the original to produce the looped trajectories $\mathbf{P}_{loop}$ and $\mathbf{Q}_{loop}$:

$$\mathbf{P}_{loop} = \begin{bmatrix}\mathbf{p}_1 & \mathbf{p}_2 & \mathbf{p}_3 & \dots & \mathbf{p}_k & \mathbf{p}_k & \mathbf{p}_{k-1} & \mathbf{p}_{k-2} & \dots & \mathbf{p}_1\end{bmatrix} \tag{2.4}$$

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

This looped trajectory is repeated k times to produce multiple back and forth scans:

$$\mathbf{P} = (\mathbf{P}_{loop})_{1\times k} \tag{2.6}$$

$$\mathbf{Q} = (\mathbf{Q}_{loop})_{1\times k} \tag{2.7}$$

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 1.3.1.6. For a rigid body with an associated reference frame $\{X\}$, moving with constant acceleration:

$$\mathbf{S}_X(t + \delta t) = \mathbf{S}_X(t)\exp(\delta t \mathbf{T}_X(t)) \tag{2.8}$$

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

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

Though a higher order integration method, such as Runge-Kutta could be used to compute a trajectory that more accurately represents a constant acceleration, this is not strictly necessary. The observer performance is unlikely to be affected by how constant the acceleration is. Furthermore, it is likely that the experimentally collected data will have even larger variations in acceleration.

To simplify the code, the position vector and orientation quaternion are computed from the screw matrix. This allows the same functions to be used in either the interpolation or numerical integration cases. Multiplying the points that make up the rigid objects can also be done more compactly with quaternions.

2.1.2 Sensor modelling: `initialisesensor`

The motion model below is used to compute the pose of the sensor. The scanning model is used to compute the set of scanning directions. These actions are performed in the `initialisesensor` function in Algorithm 1

2.1.2.1 Motion

The state of the sensor $\mathbf{X}_s(t)$ consists of terms corresponding to its motion and scanning operation. Since the motion of the sensor is restricted to zero acceleration, the state of sensor can be computed with the interpolation method and represented with position, orientation and scanning direction.

$$\mathbf{X}_s(t) = \{\mathbf{p}_s(t), \mathbf{q}_s(t), {}^A\mathbf{d}(t)\} \tag{2.11}$$

Since it has a stationary position, the position of the sensor over time is fixed at the origin of the inertial frame $\{F\}$.

$$\mathbf{p}_1 = \mathbf{p}_2 = \mathbf{p}_3 = \cdots = \mathbf{p}_k = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.12}$$

The sensor rotates between $-\phi$ and ϕ about the y-axis of inertial frame $\{F\}$. Thus, its orientation is computed by interpolating between $\mathbf{q}_1$ and $\mathbf{q}_k$ with equation 2.2.

$$\mathbf{q}_1 = \begin{bmatrix} \cos(-\phi/2) \\ \sin(-\phi/2) \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \cos(\phi/2) \\ 0 \\ -\sin(\phi/2) \\ 0 \end{bmatrix} \tag{2.13}$$

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

2.1.2.2 Scanning

The scanning behaviour of the sensor depends on the particular model used. The *Hokuyo UBG 04-LX* scanning laser range-finder will be modelled as it was the sensor used to conduct experiments in this project. This sensor produces a 785nm laser beam, projected at a precise direction. It measures the characteristics of the reflected beam to determine the position to the nearest object in the direction of the beam. The direction of the laser beam is varied by reflecting it off a rotating mirror. The rotation means that the beam direction effectively rotates with a constant velocity in a single plane. A portion of the field of view of the laser beam is obscured, so measurements will not be returned in a certain region of each revolution.

The vector ${}^A\mathbf{d}(t)$ will be used to model this scanning behaviour. To accurately model this, the following parameters are used:

- field of view $\mathbf{\Theta}$: The vector ${}^A\mathbf{d}(t)$ rotates anti-clockwise about the z axis of the sensor frame $\{A\}$. Measurements are only taken when the scan angle about is between $-\theta$ and θ about the $-z$-axis of the sensor frame $\{A\}$. In practice, the field of view is implemented as the start angle $-\theta$, direction of rotation and angular range 2θ.

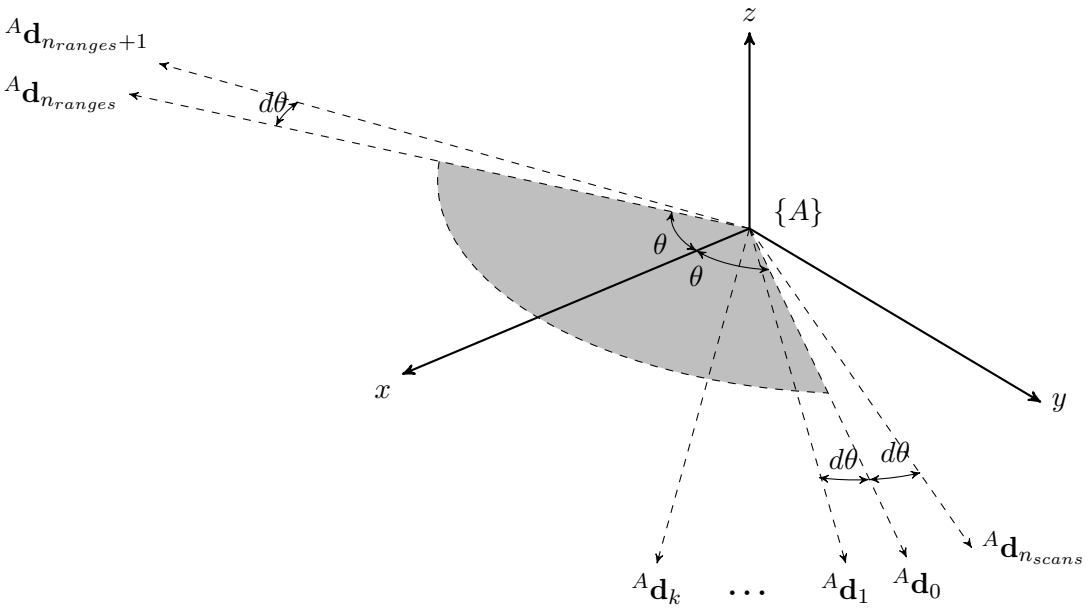


Figure 2.1: Scanning parameters

- number of scans n_{scans}: This represents the number of scan angles in a single revolution. Since measurements are limited by the field of view of the sensor, the actual number of measurements per second is $n_{ranges} = \frac{2\theta}{2\pi} n_{scans}$. The angular resolution is $d\theta = \dfrac{2\pi}{n_{scans}}$.
- revolutions per second Ω: This is measured in Hz and gives the length of each time step $d\tau = \dfrac{1}{n_{scans}\Omega}$
- n_{loops}: The number of back and forth repeats of the sensor trajectory.

From these parameters the scanning direction ${}^A\mathbf{d}(t)$ is created. At each time t, ${}^A\mathbf{d}(t)$ is either a unit vector indicating the direction of measurement in the sensor frame, or has $\mathbf{0}$ magnitude, corresponding to when ${}^A\mathbf{d}(t)$ is outside the field of view and the sensor is not returning a measurement.

$$ {}^A\mathbf{d}(t) = \begin{cases} \begin{bmatrix} \cos(-\theta + 2\pi t') \\ -\sin(-\theta + 2\pi t') \\ 0 \end{bmatrix} & \text{if } t' \leq \theta/\pi,\ t' = k\delta\tau \text{ where } k \in \mathbb{N} \\ \mathbf{0}_{3\times 1} & \text{if } t' > \theta/\pi,\ t' \neq k\delta\tau \text{ where } k \in \mathbb{N} \end{cases} \tag{2.15} $$

where

$$ t' = \mod(t, 1/d\theta)\, d\theta \tag{2.16} $$

Figure 2.1 shows the frame $\{A\}$ fixed to the sensor and the scan direction ${}^A\mathbf{d}(t)$. At time $t' = 0$, the first scan direction ${}^A\mathbf{d}_0$ has an angular displacement of $-\theta$ about the z-axis from the forward facing x-direction. After each time step $d\tau$, the scan direction rotates by $d\theta$ about the z-axis. There are n_{ranges} scan directions within the field of view of the sensor. The entire revolution is divided into n_{scans} scan directions.

To simulate range measurements, the scan direction is required in the inertial frame $\{F\}$. This is computed by multiplying with the screw matrix of the sensor.

$$ \begin{aligned} {}^F\mathbf{d}'(t) &= \mathbf{S}_s(t)\, {}^A\mathbf{d}'(t) \\ &= {}^F_F\mathbf{S}_A(t)\, {}^A\mathbf{d}'(t) \end{aligned} \tag{2.17} $$

2.1.3 Environment Modelling: `initialiseenvironment`

2.1.3.1 Motion

The pose of each object represents the pose of its centre of mass. As described in section 2.1.1 the pose is computed with interpolation for the case of zero wrench, and numerical integration in the case of non-zero wrench. The pose of the objects making up the environment is computed with the function `initialiseenvironment` in Algorithm 1. This function also creates the points and triangles that model the surface of the objects which is described below.

2.1.3.2 Rigid Objects

The environment is represented with two rectangular prisms; the cube and a larger rectangular prism enclosing both the sensor and cube, to represent the background. These objects are modelled as an ordered set of 8 points in the inertial reference frame and an ordered set of 12 triangles formed by these points. Each triangle is represented by a set of 3 integers, indicating the index of the three points that make up its vertices.

The cube points in body frame $\{B\}$ are represented with the matrix ${}^{B}\mathbf{P}$.

$$ {}^{B}\mathbf{P} = \frac{1}{2}s \begin{bmatrix} -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \end{bmatrix} \tag{2.18} $$

To represent these points in the inertial frame $\{F\}$, ${}^{F}\mathbf{P}$ is computed by rotating each point with the orientation quaternion of frame $\{B\}$ using equation 1.24 before adding the vector representing the translation of $\{B\}$ from $\{F\}$.

The triangles are represented with the matrix $\mathbf{T}$. Each triangle is represented by a row. The elements of these rows are the three vertices of the triangle and the index corresponds a point in ${}^{F}\mathbf{P}$.

$$ \mathbf{T} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 3 \\ 4 & 3 & 7 \\ 4 & 8 & 7 \\ 5 & 6 & 7 \\ 8 & 6 & 7 \\ 2 & 6 & 5 \\ 2 & 1 & 5 \\ 2 & 6 & 8 \\ 2 & 4 & 8 \\ 1 & 5 & 7 \\ 1 & 3 & 7 \end{bmatrix} \tag{2.19} $$

The points and triangles are shown in Figure 2.2.

2.1.4 Measurement Modelling

2.1.4.1 Range Computation: `computerange`

This section describes the implementation used in the `computerange` function in Algorithm 1.

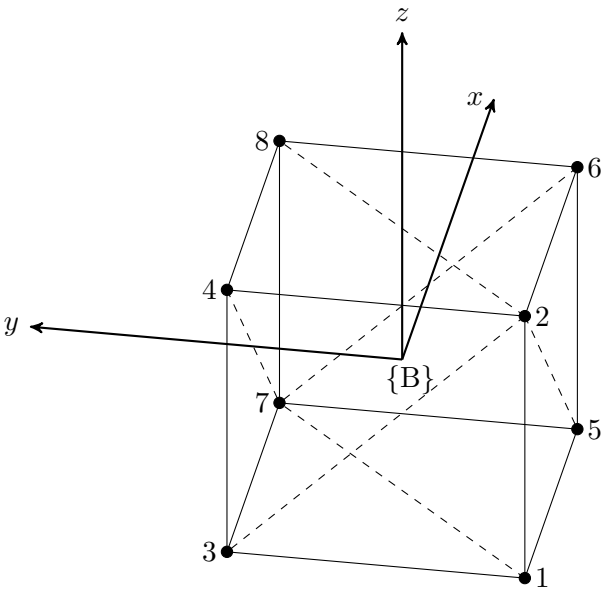


Figure 2.2: Cube modelled with ordered set of points and corresponding triangles

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, shown in Algorithm 2.

Figure 2.3 shows a simplified scenario involving the intersection of a ray with a single triangle. In practice, the algorithm is vectorised to compute the intersections with a *set* of triangles. In this vectorised implementation, many variables represent a matrix whose columns are each vectors. These variables will still be described as vectors to emphasise the operation of the Möller-Trumbore algorithm, rather than the specific implementation details.

The output variables are first initialised to the case that there is no intersection with the scanning direction. x is set to `false` and the range, angle and triangle index outputs are set to return `NaN`.

On line 8, the set of points $\mathbf{P}$ are indexed using the columns of the triangle matrix $\mathbf{T}$ to extract the three vertices corresponding to each triangle. The vectors representing the edges sharing vertex $\mathbf{V}_1$ are computed. from triangles and points, extract vertices of each triangle. The vector $\mathbf{A}$ computed on line 14 represents the translation from the ray origin $\mathbf{o}$ to $\mathbf{V}_1$.

The collection of vectors $\mathbf{B}$ is computed on line 16 by taking the cross product of the scan direction $\mathbf{d}$ and each edge $\mathbf{E}_2$. The determinant $\boldsymbol{\delta}$ of the matrix

$$\mathbf{M} = \begin{bmatrix} \mathbf{e}_1 \\ \mathbf{d} \\ \mathbf{e}_2 \end{bmatrix} \tag{2.20}$$

is computed on line 16. This is first used to determine if the scan direction $\mathbf{d}$ lies in the plane of the triangle by checking if the determinant is close to zero. If so, no intersection can occur. The zero determinant values are then set to `NaN` to avoid a division by zero later.

Beginning on line 21, determinant $\boldsymbol{\delta}$ is used to compute the barycentric coordinates $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, and the distance $\mathbf{s}$ from the origin to the triangle plane along the scan direction $\mathbf{d}$.

The barycentric coordinates are used to determine if the intersection between the scan direction $\mathbf{d}$ and the plane of the triangle lies within the triangle itself.

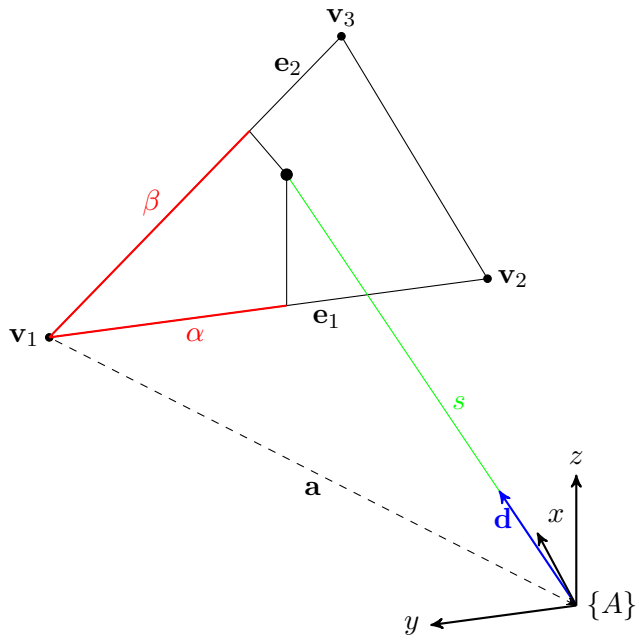


Figure 2.3: Ray-triangle intersection

The vector $\mathbf{x}$ on line 27 now indicates which triangles intersected with the scan direction. $\mathbf{x}$ is used to mask the ranges to the triangles $\mathbf{s}$, to give $\mathbf{r}$; the range to each intersecting triangle.

The minimum range r and triangle index m are determined before computing the angle of incidence θ between $\mathbf{d}$ and the closest triangle.

2.1.4.2 Sensor Noise: `addnoise`

In Algorithm 1, the function `addnoise` takes the ground truth range measurements and produces the noisy range measurements that the observer will actually receive. The noise function f_s is

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

where $\theta(t)$ is the incidence angle of the measurement, ϕ represents the surface properties of the object k that was measured.

For the Hokuyo UBG-04LX-F01 sensor used, range measurements taken at various distances and incidence angles were used to estimate the noise model f_{UBG} which is provided in section 3.1.

2.1.5 Observer Implementation

FIGURES ARE USED TO EXPLAIN FUNCTIONS. FIGURES THEMSELVES NEED SOME EXPLAINING - IE TRIANGLES ON SURFACE, NOT INSIDE CUBE FOR ORIENTATION

2.1.5.1 Estimate: `estimatestate`

The state of the cube at each time step is estimated using the numerical integration method described in section 2.1.1.2. This state estimation is implemented in the function `estimatestate` in Algorithm 1.

Algorithm 2: Möller-Trumbore ray-triangle intersection

input : $\mathbf{o}$ - ray origin
$\mathbf{d}$ - ray direction vector
$\mathbf{P}$ - cube in inertial frame
$\mathbf{T}$ - triangle matrix

output : x - True/False - measurement corresponds to object
r - distance to object in m
θ - incidence angle in rad
m - index of triangle hit

begin
  `/* initialise outputs */`
  $x \longleftarrow 0$
  $r \longleftarrow NaN$
  $\theta \longleftarrow NaN$
  $m \longleftarrow NaN$
  `/* triangle vertexes and edges */`
  $\mathbf{V}_1 \longleftarrow \mathbf{P}[\mathbf{T}[:, 1]]$
  $\mathbf{V}_2 \longleftarrow \mathbf{P}[\mathbf{T}[:, 2]]$
  $\mathbf{V}_3 \longleftarrow \mathbf{P}[\mathbf{T}[:, 3]]$
  $\mathbf{E}_1 \longleftarrow \mathbf{V}_2 - \mathbf{V}_1$
  $\mathbf{E}_2 \longleftarrow \mathbf{V}_3 - \mathbf{V}_1$
  $m = \texttt{size}(\mathbf{V}_1, 1)$
  $\mathbf{A} \longleftarrow \mathbf{o}_{[m \times 1]} - \mathbf{V}_1$
  `/* determinant */`
  $\mathbf{B} \longleftarrow \mathbf{d}_{[m \times 1]} \times \mathbf{E}_2$
  $\boldsymbol{\delta} \longleftarrow \mathbf{E}_1 \cdot \mathbf{B}$
  $\mathbf{y} \longleftarrow |\boldsymbol{\delta}| \leq \mathbf{0}$
  $\boldsymbol{\delta}[\mathbf{y}] \longleftarrow \mathbf{NaN}$
  `/* barycentric coordinates */`
  $\boldsymbol{\alpha} \longleftarrow (\mathbf{A} \cdot \mathbf{B})/\boldsymbol{\delta}$
  $\mathbf{Q} \longleftarrow \mathbf{A} \times \mathbf{E}_1$ *(along dim 2)
  $\boldsymbol{\beta} \longleftarrow (\mathbf{d}_{[n \times 1]} \cdot \mathbf{Q})/\boldsymbol{\delta}$ *(along dim 2)
  $\mathbf{s} \longleftarrow (\mathbf{E}_2 \cdot \mathbf{Q})/\boldsymbol{\delta}$
  `/* intersection vector */`
  $\mathbf{z} \longleftarrow \mathbf{y}$ **and** $(\boldsymbol{\alpha} \geq \mathbf{0})$ **and** $(\boldsymbol{\beta} \geq \mathbf{0})$ **and** $(\boldsymbol{\alpha} + \boldsymbol{\beta} \leq \mathbf{1})$
  $\mathbf{x} \longleftarrow \mathbf{z}$ **and** $(\mathbf{s} \geq \mathbf{0})$
  if $\texttt{any}(\mathbf{x})$ **then**
    $x \longleftarrow 1$
    $\mathbf{x}[\textbf{not } \mathbf{x}] \longleftarrow \mathbf{NaN}$
    $\mathbf{r} = \mathbf{s} \circ \mathbf{x}$
    $r = \texttt{min}(\mathbf{r})$
    $m = \texttt{find}(\mathbf{r} = r, 1)$
    $\mathbf{e}_1 \longleftarrow \mathbf{E}_1[t, :]$
    $\mathbf{e}_2 \longleftarrow \mathbf{E}_2[t, :]$
    $\mathbf{n} = \mathbf{e}_1 \times \mathbf{e}_2$
    $\theta = \texttt{atan2}(|\mathbf{d} \times \mathbf{n}|, \mathbf{d} \cdot \mathbf{n})$
    $\theta = \texttt{min}(\theta, \pi - \theta)$
  end
end

2.1.5.2 Object/background sepraration: `identifyobject`

The observer update function will use range measurements to estimate the state of the cube $\mathbf{X}_c$. In order to perform accurately, the observer must only use range measurements that correspond to the cube. The function `identifyobject` in Algorithm 1 uses range measurements and knowledge of the configuration of the environment to separate measurements of the cube and background.

The binary variable c indicates whether the current range measurement corresponds to cube ($c =$ `true`) or the background ($c =$ `false`). It is assumed that initially the sensor will be observing the background, so $c_0 =$ `false`.

The scheme used to identify range measurements corresponding to the cube is shown in Algorithm 3. There are two assumptions that may be used.

1. The *difference assumption* relies on the assumption that the cube and background objects are continuous. Differences in consecutive range measurements larger than Δ_{max} indicate a discontinuity, implying that a new object is being measured. When this occurs, the value of c changes.

2. The *range assumption* is used when the maximum distance to the cube and minimum distance to the background are restricted. Range measurements within r_{max} correspond to the cube while larger ranges correspond to the background.

Algorithm 3: Target/background object separation

input : $differenceAssumption$ - true/false
$rangeAssumption$ - true/false
Δ_{max} - max diff between measurements of same object
r_{max} - max range for cube
c - true/false - current measurement is of cube
$\mathbf{r}_{i+1}$ - distance to object at $t = i + 1$
$\mathbf{r}_i$ - distance to object at $t = i$

output : c - true/false

```
1  begin
2      if differenceAssumption then
3          if |r_{i+1} − r_i| > Δ_max then
4              c =  mod (c + 1, 2)
5          end
6      end
7      if rangeAssumption then
8          if r_{i+1} > r_max then
9              c = 0
10         end
11     end
12 end
```

2.1.5.3 Update: `updatestate`

If the `identifyobject` function identifies a range measurement as corresponding to the cube, the `updatestate` function is used to update the state estimate of the cube $\hat{\mathbf{X}}_c$. $\hat{\mathbf{X}}_c$ is updated using the previous cube state estimate, current sensor state, and sets of measured and predicted

range measurements chosen according to a set of indexes $\mathbf{a}(t)$.

$$\hat{\mathbf{X}}_c(k+1) = f(\mathbf{X}_s(t), \hat{\mathbf{X}}_c(k), \mathbf{r}(\mathbf{a}(t)), \hat{\mathbf{r}}(\mathbf{a}(t))) \tag{2.22}$$

The pose of $\hat{\mathbf{X}}_c$ is corrected by adjusting $\hat{\mathbf{S}}_c$, $\hat{\mathbf{T}}_c$ or $\hat{\mathbf{W}}_c$, or some combination of the three. The orientation is adjusted by rotating about an axis $\mathbf{r}_{update}$. The position is adjusted by translating in the direction of $\mathbf{p}_{update}$ $\mathbf{r}_{update}$ and $\mathbf{p}_{update}$ are scaled differently, depending on whether they are applied to $\hat{\mathbf{S}}_c$, $\hat{\mathbf{T}}_c$ or $\hat{\mathbf{W}}_c$. The size update s_{update} is independent of the pose update scheme used.

Input ranges:
A set of four range measurements forming a quadrilateral are used in the state update. The four ranges are chosen with an ordered sequence of indexes $u(t)$. At a time step ii, the set of time steps used is in the update function is

$$\mathbf{u}(ii) = \{ii, (ii-1), (ii-n_{scans}), (ii-1-n_{scans})\} \tag{2.23}$$

It is possible that the measured or predicted ranges do not exist for some time steps in $\mathbf{u}(ii)$, as the range may have corresponded to the background rather than the cube. Thus, the measurement indexes $\tilde{\mathbf{u}}(ii)$ and estimation indexes $\hat{\mathbf{u}}(ii)$ will be subsets of, but not necessarily congruent to $\mathbf{u}(ii)$.

Orientation update:

The method used to correct the orientation of the cube state estimate $\hat{\mathbf{X}}_c$ is shown in Figure 2.4.

In order to estimate the orientation of the cube, at least 3 ranges are required from both the prediction and measurement: $|\hat{\mathbf{u}}| \geq 3$ and $|\tilde{\mathbf{u}}| \geq 3$. If all four indexes are present, the ranges from the last time step $(ii-1-n_{scans})$ is ignored.

For both the measurement and prediction, the points of intersection ${}^F\mathbf{P}(\mathbf{u})$ between the set of scanning directions ${}^F\mathbf{D}(\mathbf{u})$ and the cube are computed using the set of range measurements $\mathbf{r}(\mathbf{u})$.

$${}^F\mathbf{P}(\mathbf{u}) = {}^F\mathbf{D}(\mathbf{u})\mathbf{r}(\mathbf{u}) \tag{2.24}$$

The normal to the plane formed by the three points is then computed.

$$\mathbf{n} = [{}^F\mathbf{P}(u_2) - {}^F\mathbf{P}(u_1)] \times [{}^F\mathbf{P}(u_3) - {}^F\mathbf{P}(u_1)] \tag{2.25}$$

Because a cube has 24 regular isometries, it is not necessary to effectively align the reference frames of the estimated and true cubes. Any face of the estimated cube can be aligned with any face of the true cube, and the maximum rotation correction will be $\pi/4$ radians.

The angle between the two normals ψ is computed as

$$\psi = \texttt{atan2}(|\hat{\mathbf{n}} \times \tilde{\mathbf{n}}|, \hat{\mathbf{n}} \cdot \tilde{\mathbf{n}}) \tag{2.26}$$

The axis $\mathbf{r}_{update}$ that the estimated cube orientation $\hat{\mathbf{R}}_c$ will be rotated by is computed by taking the cross product of the predicted and measured normals. The direction of rotation is changed if the angle ψ between the two normals is greater than $\pi/4$ radians.

$$\mathbf{r}_{update} = \texttt{sign}\left(\frac{\pi}{4} - \psi\right)(\hat{\mathbf{n}} \times \tilde{\mathbf{n}}) \tag{2.27}$$

where

$$\texttt{sign}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases} \tag{2.28}$$

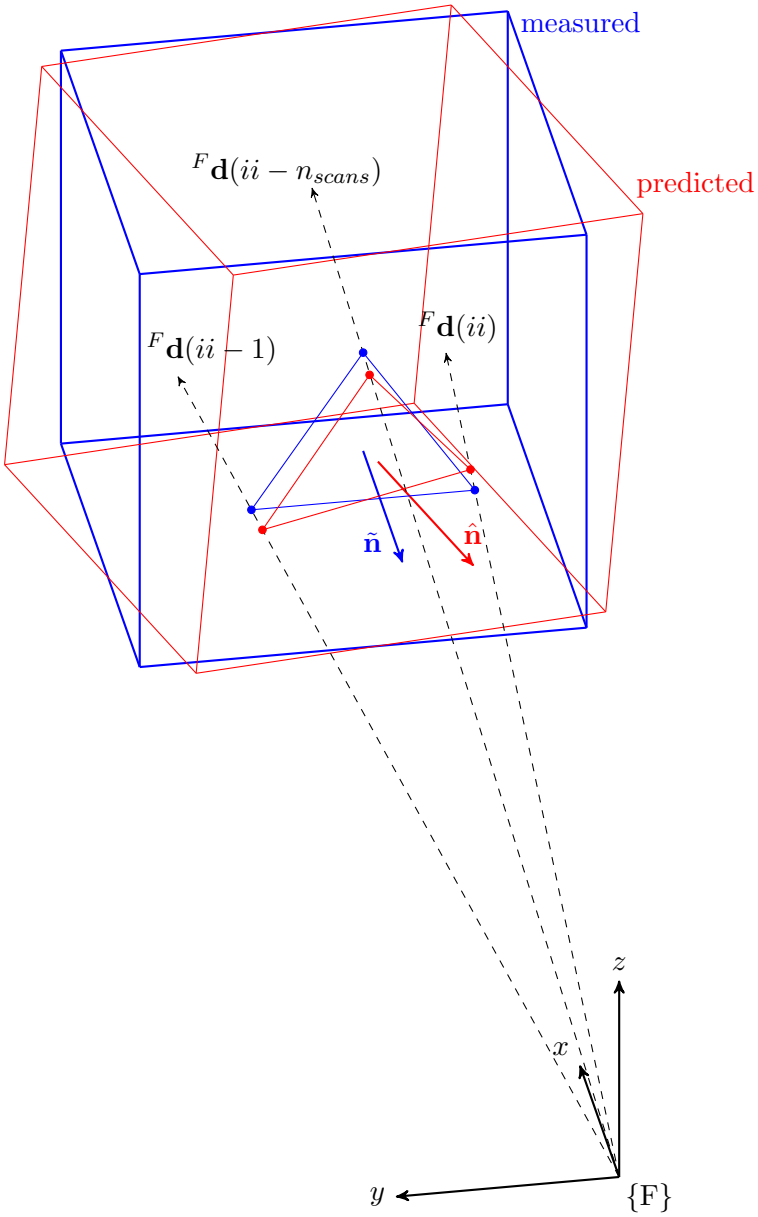


Figure 2.4: Orientation update

To update the screw matrix $\hat{\mathbf{S}}_c$, $\mathbf{r}_{update}$ is converted to a rotation matrix $\mathbf{R}_{update}$ with Rodrigues' rotation formula (equation 1.19). The correction is then applied as:

$$\hat{\mathbf{R}}_c(k+1) = R_{scale}\mathbf{R}_{update}\hat{\mathbf{R}}_c(k) \tag{2.29}$$

To update via the screw or wrench, $\mathbf{r}^{\wedge}_{update}$ is scaled and then added to the angular velocity or angular acceleration respectively.

$$\hat{\boldsymbol{\omega}}^{\wedge}_c(k+1) = \hat{\boldsymbol{\omega}}^{\wedge}_c(k) + \omega_{scale}\mathbf{r}^{\wedge}_{update} \tag{2.30}$$

$$\hat{\boldsymbol{\alpha}}^{\wedge}_c(k+1) = \hat{\boldsymbol{\alpha}}^{\wedge}_c(k) + \alpha_{scale}\mathbf{r}^{\wedge}_{update} \tag{2.31}$$

The scale factor depends angle depends on whether $\mathbf{r}_{update}$ is applied via $\hat{\mathbf{S}}_c$, $\hat{\mathbf{T}}_c$ or $\hat{\mathbf{W}}_c$.

Position update:
The method used to correct the position of $\hat{\mathbf{X}}_c$ is shown in Figure 2.5.

In order to estimate the position of the cube, at least 1 range measurement is required from both the prediction and measurement: $|\hat{\mathbf{u}}| \geq 1$ $|\tilde{\mathbf{u}}| \geq 1$. Additionally, scan directions at time steps $ii, (ii-1)$ and $(ii-1-n_{scans})$ are required. The scan direction ${}^{F}\mathbf{d}(t)$ must be within the sensor's field of view at these time steps.

The points of intersection are calculated using equation 2.24. The average of these points is computed to give the mean estimated position $\hat{\boldsymbol{\mu}}_{\mathbf{p}}$ and the mean measured position $\tilde{\boldsymbol{\mu}}_{\mathbf{p}}$.

The x, y and z components of the update vector may very significantly in size due to the scanning behaviour of the sensor. It is necessary to scale the position update vector according to these components. The mean of all predicted and measured ranges μ_r is computed. Four points $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2$ and $\mathbf{p}_3$ are computed to be used in scaling:

$$\begin{array}{rcl} \mathbf{p}_0 & = & \mathbf{p}_s(t) = {}^{F}_{F}\mathbf{p}_A(t) \\ \mathbf{p}_1 & = & \mu_r {}^{F}\mathbf{d}(ii) \\ \mathbf{p}_2 & = & \mu_r {}^{F}\mathbf{d}(ii-1) \\ \mathbf{p}_3 & = & \mu_r {}^{F}\mathbf{d}(ii-1-n_{scans}) \end{array} \tag{2.32}$$

The update vector is computed by scaling the mean intersection points with these four points:

$$\mathbf{p}_{update} = \begin{bmatrix} \frac{1}{|\mathbf{p}_1-\mathbf{p}_0|} & 0 & 0 \\ 0 & \frac{1}{|\mathbf{p}_2-\mathbf{p}_1|} & 0 \\ 0 & 0 & \frac{1}{|\mathbf{p}_3-\mathbf{p}_2|} \end{bmatrix} (\tilde{\boldsymbol{\mu}}_{\mathbf{p}} - \hat{\boldsymbol{\mu}}_{\mathbf{p}}) \tag{2.33}$$

The screw, twist and wrench are corrected using $\mathbf{p}_{update}$. The scaling factor used depends on whether the update is performed via the screw, twist or wrench.

$$\hat{\mathbf{p}}_c(k+1) = \hat{\mathbf{p}}_c(k) + p_{scale}\mathbf{p}_{update} \tag{2.34}$$

$$\hat{\mathbf{v}}_c(k+1) = \hat{\mathbf{v}}_c(k) + v_{scale}\mathbf{p}_{update} \tag{2.35}$$

$$\hat{\mathbf{a}}_c(k+1) = \hat{\mathbf{a}}_c(k) + a_{scale}\mathbf{p}_{update} \tag{2.36}$$

Size update:
In order to correct the size of the cube, at least 1 range measurement is required from both the prediction and measurement: $|\hat{\mathbf{u}}| \geq 1$ $|\tilde{\mathbf{u}}| \geq 1$. The size update scheme also differs based on the sets predicted and measured ranges.

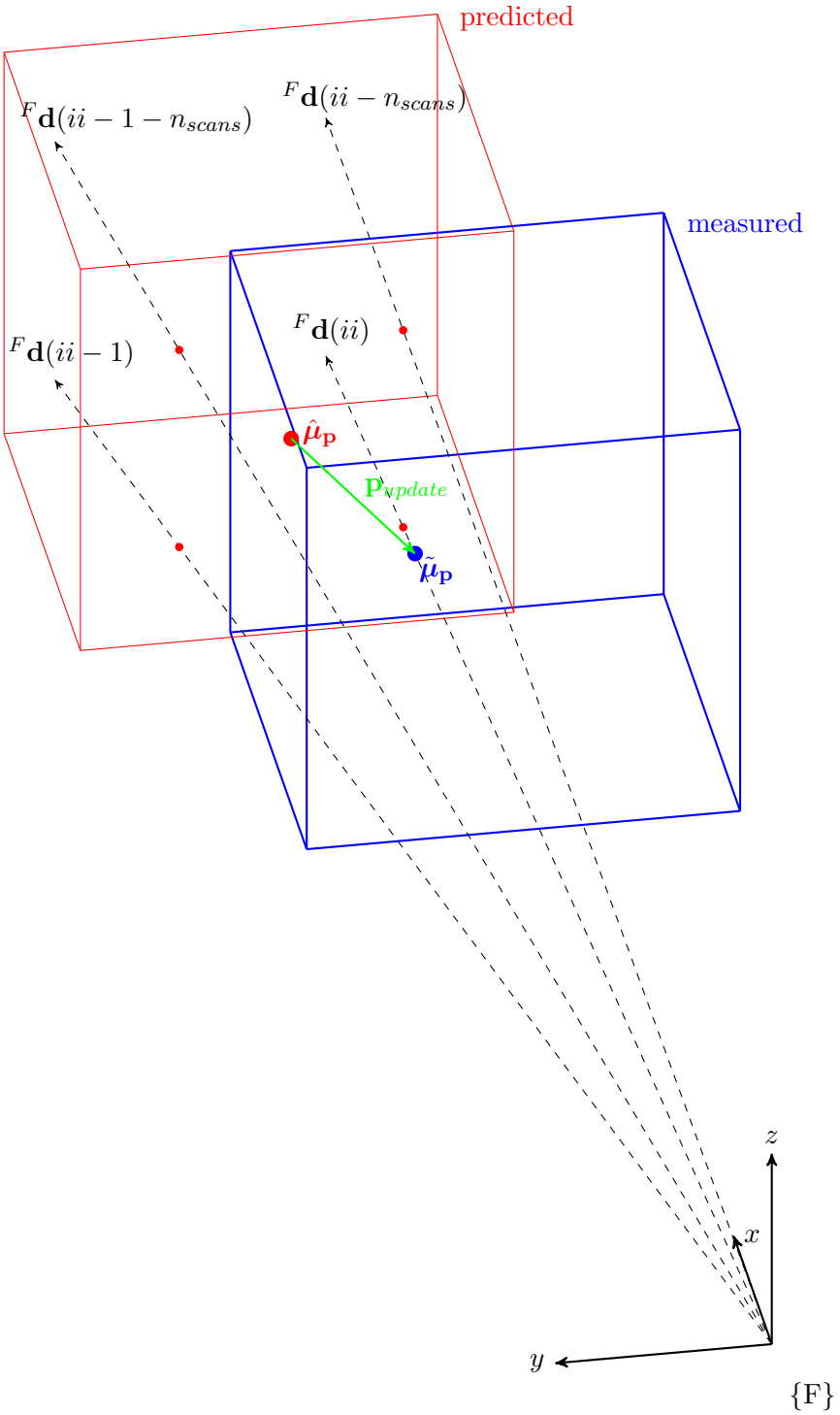


Figure 2.5: Position update

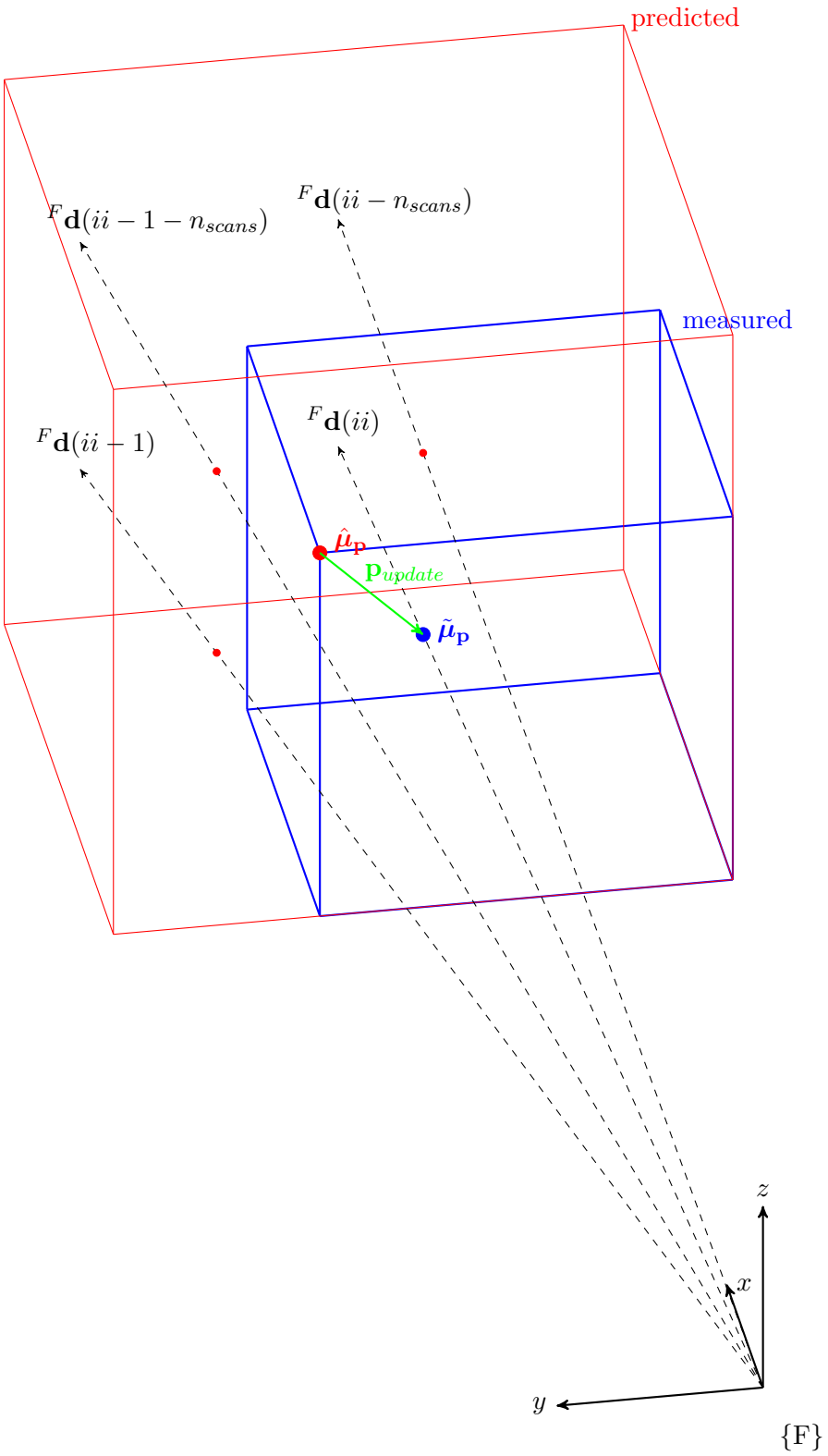


Figure 2.6: Size update - case 1

For the case where a different pattern of ranges is observed($\hat{\mathbf{u}} \neq \tilde{\mathbf{u}}$), the update method is shown in Figure 2.6.

The dot product from the vector $\mathbf{p}_{update}$ computed for the position update and the current scan direction is taken:

$$s_{update} = \mathbf{p}_{update} \cdot {}^{F}\mathbf{d}(ii) \tag{2.37}$$

For the case where the same pattern of ranges is observer ($\hat{a} = \tilde{a}$), the update method is shown in Figure 2.7.

The size update is taken as the difference in the means of the measured and predicted ranges:

$$s_{update} = \tilde{\mu}_r - \hat{\mu}_r \tag{2.38}$$

In both cases, the cube size estimate is updated by scaling s_{update} and adding this to the previous estimate:

$$s(k+1) = s(k) + s_{scale} s_{update} \tag{2.39}$$

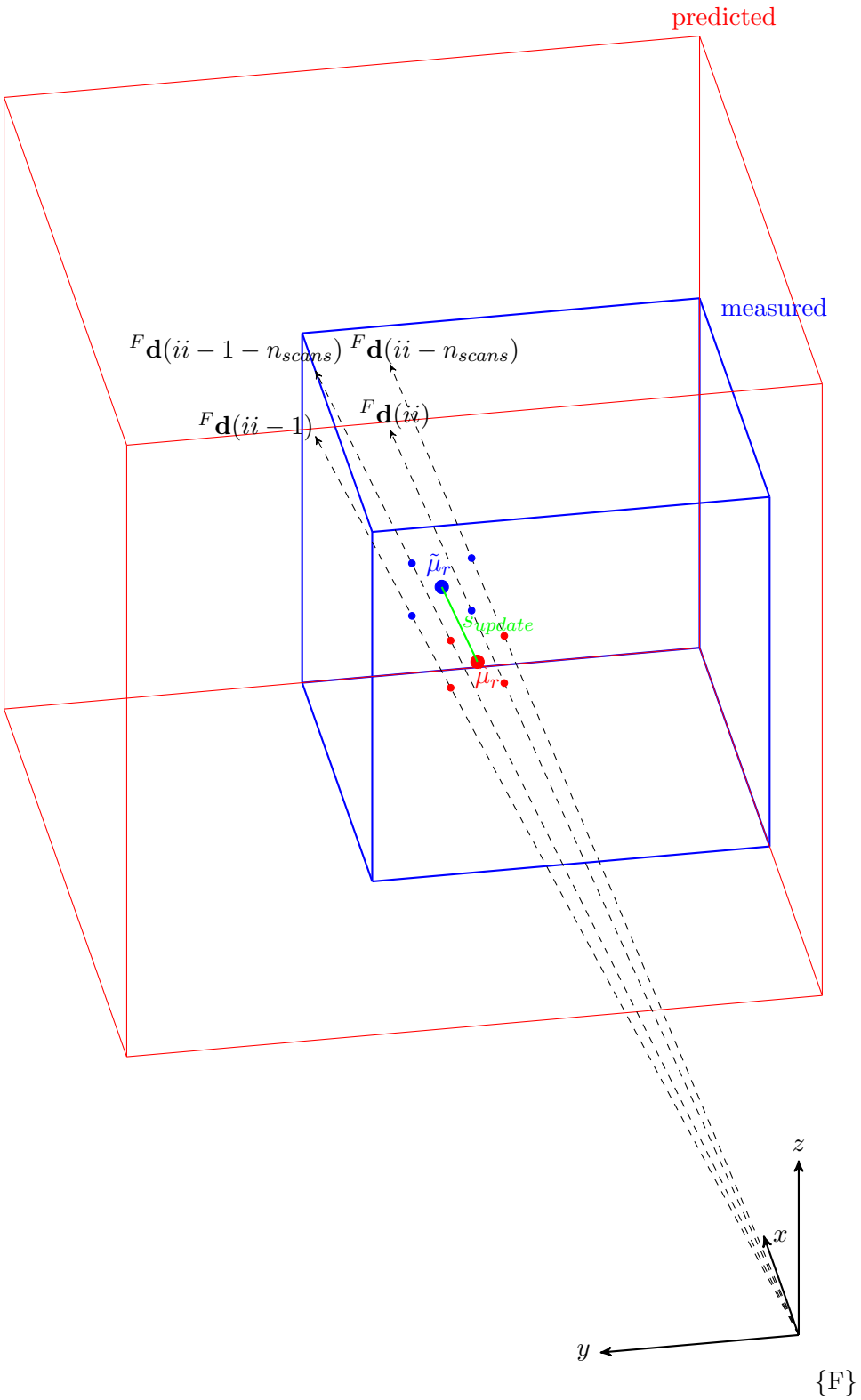


Figure 2.7: Size update - case 2

2.2 Results

2.2.1 testing scheme? think of a title...

tested stationary, rotating, translating, rotating & translating for cases where 1,2,3 faces visible to sensor. tested with and without noise.
Would take too long to perform exhaustive analysis of solution space, testing all the parameters. Instead, will present key results...

2.2.2 Orientation correction

stationary - converges for no noise Figure 2.8.
Doesn't converge to 0 due to noise, but still stable. **THIS DUE TO NOISE FLOOR - INVESTIGATE IF THERE IS TIME - REDUCE NOISE AND SEE IF IT GETS CLOSER** Figure 2.9.

Figure 2.10 - rotating. Updated orientation via screw. Very 'jagged'. Lags slightly, catches up.
Figure 2.11 - rotating. Updated orientation via twist. over/undershoots much more than screw. **MENTION: control theory solution - feed forward control. anticipate overshoot, correct before it occurs. FUTURE WORK - DEVELOP OBSERVER THEORY ANALOGUE**

Figure 2.12 - noisy, rotating. Updated orientation via screw, kind of keeps up with rotation but error caused by noise. Figure 2.13 - noisy, rotating. Updated orientation via twist, overshoot error.

2.2.3 Position correction

stationary - works for only 1 face visible. Without noise - Figure 2.14. With noise - 2.15.

doesnt work for 2/3 faces (results 20). Figure 2.16. Requires perpendicular - multiple faces and edges mean update position vector points in wrong direction.

2.2.4 Size correction

works for stationary and moving, noise and no noise, 1,2,3 faces
Figure 2.17 - noise, stationary, 3F visible - size correction
Figure 2.18 - noise, rotating + translating - size correction

2.2.5 Orientation and size

works in same cases orientation alone works
Figure 2.19:stationary, noisy, orientation & size works (results 9).
Figure 2.20 - rotating, no noise, orientation updated via screw, size updated as normal.
Figure 2.21 - rotating, no noise, orientation updated via twist, size updated as normal.

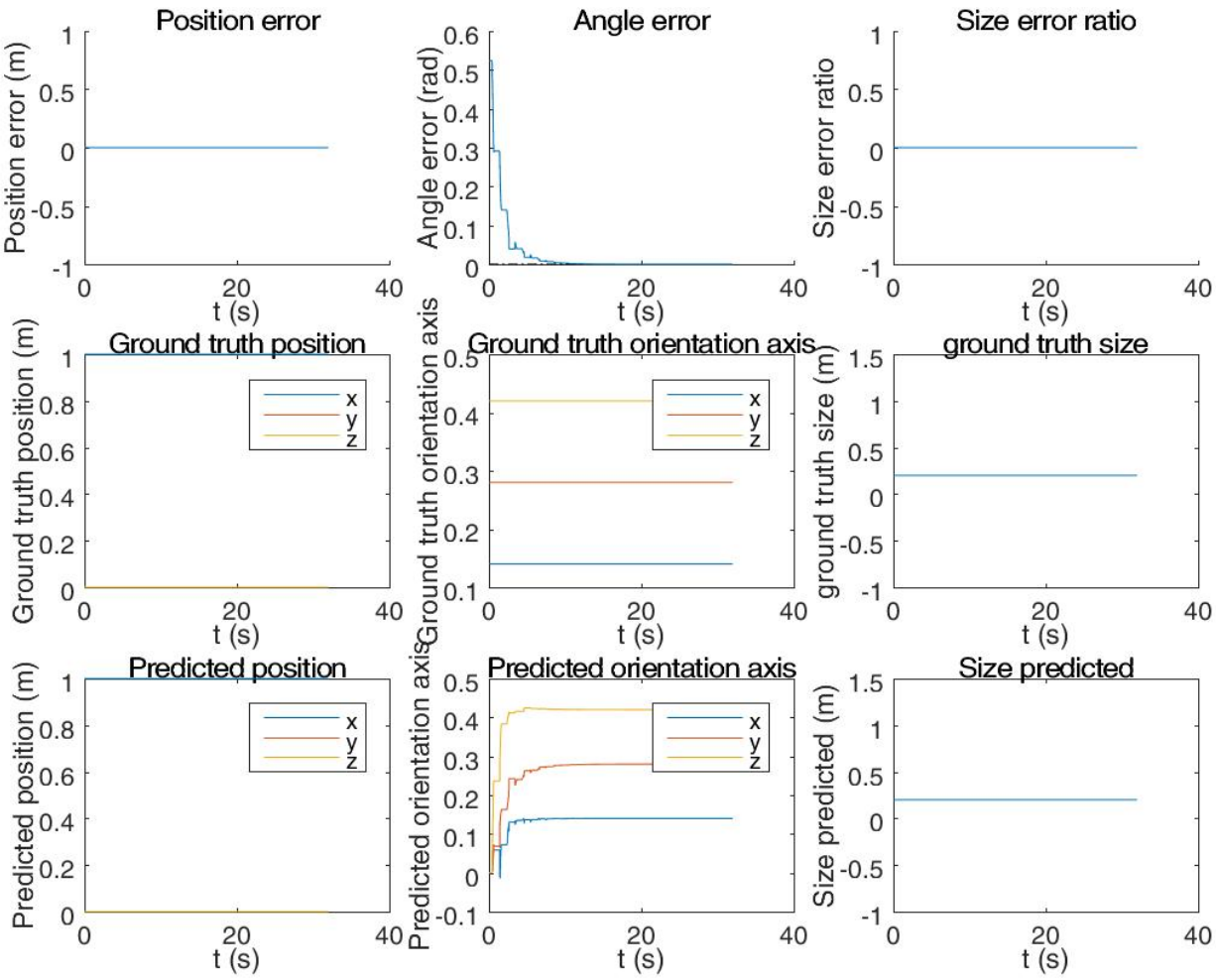


Figure 2.8: Orientation correction - no noise

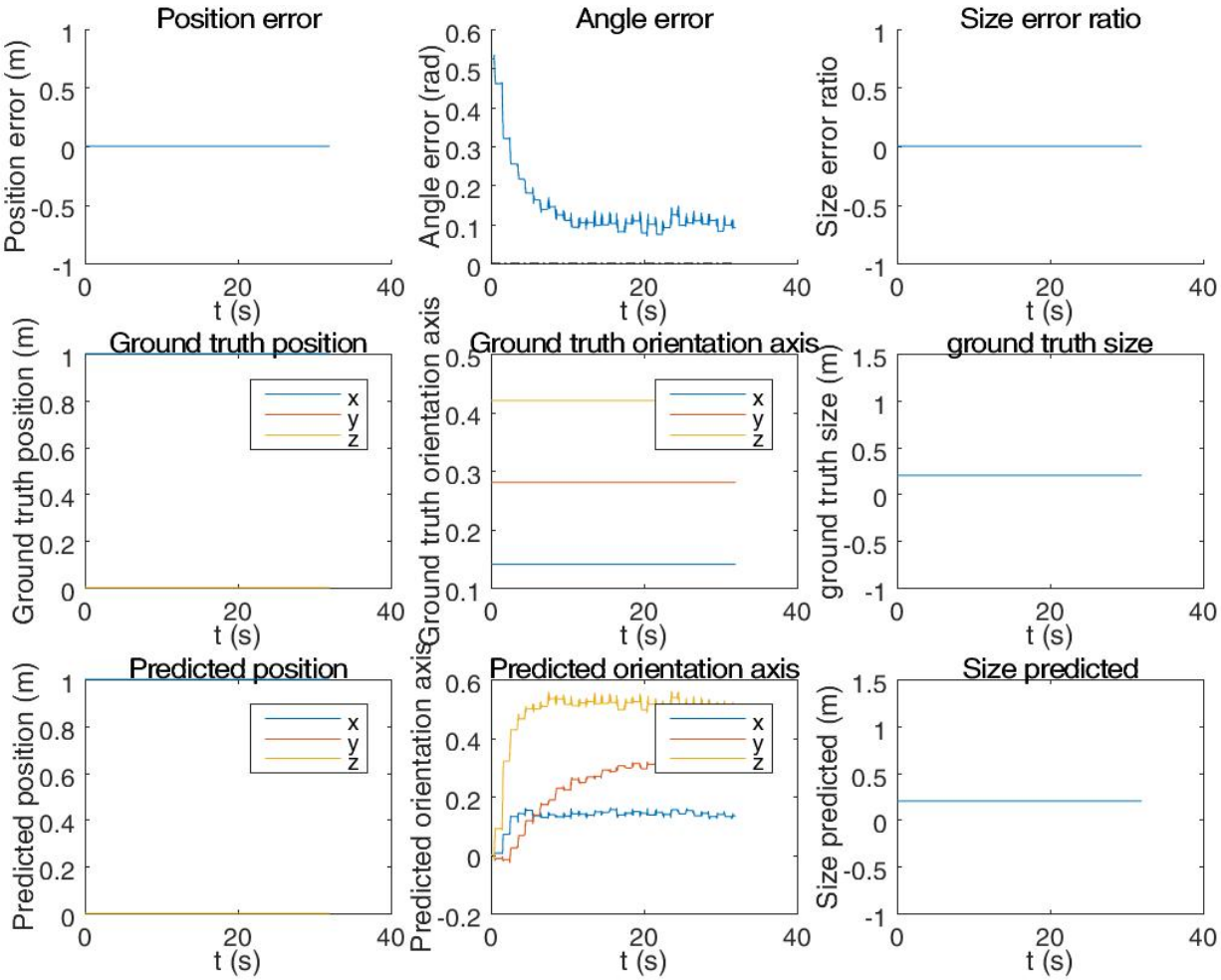


Figure 2.9: Orientation correction - noise

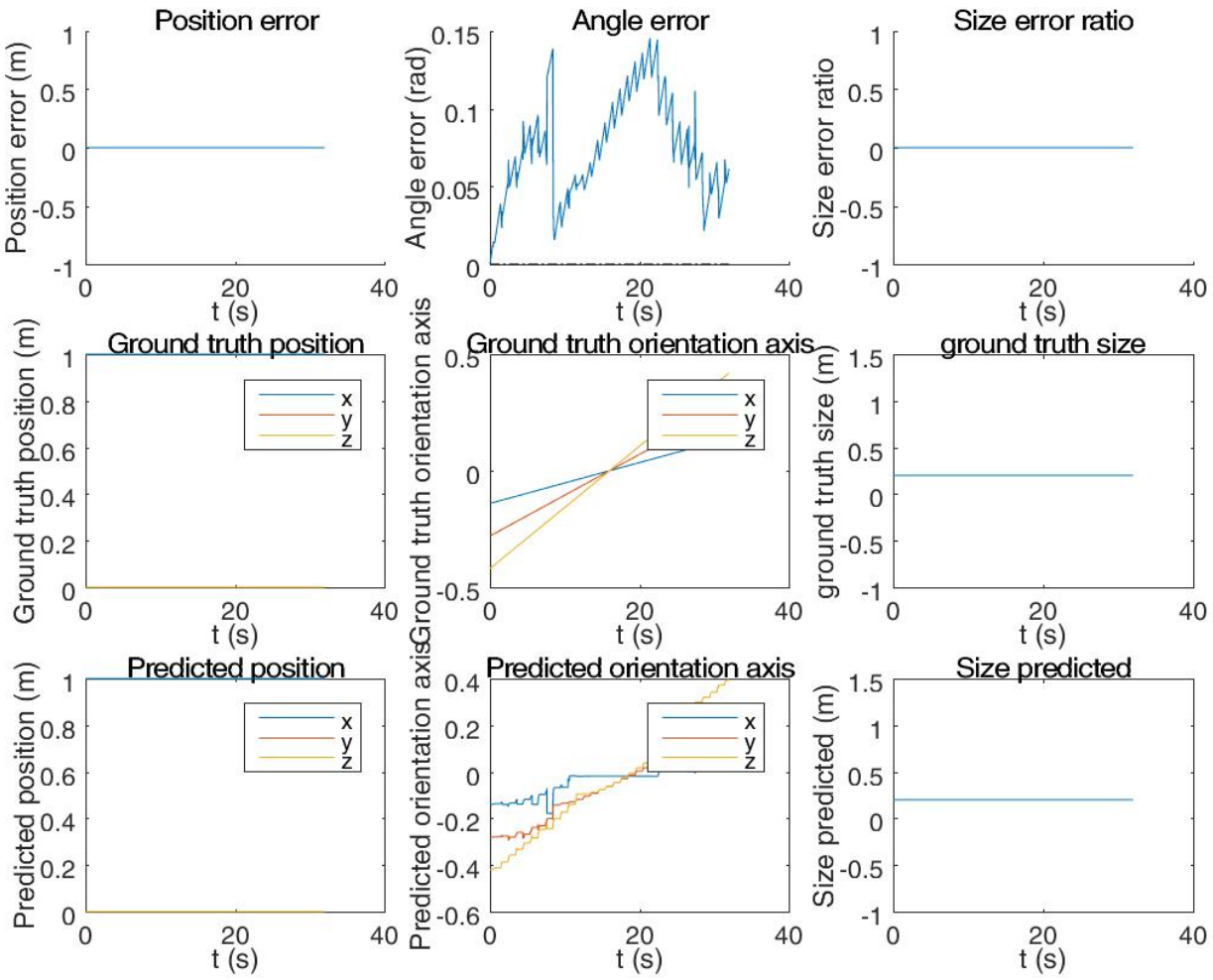


Figure 2.10: rotating - Orientation correction via screw - no noise

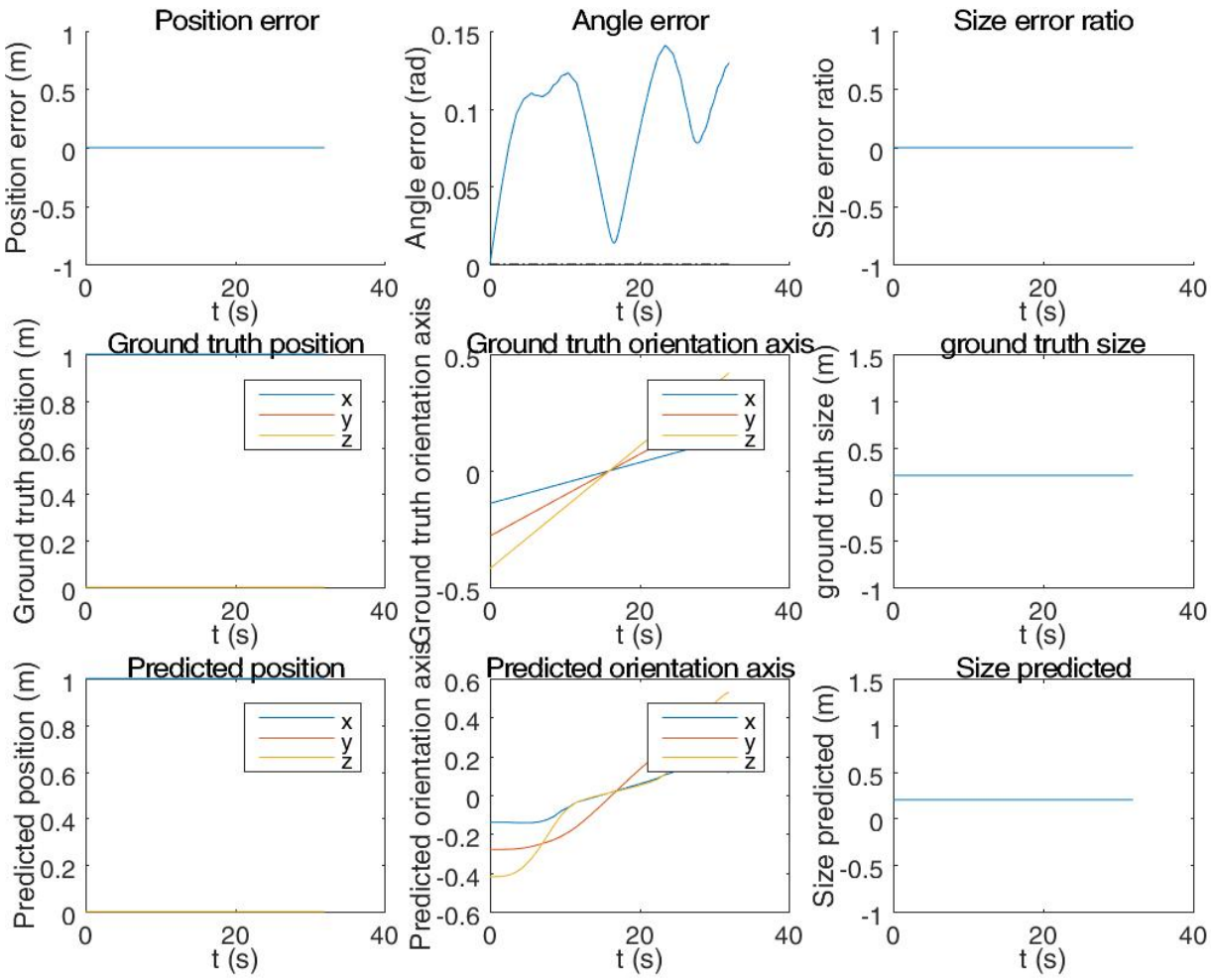


Figure 2.11: rotating - Orientation correction via twist - no noise

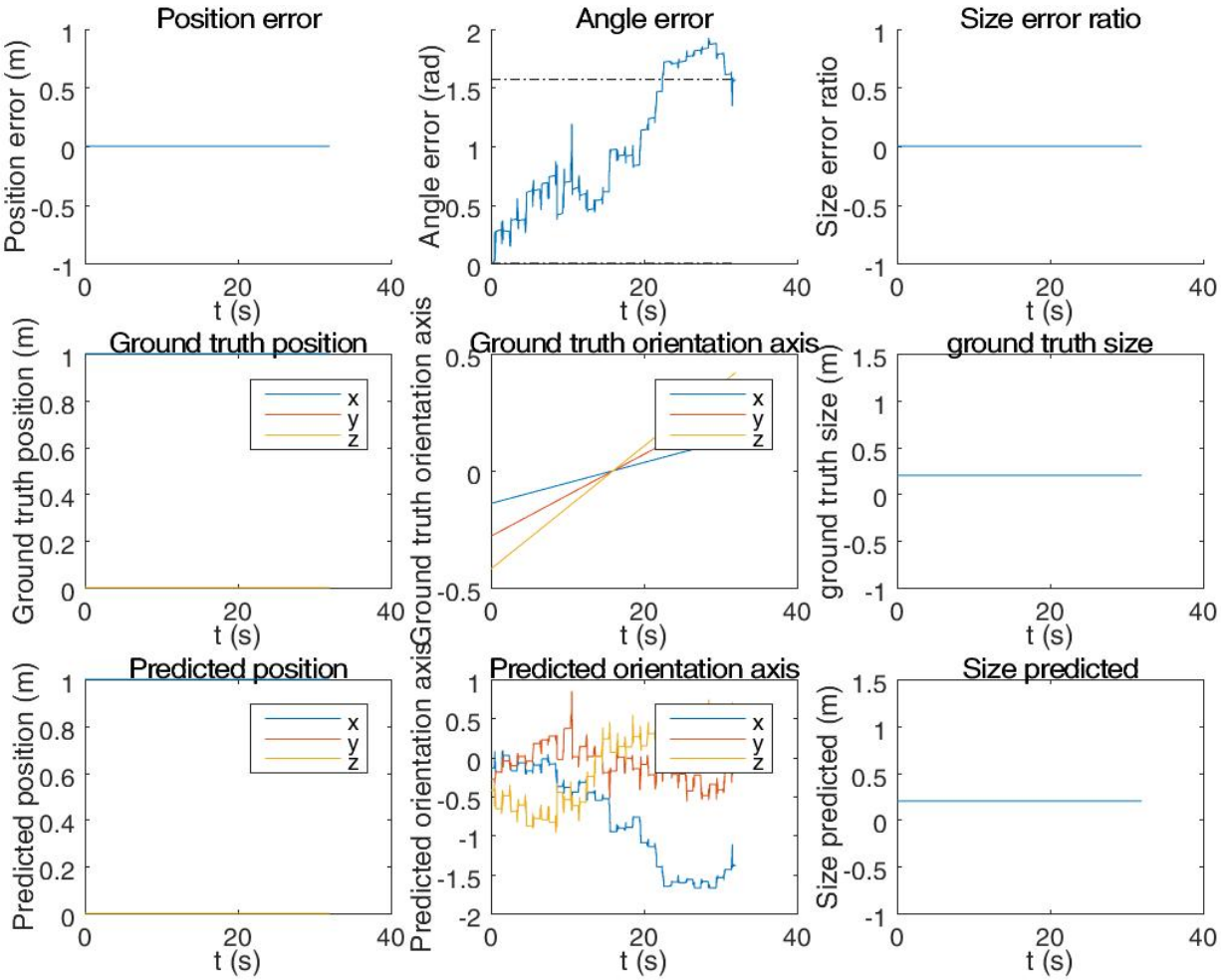


Figure 2.12: rotating - Orientation correction via screw - noise

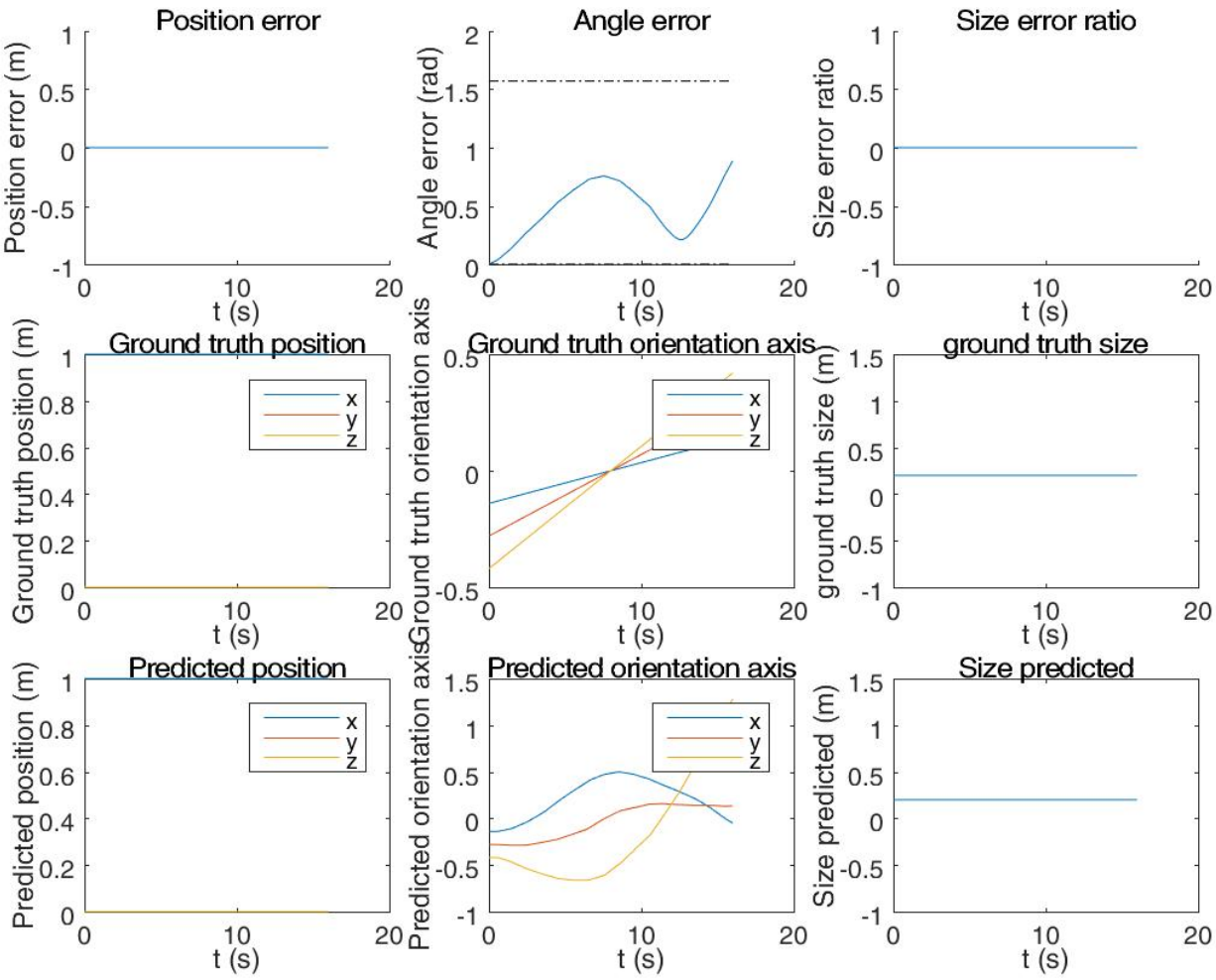


Figure 2.13: rotating - Orientation correction via twist - noise

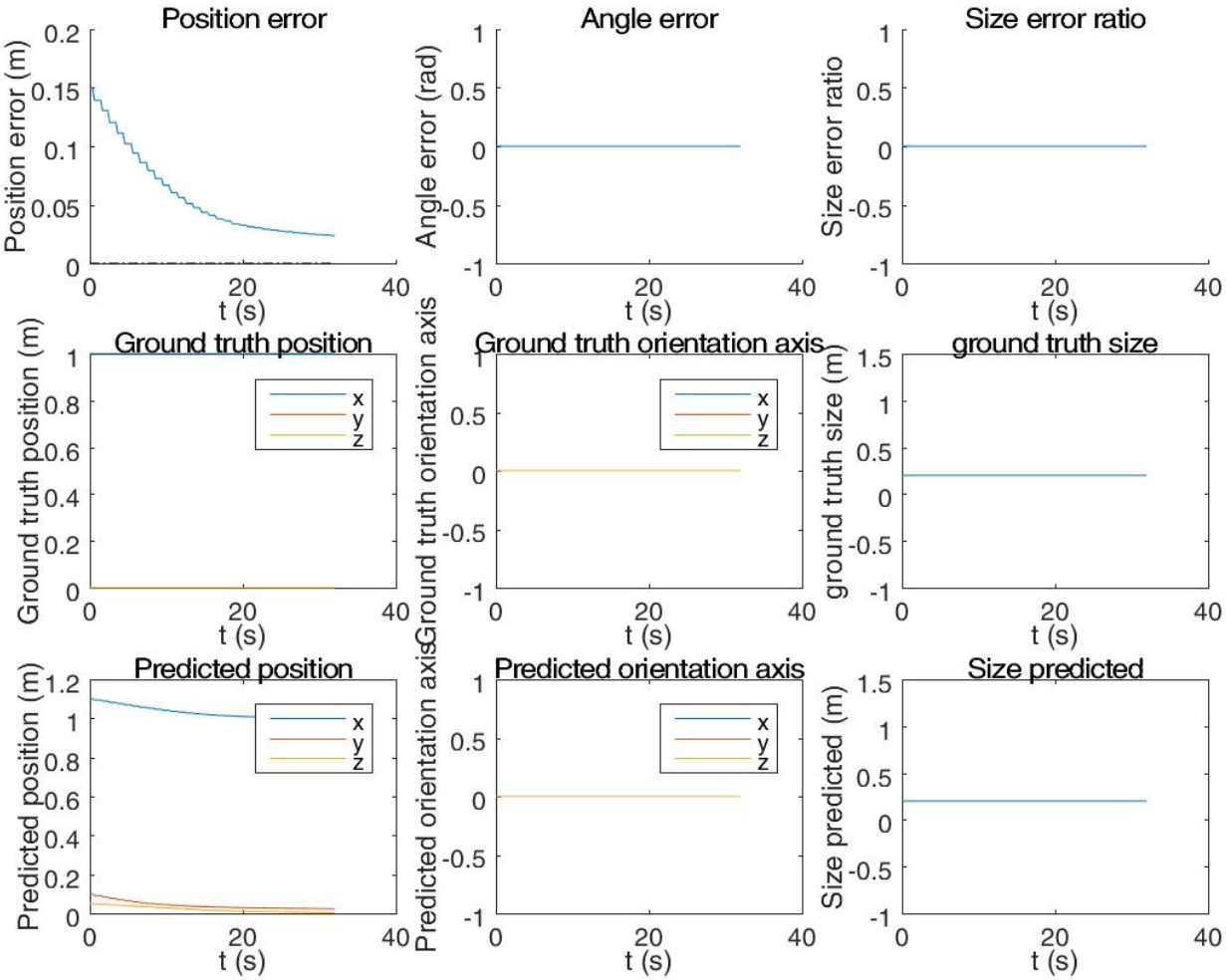


Figure 2.14: stationary, 1 face visible - position correction

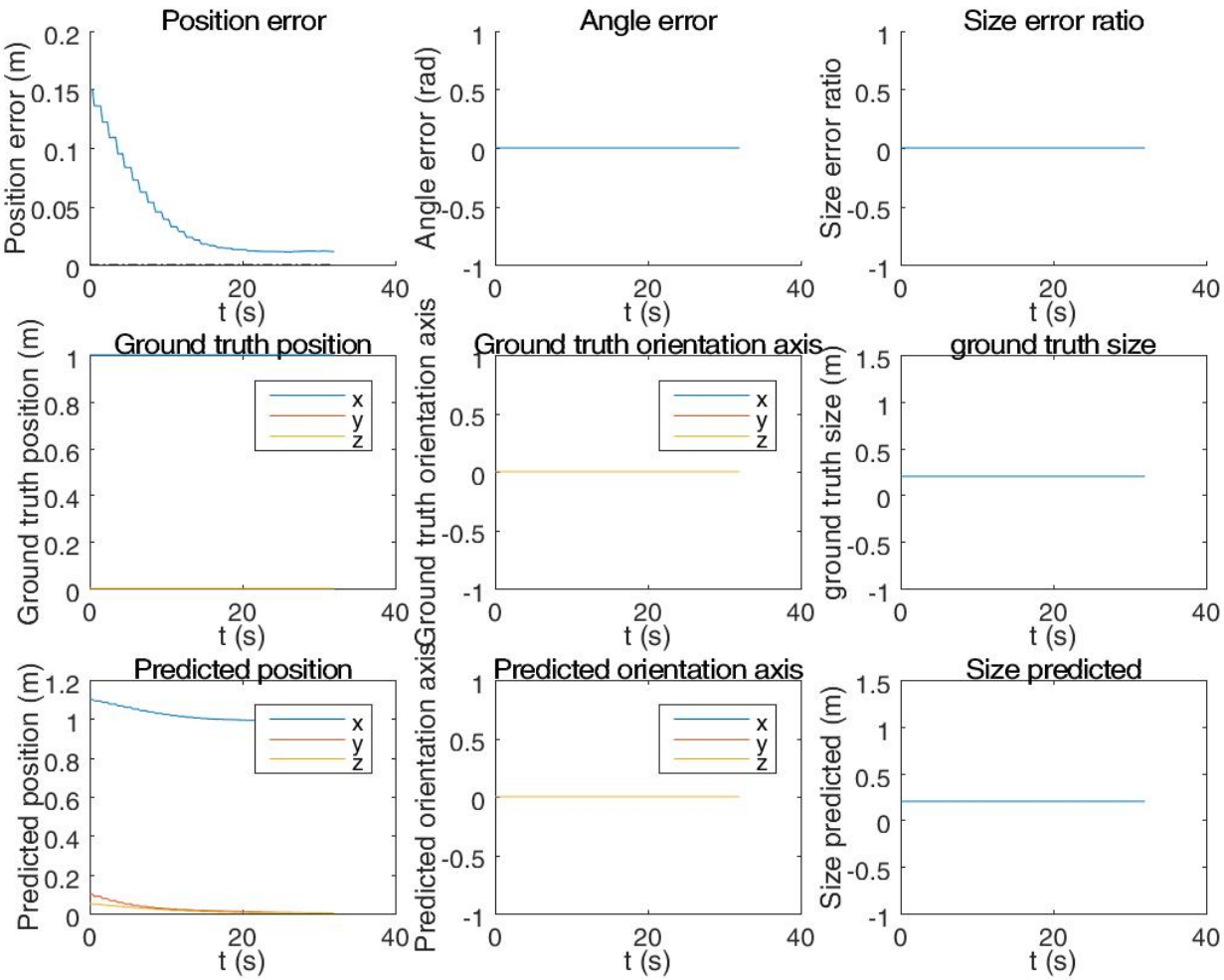


Figure 2.15: stationary with noise, 1 face visible - position correction

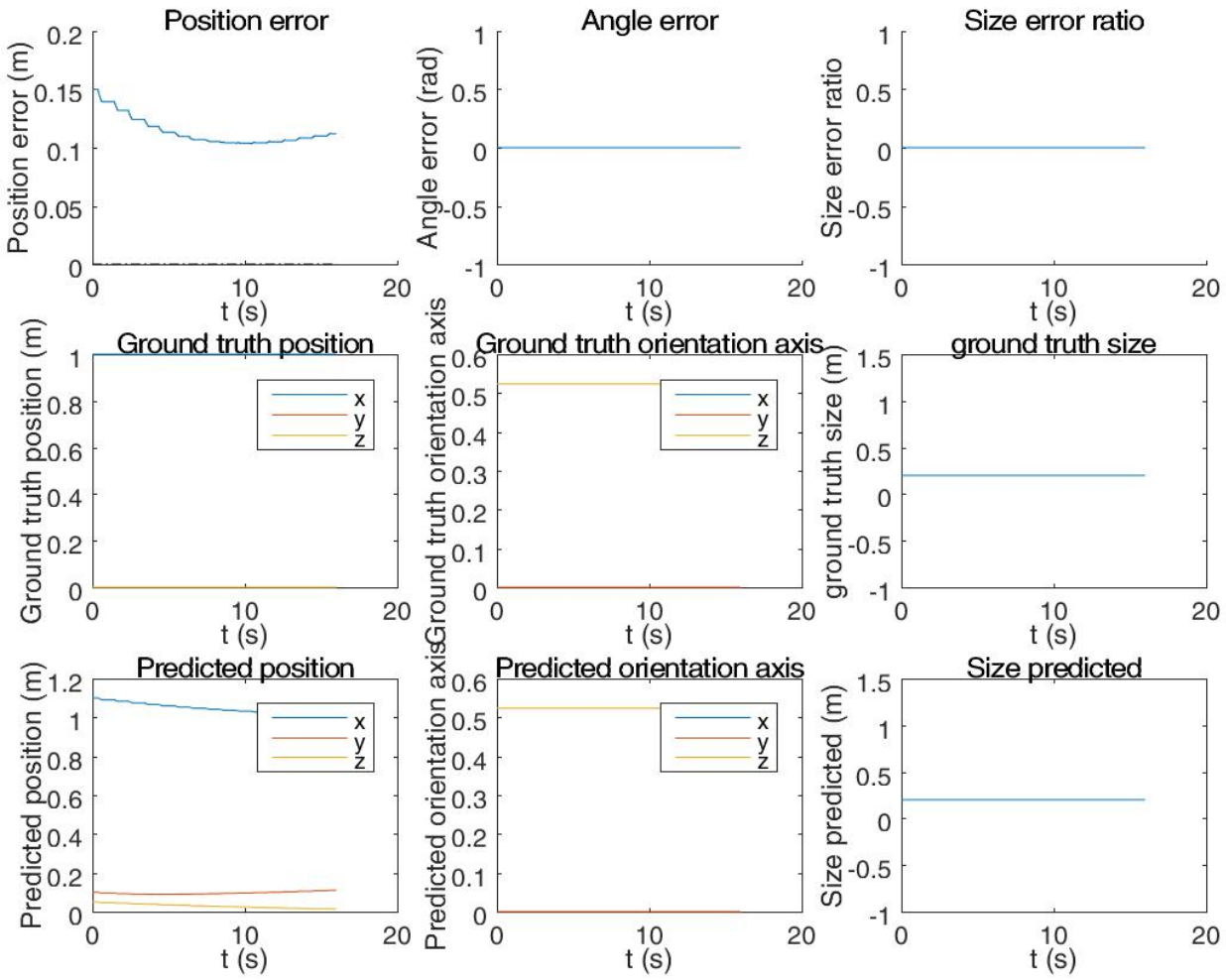


Figure 2.16: stationary with noise, 2 faces visible - position correction doesn't work

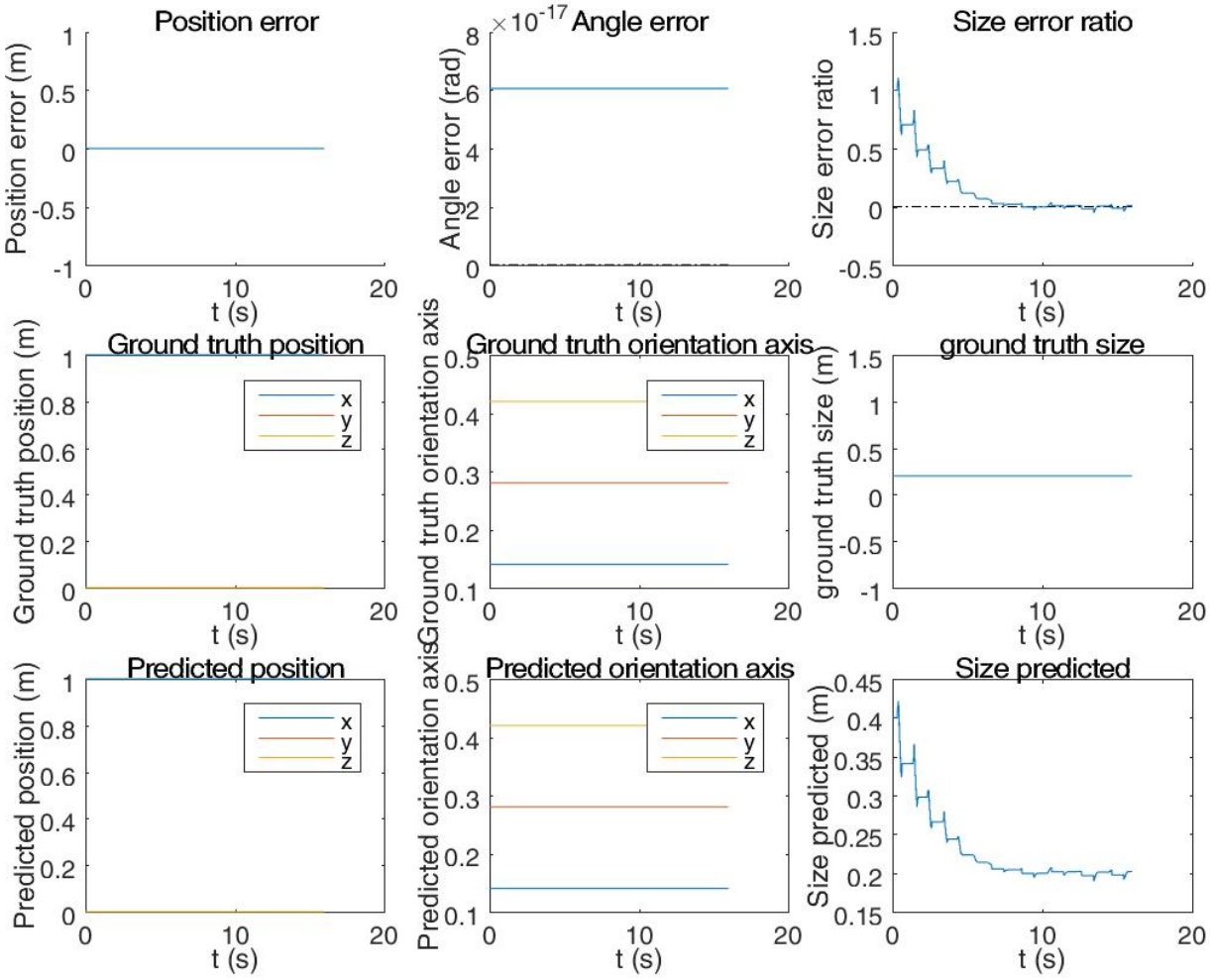


Figure 2.17: stationary with noise, 3 faces visible - size correction

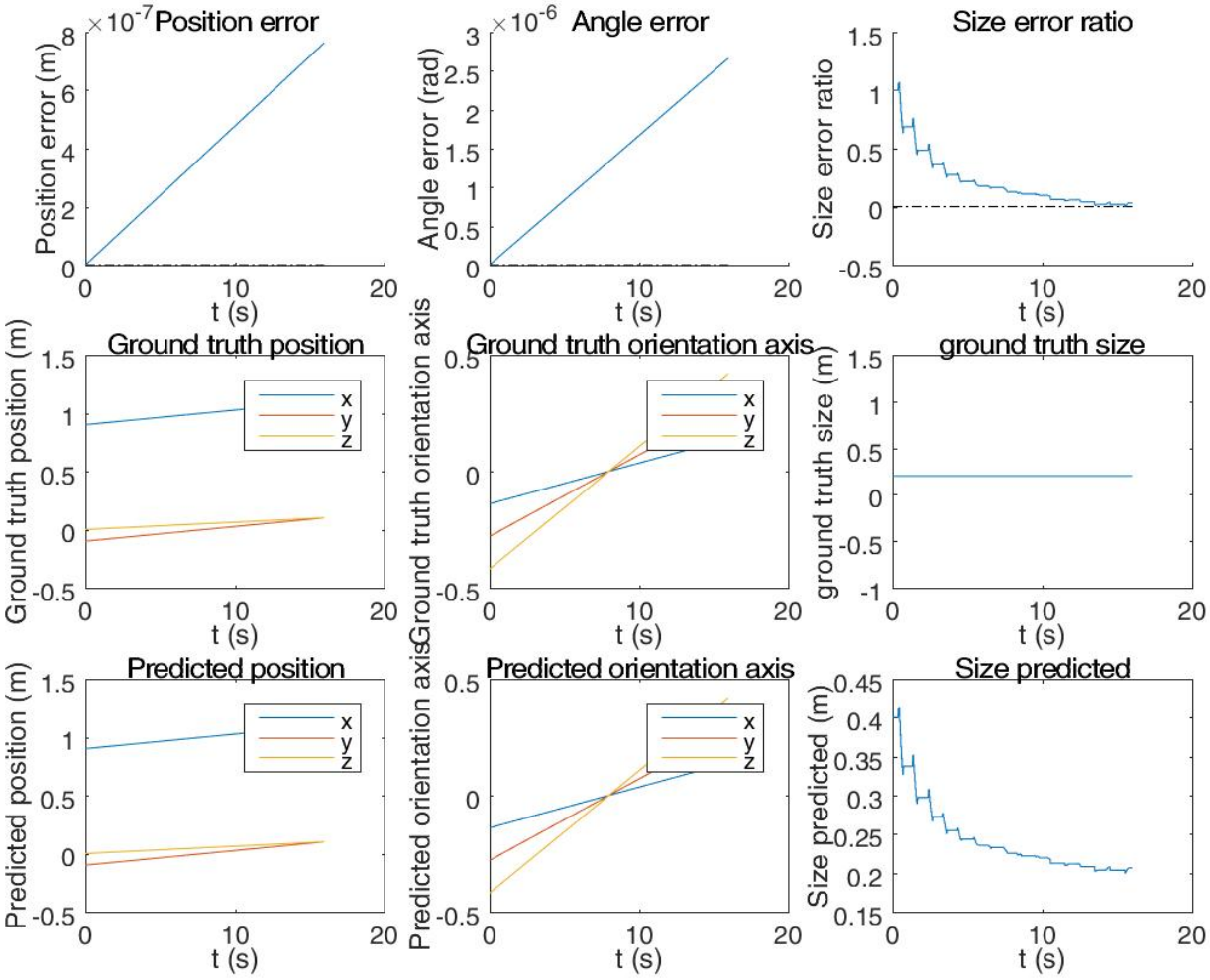


Figure 2.18: rotating + translating with noise, 3 faces visible - size correction

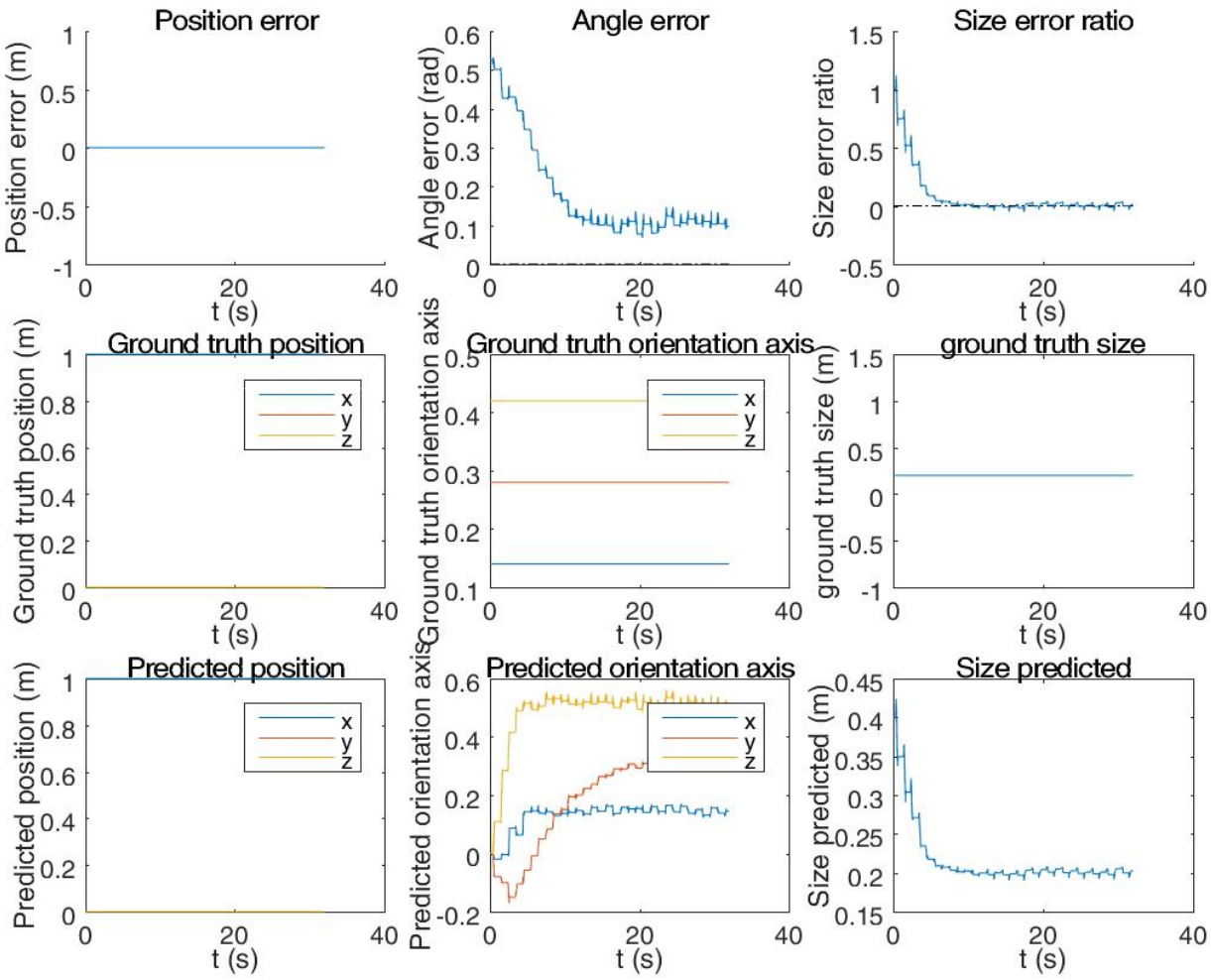


Figure 2.19: stationary with noise, 3 faces visible - orientation and size correction

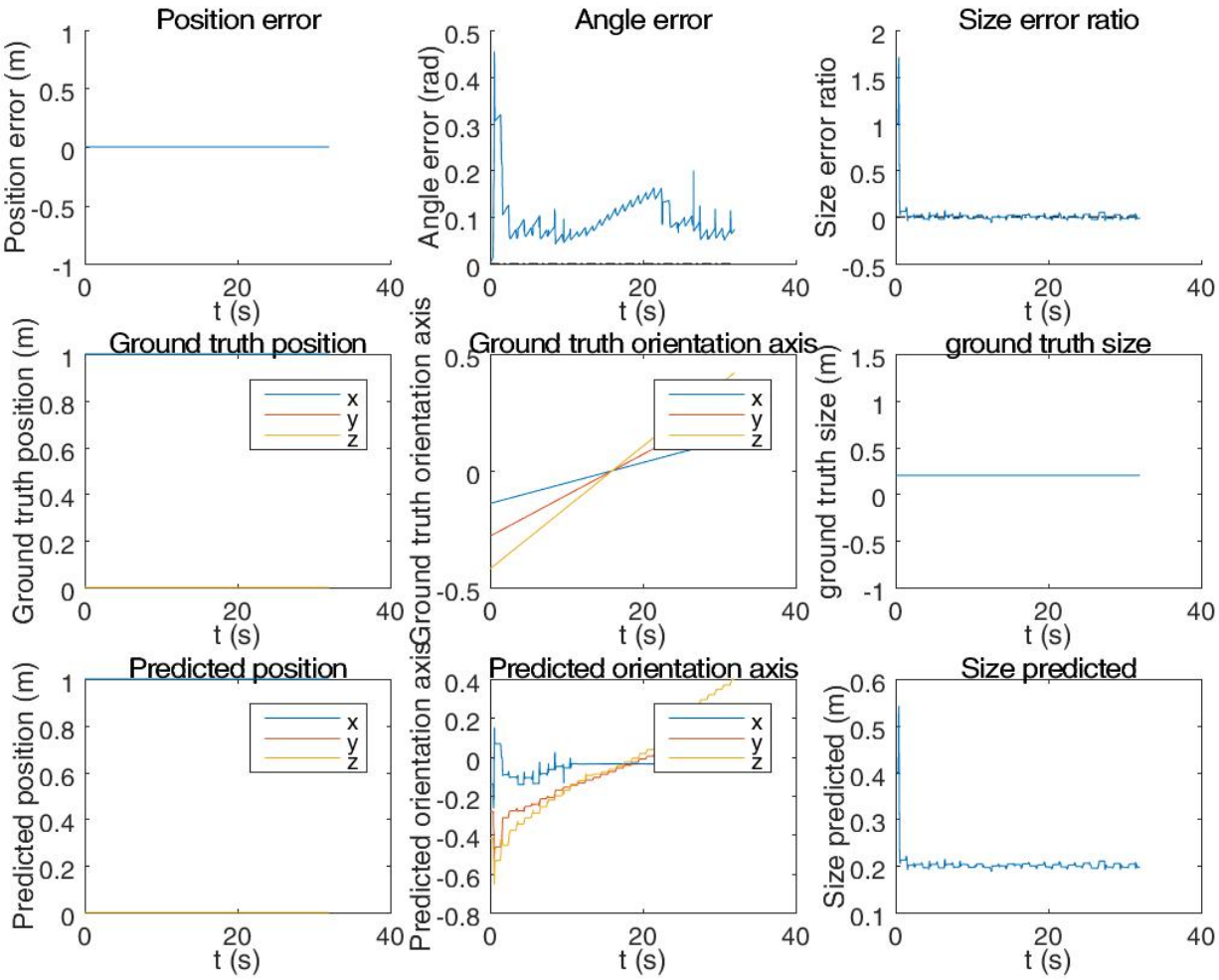


Figure 2.20: rotating with no noise, 3 faces visible - orientation (via screw) and size correction

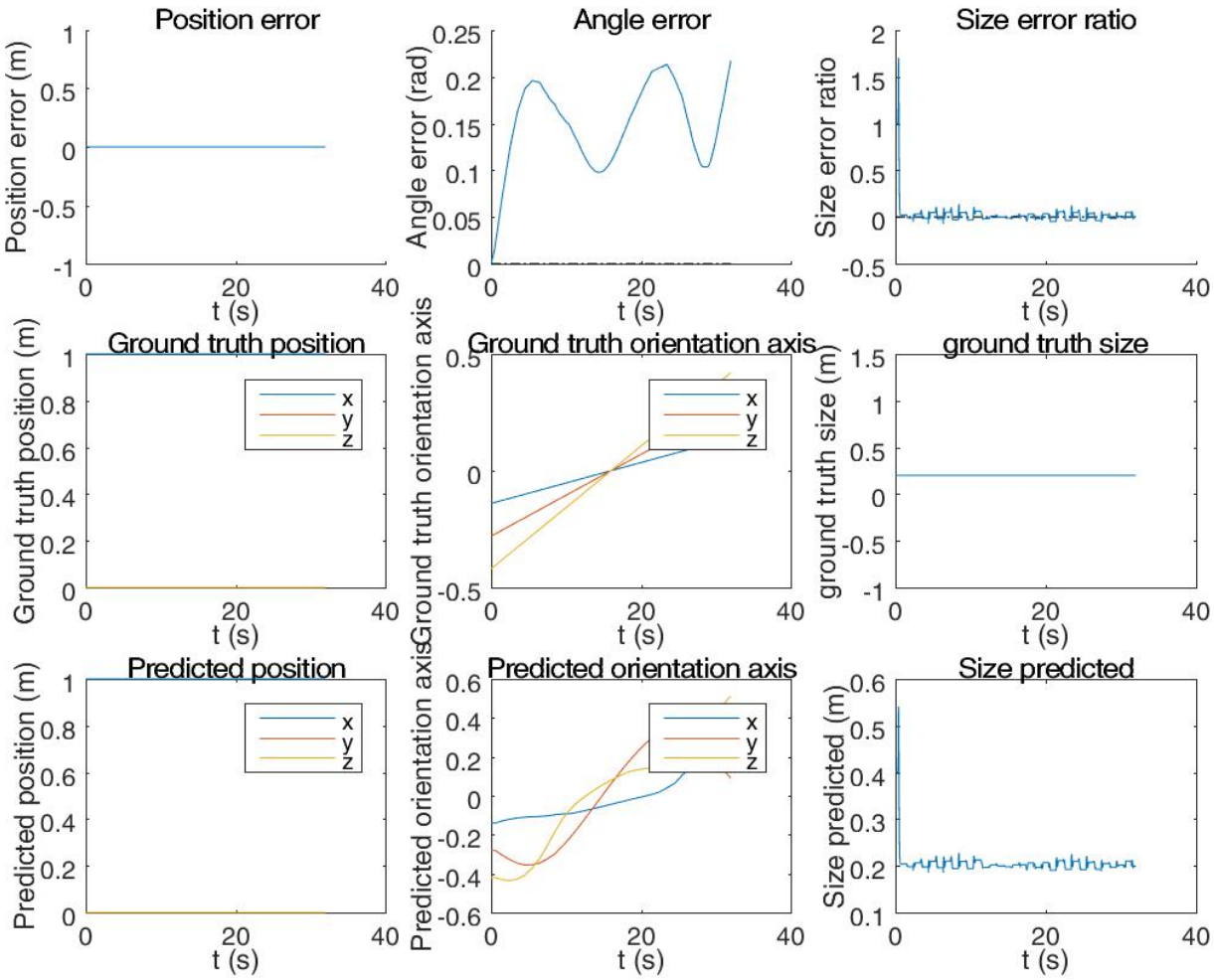


Figure 2.21: rotating with no noise, 3 faces visible - orientation (via twist) and size correction

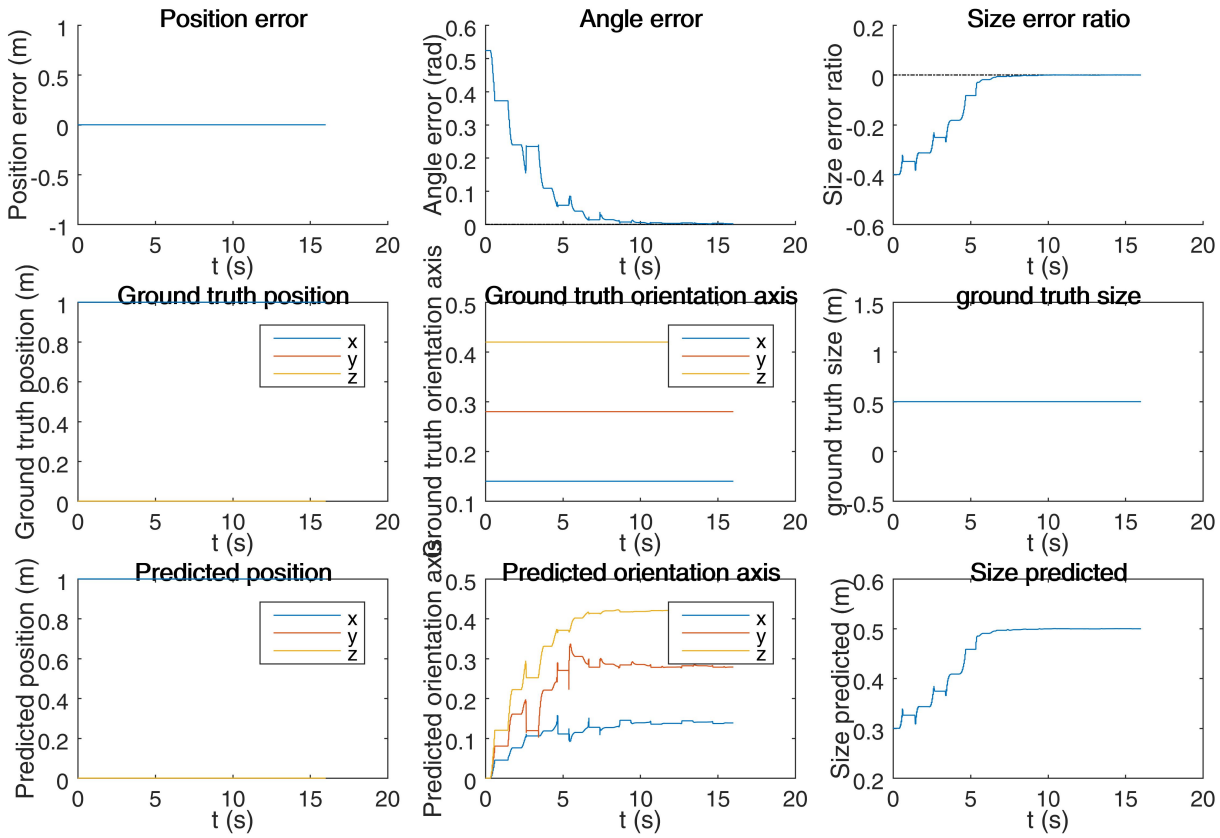


Figure 2.22: Tetrahedron: orientation & size correction

2.2.6 effect of initial conditions

stationary orientation and size, with noise. plot time to within 5% for range of initial angle and size error. see what happens

2.2.7 discussion

Size most robust, then orientation, then position. When combining, performance limited by less robust update.

Observer does not rely on cube shape. Replaced cube with tetrahedron, still works. Figure 2.22.

Seems to be a good approach. To improve, need to fix position. Also, overshoot when adjusting via twist matrix. Twist needs very fine tuning. Instead, do some combination of screw and twist to maximise speed, minimise overshoot.

Chapter 3

Experiment

Sensor:
Hokuyo UBG 04-LX - 2D scanning laser rangefinder. Measures range

Robot arm:
Kinova Jaco - can program to move with and record pose

Target object:
0.1×0.1 m MDF cube, spray painted matte white

1. Collected measurements to model sensor noise for simulation
2. Collected measurements of moving cube + measured ground truth cube state - for testing observer with real world conditions

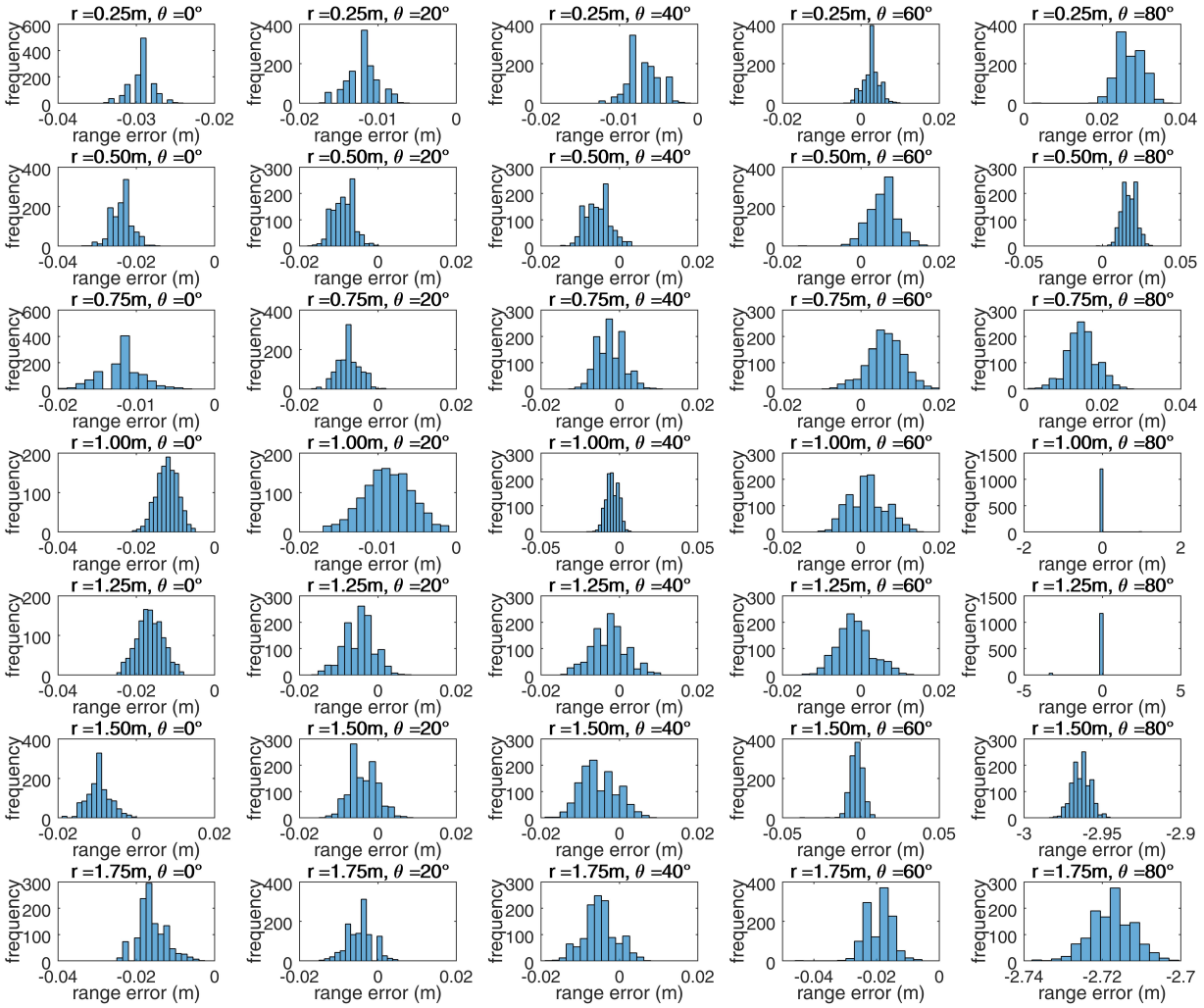


Figure 3.1: $r_{error}(r, \theta)$ approximately normally distributed

3.1 Sensor Noise Characterisation

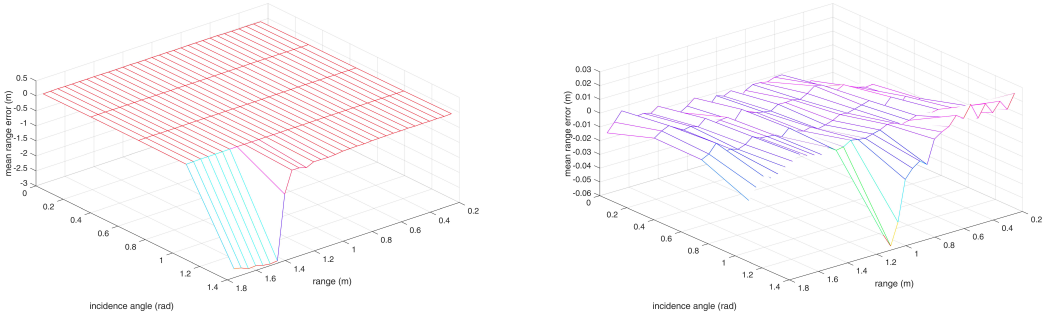
3.1.1 Setup

physical setup:
measurement configurations:
-5cm increments from 0.25-1.75m
-vary incidence angle: 0,20,40,60,80 degrees

3.1.2 Results

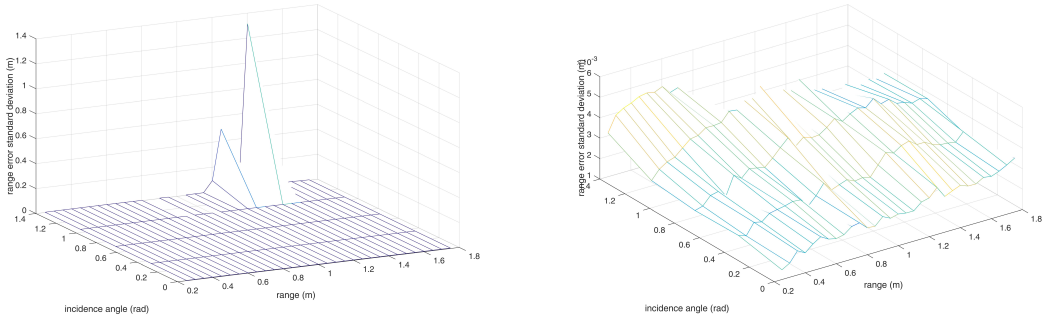
Error distribution: Histograms (Figure 3.1) - range error approximately normally distributed:

Mean range error as function of (range,incidence angle) - Figure 3.2



(a) (b)

Figure 3.2: mean range error vs (r, θ). (a) large error at high angles and range, (b) overall shape



(a) (b)

Figure 3.3: range error σ vs (r, θ). (a) outliers/large std dev at high angles and range, (b) overall shape

Std dev range error as function of (range,incidence angle) - Figure 3.3

Fitted 4th degree (in x and y) polynomials to data points - Figure 3.4. INCLUDE FIT DATA

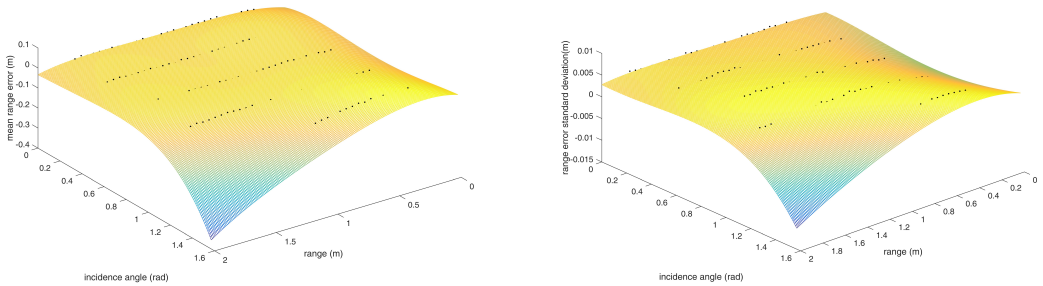
Noise Model:

$$r_{error} = r + \mathcal{N}(\mu, \sigma) \tag{3.1}$$

$$\begin{aligned} \mu =& a_{00} + a_{10}r + a_{01}\theta + a_{20}r^2 + a_{11}r\theta + a_{02}\theta^2 \\ &+ a_{30}r^3 + a_{21}r^2\theta + a_{12}r\theta^2 + a_{03}\theta^3 + a_{40}r^4 \\ &+ a_{31}r^3\theta + a_{22}r^2\theta^2 + a_{13}r\theta^3 + a_{04}\theta^4 \end{aligned} \tag{3.2}$$

$$\begin{aligned} \sigma =& b_{00} + b_{10}r + b_{01}\theta + b_{20}r^2 + b_{11}r\theta + b_{02}\theta^2 \\ &+ b_{30}r^3 + b_{21}r^2\theta + b_{12}r\theta^2 + b_{03}\theta^3 + a_{40}r^4 \\ &+ b_{31}r^3\theta + b_{22}r^2\theta^2 + b_{13}r\theta^3 + b_{04}\theta^4 \end{aligned} \tag{3.3}$$

*outliers ignored in model. if angle $>$ 75 deg and range $>$ 0.8m, range measurement = NaN. EXPLANATION: as angle increases, measured range becomes less than ground truth - due to beam width. part of beam hits part of object that is closer due to angle. However, as angle increases further, not enough light returns to make measurements. At high ranges and angles, measurement is not of object, but due to reflections that take much longer. Measurement ends up being near maximum range or even INF. Instead, will set as NaN - no value returned.



(a) (b)

Figure 3.4: polynomials fitted to range error mean & standard deviation data points to model noise

Table 3.1: a_{ij} coefficients

	j_0	j_1	j_2	j_3	j_4
i_0	-0.06529	0.2126	-0.533	0.4629	-0.1223
i_1	0.2024	-0.1906	0.4006	-0.1791	0
i_2	-0.3074	0.0228	-0.0716	0	0
i_3	0.2053	0.01455	0	0	0
i_4	-0.04912	0	0	0	0

Table 3.2: b_{ij} coefficients

	j_0	j_1	j_2	j_3	j_4
i_0	0.001242	0.2126	-0.01128	0.01162	-0.002746
i_1	0.00352	0.006146	0.01021	-0.007316	0
i_2	-0.005138	-0.00626	-0.0005068	0	0
i_3	0.004067	0.001337	0	0	0
i_4	-0.001092	0	0	0	0

Coefficients in tables 3.1 and 3.2

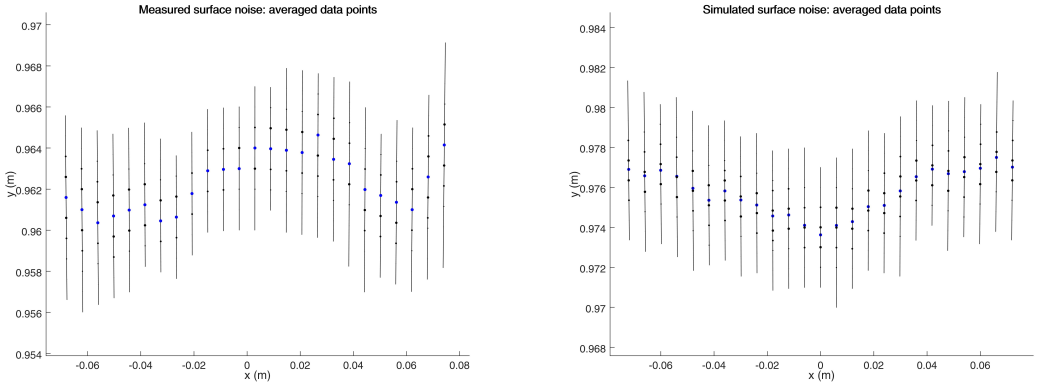
Surface noise: random walk model
have observed error mostly independent of range and angle - seems to be some compensation performed by sensor to get straight lines - range across smooth surfaces that should be appear as straight lines varies regularly. Could be surface properties of objects, though error is larger than expected in this case. Profile of error sinusoidal (regular-ish peaks and valleys), peaks = 5mm deep & 50mm wide approximately

add random walk to each scan

$$error = a \sum_{n=1}^{n_{Steps}} -1 + 2 \lfloor \mathcal{R} \rfloor \tag{3.4}$$

where $\mathcal{R}$ is a random variable following a uniform distribution on [0,1] Doesn't work too well over long surfaces (1m) - can get too large sometimes, but fits measured data for small objects like cube

Figure 3.5: comparison of simulated & measured surface noise:



(a) (b)

Figure 3.5: Comparision of (a) measured and (b) simulated surface noise

3.2 Testing Data Collection

3.2.1 Setup

physical setup
configurations/motions:
-stationary
-rotating
-translating
(all above with 1,2,3 faces visible to sensor)

arm forward kinematics $\rightarrow$ cube pose
estimate sensor angle with horizontal with wall calibration data

3.2.2 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.