フレームシーケンシャル方式の観察者の位置に連動して変化する立体視映像の作成

岐阜大学　工学部　電気電子・情報工学科　木島研究室
野倉大輝　脇田航平　前田優樹　小木曽直輝　藤田健司

目次

第 1 章　はじめに

本取り組みを理解するための基本的な知識を解説した後，本取り組みの目的を述べる．

1.1　人間が奥行きの知覚を可能としている理由

私たちが奥行きの知覚を可能としている理由は，2 つの眼が離れていることと，脳での処理によるためである．右眼と左眼の場所は離れており，見える像も異なる．したがって，眼からは異なる 2 次元像が 2 つ得られる．しかし，私たちが知覚する像は 1 つであり，見えるモノに対して奥行きを知覚することが可能である．それは，人間の右眼・左眼それぞれで得られた情報を脳で処理して 1 つの像にするためである．異なる 2 つの 2 次元像から，モノの距離を一意に特定でき，私たちは，奥行きを知覚することが可能である．

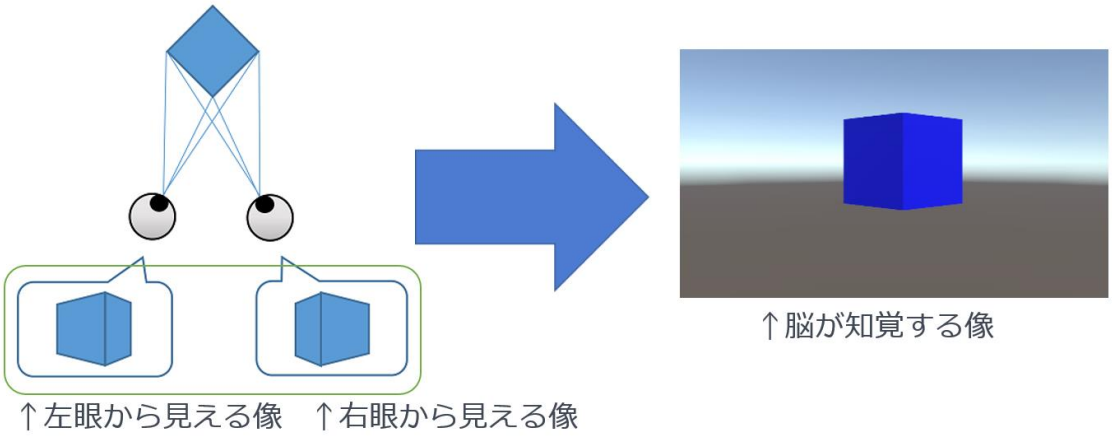

図 1.1　人間が奥行きの知覚を可能としている理由
（[1]を加工）

1.2　フレームシーケンシャル方式

フレームシーケンシャル方式とは，映像を 3D に見る方法の一つで，3D 機能のあるディスプレイとディスプレイに同期した特殊なメガネ（アクティブシャッターメガネ）（図 1.2）を用いる．ディスプレイは，左眼用の映像，右眼用の映像を超高速で切り替えて表示する．メガネは，ディスプレイに合わせて，左眼用の映像が表示されているときは，右眼のシャッターを閉じ，右眼の映像が表示されているときは，左眼のシャッターを閉じる（図 1.3）．これにより，左眼に左眼用の映像，右眼に右眼用の映像を見せることが可能であるため，ディスプレイに表示されている映像から現実世界のような奥行きを知覚することが可能である．

図 1.2　アクティブシャッターメガネ（[2]より引用）

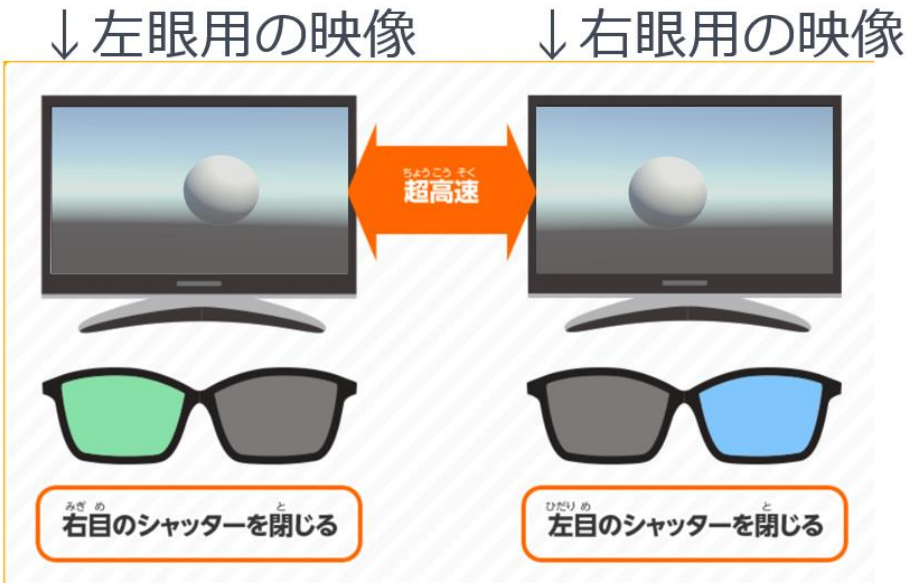

図 1.3　フレームシーケンシャル方式（[3]を加工）

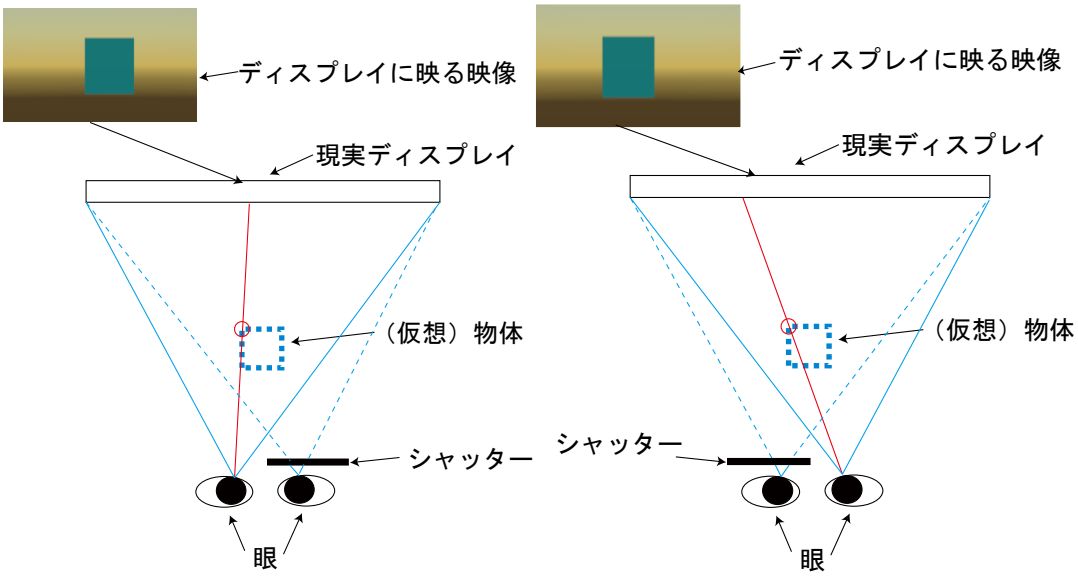

図 1.4　3D 機能のあるディスプレイとアクティブシャッターメガネを用いた立体視

1.3　自己運動視

物体の位置が不変であり，観察者の位置が可変であるとき，観察者の位置によって物体の見え方や見える場所が異なる．このことを自己運動視と呼ぶ（図 1.5）．

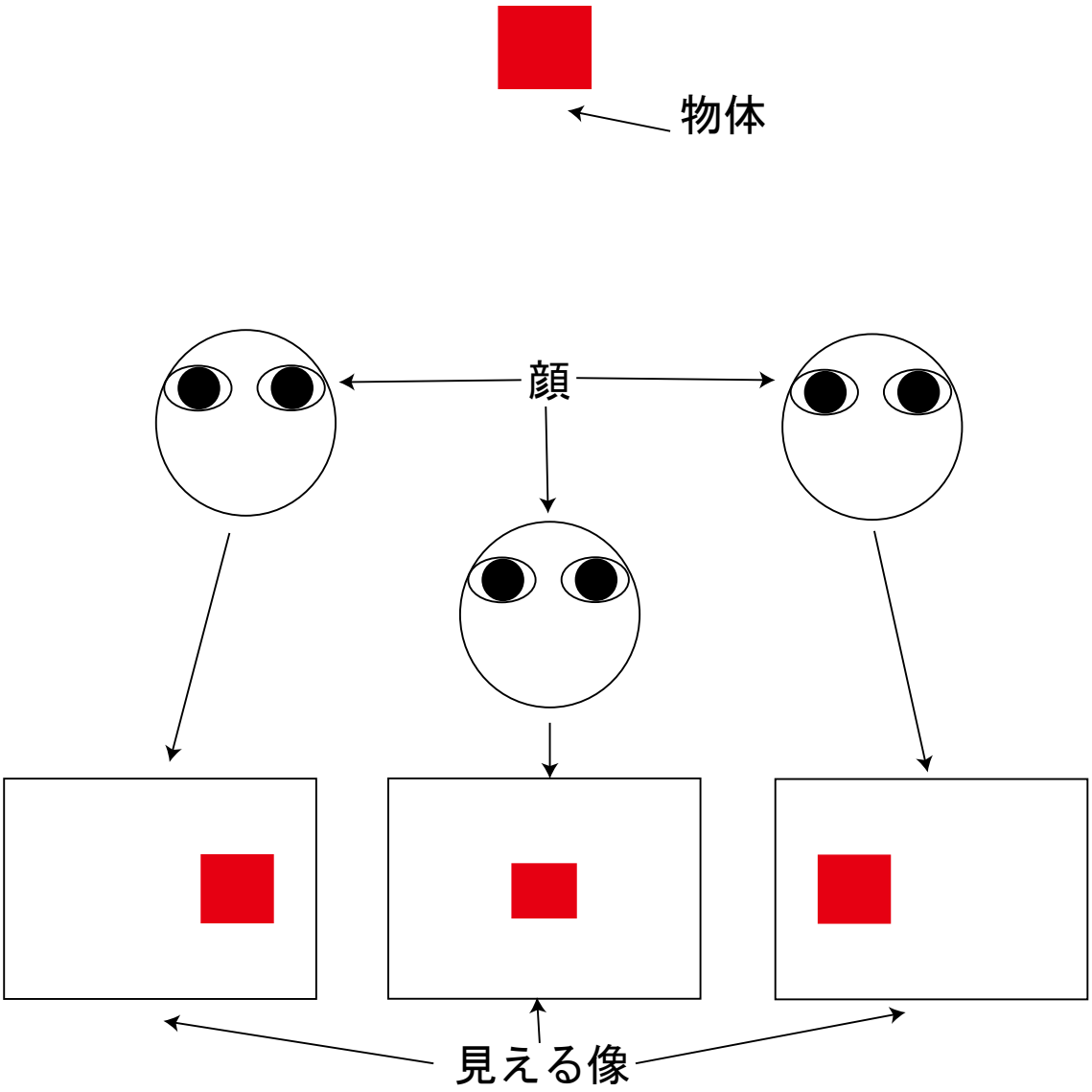

図 1.5　自己運動視

1.4　本取り組みの目標

本取り組みでは，1.2 節で述べた 3D 機能のあるディスプレイとアクティブシャッターメガネを利用して，

・何処から見ても，同じ場所に物体が見える（自己運動視（1.3 節）が可能）.

・竹籤で同じ大きさの物体を作成し，手を伸ばすと物体がそこにあるように知覚する.

ということを目標とする.

仕様について，より詳しく書くと以下のようになる.

1. ディスプレイは左眼用と右眼用の画像を高速で切り替えて表示する.
2. 人がディスプレイと同期したアクティブシャッターメガネを付けてディスプレイを見ると任意の場所に物体が存在するように見える.
3. この時，人がどの場所から見ても（ただし，見るときは正面を見る）見える物体の位置は変わらない.
4. 実際には物体は存在しないが，見える場所に竹籤製の同じ大きさの物体がある為，手を伸ばすと触覚でも物体を知覚可能である.

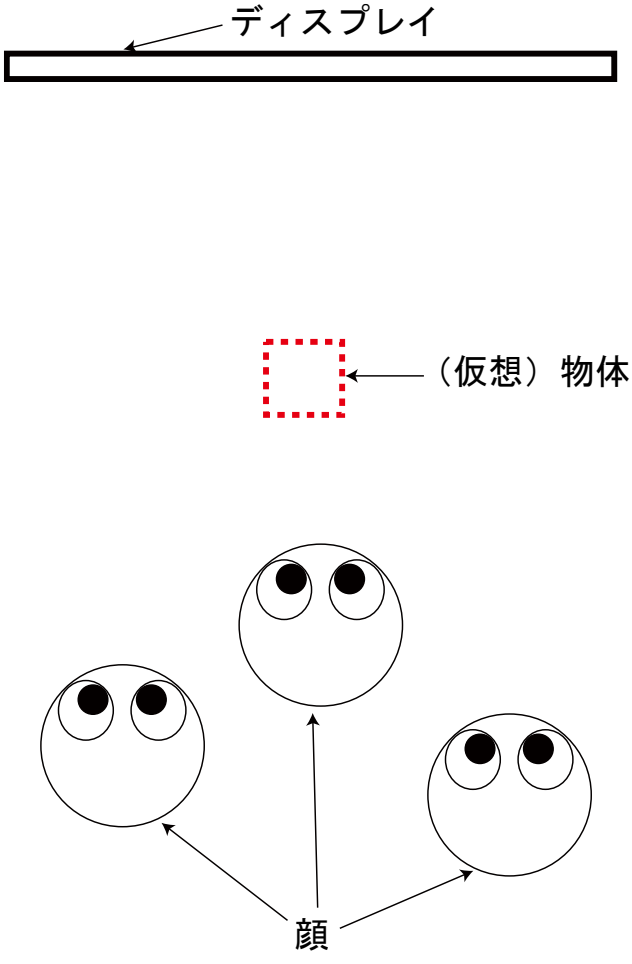

図 1.6　本取り組みの目標イメージ図

第 2 章　作成方法

本取り組みの実現のために必要な要素は 2 つあり，
・立体視可能な映像の作成
・ユーザーの視点位置の取得
である．初めに，作成で使用したものを書き，この 2 つについてそれぞれ記述する．

2.1　使用したもの

本取り組みで使用した主なものは以下のとおりである．
・3D 機能のあるディスプレイ：
Panasonic 製　地上・BS・110 度 CS デジタルハイビジョンプラズマテレビ TH-P50VT2[4]

・アクティブシャッターメガネ：
Panasonic 製　3D グラス TY-EW3D2LW[5]

・映像の作成：
Unity 2020.2.5f1[6]

・位置の取得：
HTC VIVE

SteamVR Plugin[7]

・PC

・HMD のヘッドバンド

2.2　立体視可能な映像の作成

2.2.1　現実世界と仮想世界

私たちは，立体視可能な映像の作成に，仮想世界である物質を撮影した映像を，現実世界でディスプレイ（3D 機能のあるテレビ）を通して見る，ということで実現した．この時，撮影する映像は右眼用と左眼用の 2 つで，アクティブシャッターメガネを通して，2 つの眼に別の映像を見せている．仮想世界の作成と映像の撮影は，Unity を用いた．使用した版は，2020.2.5f1 である．

（※Unity は Unity Technologies 社が 2004 年に開発したゲームエンジンである．スマートフォンゲームの多くは，Unity のゲームエンジンが使用されている．）

現実世界には，重要な要素として，3D 映像を表示させるディスプレイとユーザーの眼がある．一方，仮想世界には，ユーザーに知覚させたい仮想物体のほかに，現実世界でユーザーの眼に当たるカメラ 2 台と現実世界のディスプレイに対応する仮想ディスプレイを設置する．この時，対応する要素は，それぞれの世界で相対的に同じ位置，大きさで設置する．すると現実世界ではユーザーは，仮想物体を見ることができる．

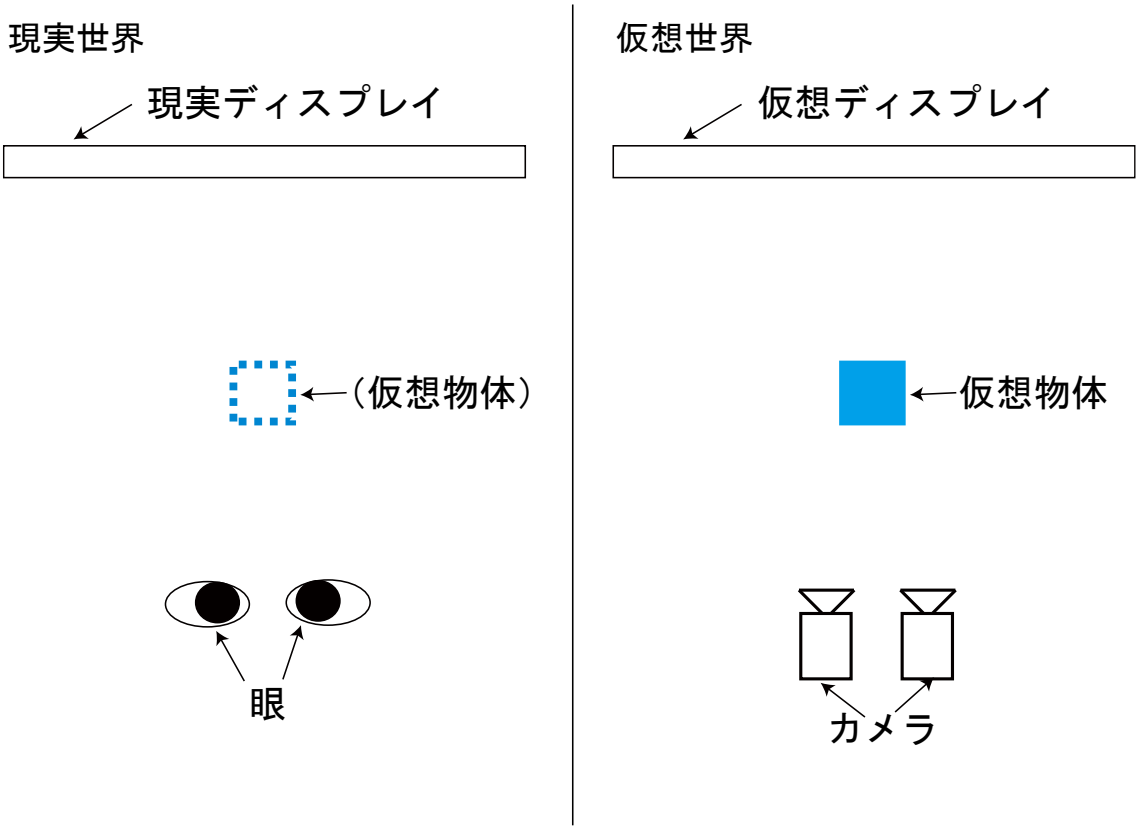

図 2.1　現実世界と仮想世界（左：現実世界　右：仮想世界）

※現実世界では，物体が実際に存在するわけでない．そこにあるように見えるという意味である．

仮想世界から眼までの情報（光線）の伝達の仕方をみてみると（図 2.2，図 2.3），

1.（仮想世界で）「カメラ→仮想ディスプレイ」の光線群

（※実際は，（仮想世界で）「外→仮想ディスプレイ→カメラ」の光線群であるが，本質的には同じであるため，ここでは説明の便宜上このように考える）

2.（仮想世界でカメラから仮想ディスプレイを通る光線を撮影した）2 次元画像

（※実際は，カメラのセンサーが写す 2 次元画像であるが，本質的には同じであるため，ここでは説明の便宜上このように考える）

3.（現実世界で）「ディスプレイ→ユーザーの眼」の光線群

となる．尚，仮想世界（Unity）のカメラは，ピンホールカメラとする．

本質的には，図 2.2 のように，仮想世界でカメラから仮想ディスプレイを通る光線を，現実世界で現実ディスプレイから方向を逆にして光線を出しているのと同じである．

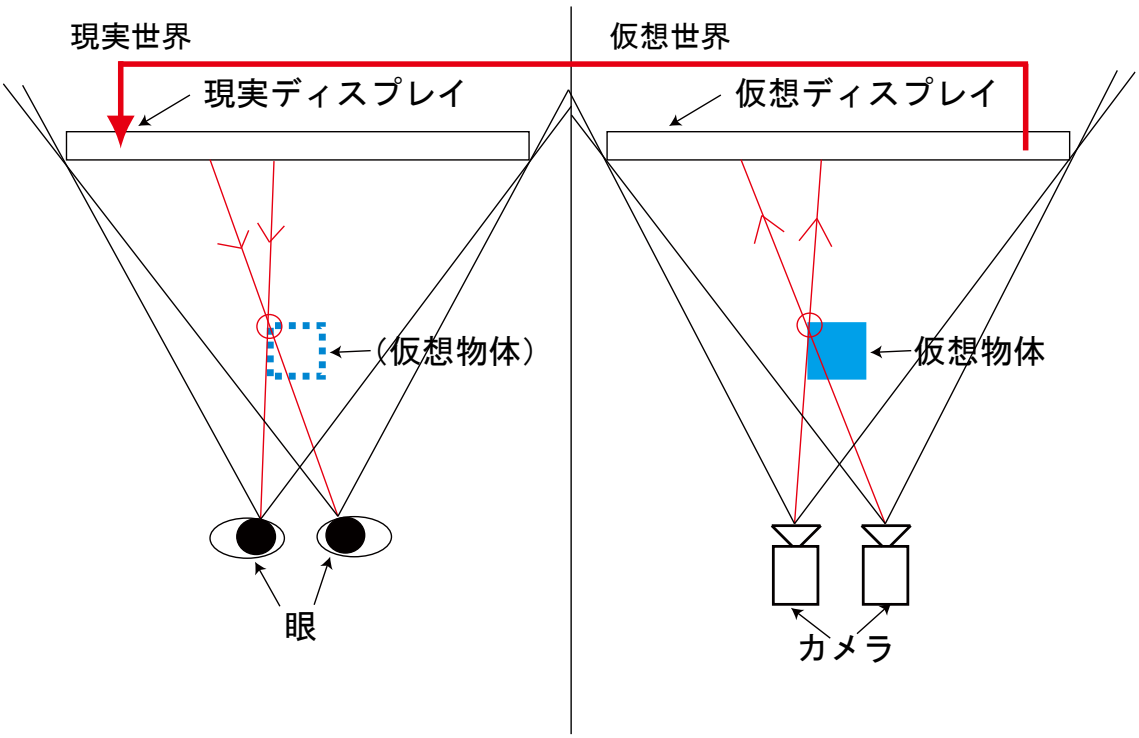

図 2.2　光線の伝達

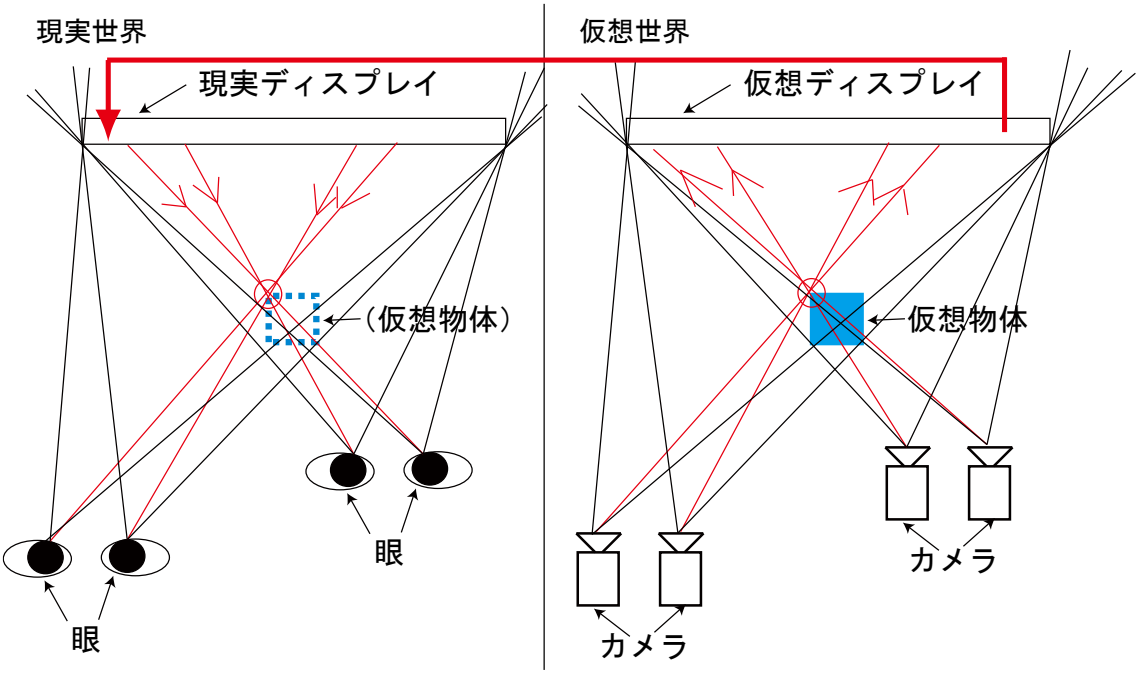

図 2.3　光線の伝達（自己運動視）

2.2.2　Unity におけるカメラの設定

本取り組みでは，どこから見ても見える物体の位置は変わらないことを目標としている．目標を達成するためには，ユーザーの眼と仮想世界でのカメラの設定を同じようにする必要がある．図 2.4 のように，現実世界においてユーザーがディスプレイの中央正面にいない時，仮想世界のカメラは，ディスプレイの両端をとらえるように傾くのではなく，正面を向いていなければならない（図 2.5）．これは，カメラのセンサーはカメラの向きに対して垂直であり，そのレンダリングされる領域を切り取る断面はセンサーに対して並行であるが，そのレンダリング画像をディスプレイ上に映し出すという設計上，センサーはディスプレイと並行していなければならないためである．よってカメラがディスプレイと並行であることかつ，カメラが映す領域はディスプレイを通るようにするという 2 点を同時に考えなければならない．

これを実現するために Unity において Lens Shift と FOV（Field of View）というものがある．以下にこの 2 つの設定について説明する．

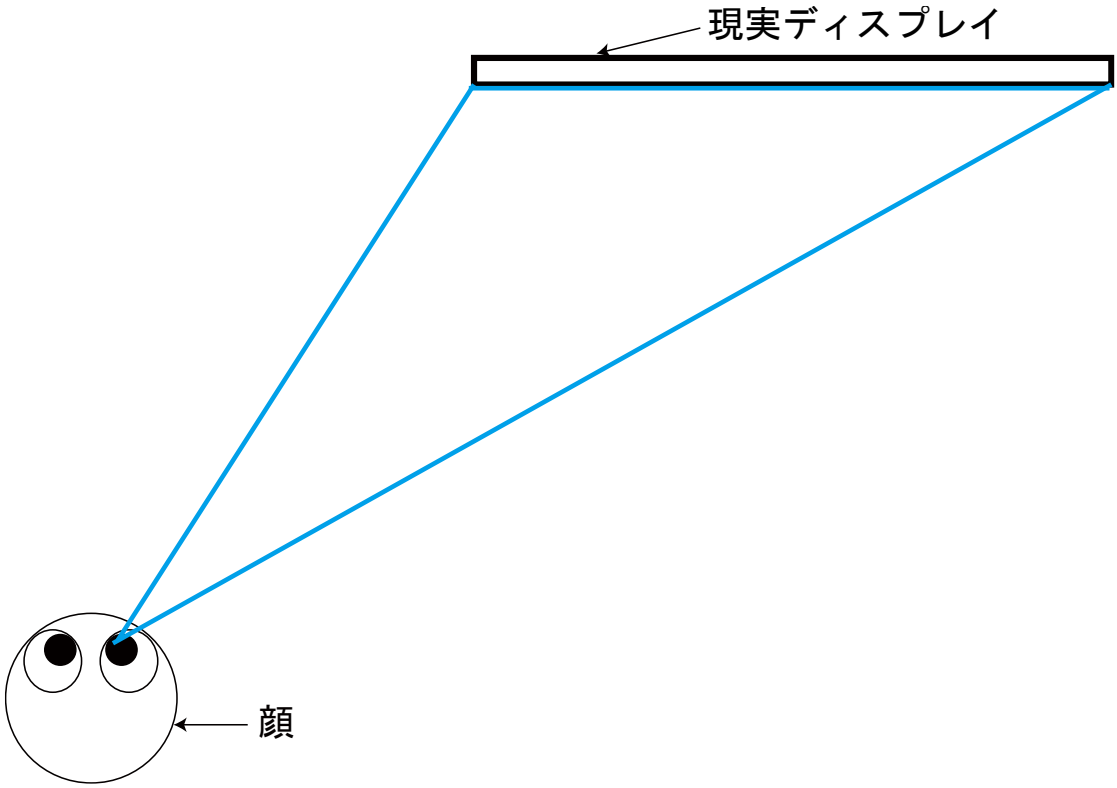

図 2.4　現実世界においてディスプレイから離れて見たとき

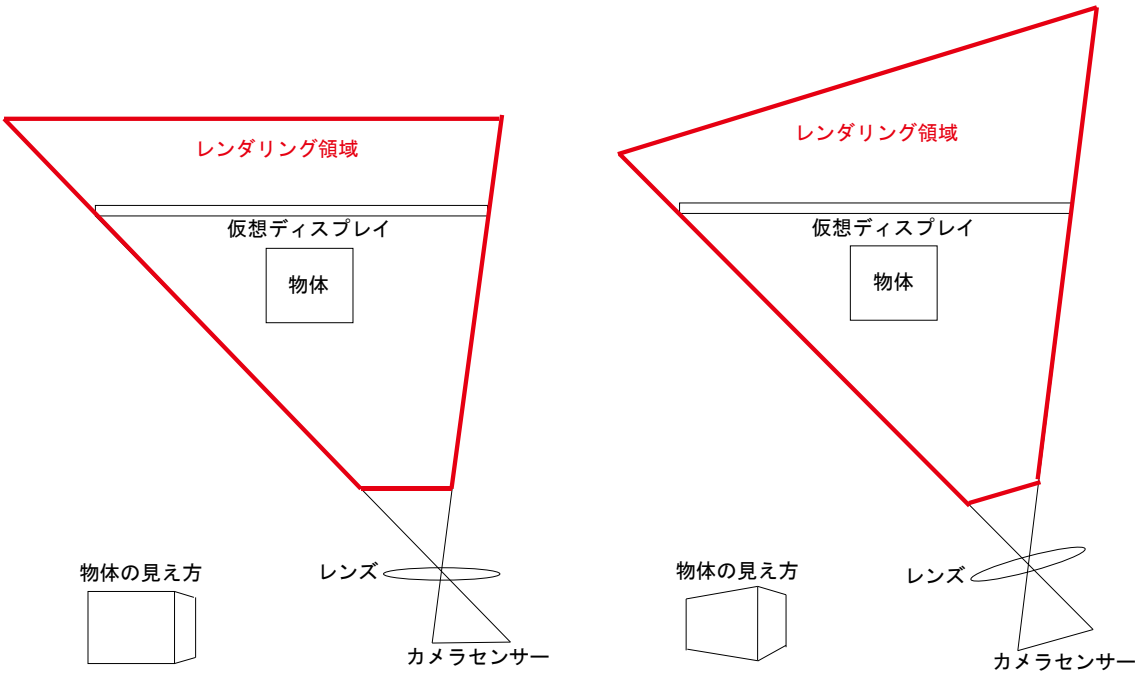

図 2.5　仮想世界において図 2.4 に対応したカメラ（左：正解 右：誤り）

2.2.2.1　Lens Shift

Unity において，投影面を上下左右に移動させるパラメータとして，Lens Shift がある．図 2.6 は，Lens Shift を変化させた例である．

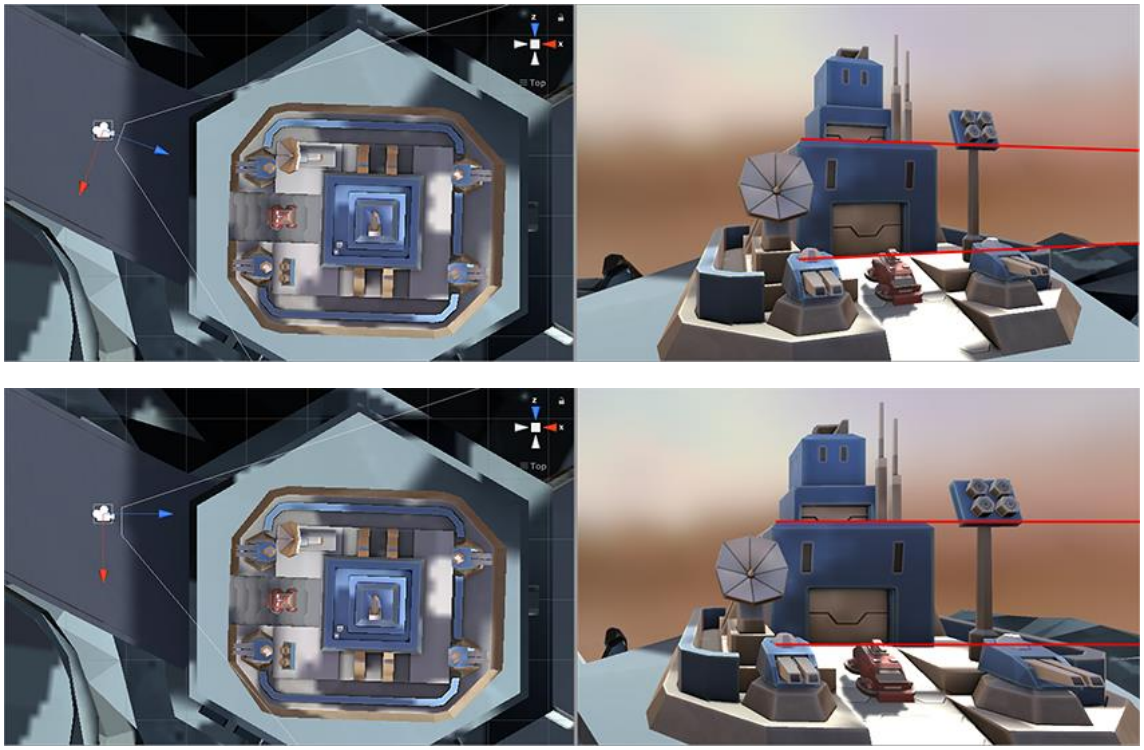

図 2.6　Lens Shift の有無の撮影画像（上：Lens Shift あり　下：Lens Shift なし）
（[8]より引用）

公式ドキュメント[9]によると，Lens Shift の値は，カメラセンサーのサイズの倍数だけ投影面が移動する（図 2.7）．また，パラメータは x 軸，y 軸の 2 つある．

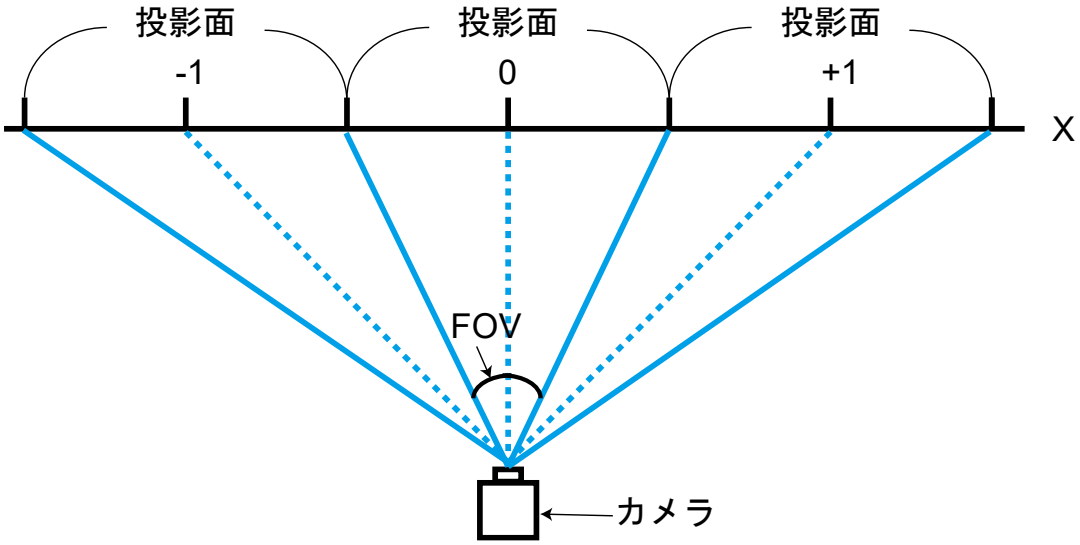

図 2.7　Lens Shift

2.2.2.2　FOV（Field of View）

Unity においてカメラの視野角のパラメータとして，FOV（Field of View）がある．そして FOV の軸は垂直方向か水平方向かを選択することができ，片方の軸の視野角が決まれば，もう一方の軸の視野角もカメラセンサーのサイズ（撮影範囲の比）によって一意に決まる（図 2.8）．単位は，[degree]である．

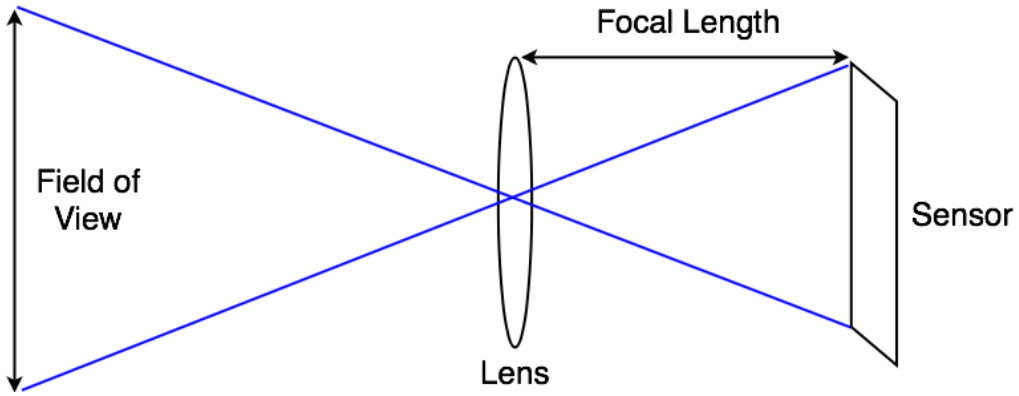

図 2.8　FOV とカメラセンサーのサイズの関係
（[8]より引用）

2.2.2.3　Unity におけるカメラの設定

初めに，カメラセンサーのサイズを仮想ディスプレイの比と同じにする．

次に，カメラの位置に合わせて先ほど説明した Lens Shift と FOV を適切に変化させ，カメラの撮影領域（Unity における視錐台）が，図 2.5 のように仮想ディスプレイの四隅がちょうど入るようにする．

FOV は，カメラの正面にディスプレイがあると考えた時の角度となる（図 2.9）．

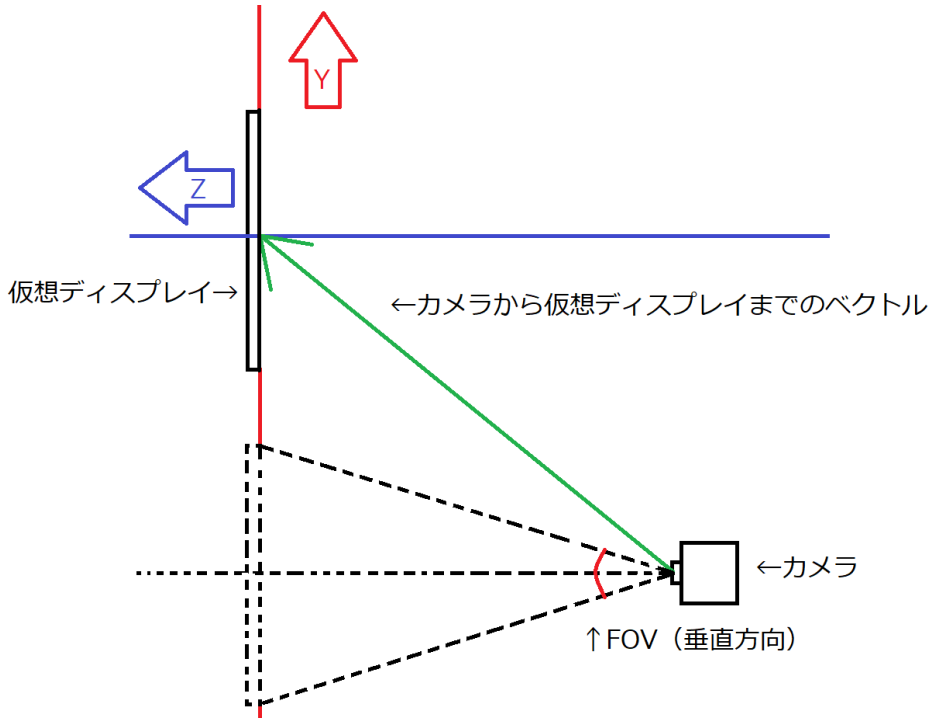

図 2.9　カメラの位置が変化したときの仮想世界の座標平面（yz 平面）

この時 FOV は以下の式で求めることが可能である．ただし，逆三角関数の計算をする際，Math ライブラリの Math.atan()といった関数を用いるのであれば，返り値の単位は，[rad]である．2.2.2.2 節で述べたように，FOV の単位は，[degree]であるため，式 2.1 の左辺に[rad]から[degree]への変換を加えなければならない．

$$\mathrm{FOV}\left(\text{垂直方向}\right) = 2 \times \arctan\left(\frac{\frac{\textit{仮想ディスプレイの高さ}}{2}}{\textit{カメラから仮想ディスプレイまでのベクトルのz成分}} \right)$$

式 2.1　FOV（垂直方向）を求める式

Lens Shift の x 軸は, カメラから仮想ディスプレイまでのベクトルの x 成分(被除数)と仮想ディスプレイの横幅（除数）の商である．一方で，y 軸は，カメラから仮想ディスプレイまでのベクトルの y 成分（被除数）と仮想ディスプレイの縦幅（除数）の商である．式にすると以下の式 2.2，式 2.3 のようになる．

$$Lens\ Shift\left(x軸\right) = \frac{カメラから仮想ディスプレイまでのベクトルのx成分}{仮想ディスプレイの横幅}$$

式 2.2　Lens Shift（x 軸）を求める式

$$Lens\ Shift\left(y軸\right) = \frac{カメラから仮想ディスプレイまでのベクトルのy成分}{仮想ディスプレイの縦幅}$$

式 2.3　Lens Shift（y 軸）を求める式

実時間について，カメラの位置の変化に合わせて，FOV，Lens Shift の値をこのように変更すれば，カメラが移動しながらも，カメラは，正面から見たときの仮想ディスプレイを追った映像が得られる．

2.2.3　現実ディスプレイに表示する映像の作成

私たちが用いた 3D 機能のあるテレビの 3D 機能は，映像を左半分と右半分の 2 つに分け，その左半分・右半分の映像それぞれを横に引き伸ばして長さを 2 倍にし，交互に表示するというものである．そのため，仮想世界で左眼にあたるカメラで得られた映像，右眼にあたるカメラで得られた映像をそれぞれ横方向に関して半分にし，Unity の 3D オブジェクトの Panel に貼り付ける．その Panel を別のカメラが撮影し，得られた映像を現実ディスプレイに表示させる．これにより，3D 機能のあるテレビ（現実ディスプレイ）の 3D 機能を使うことで右眼・左眼の映像を現実ディスプレイが表示することが可能である（図 2.10）．

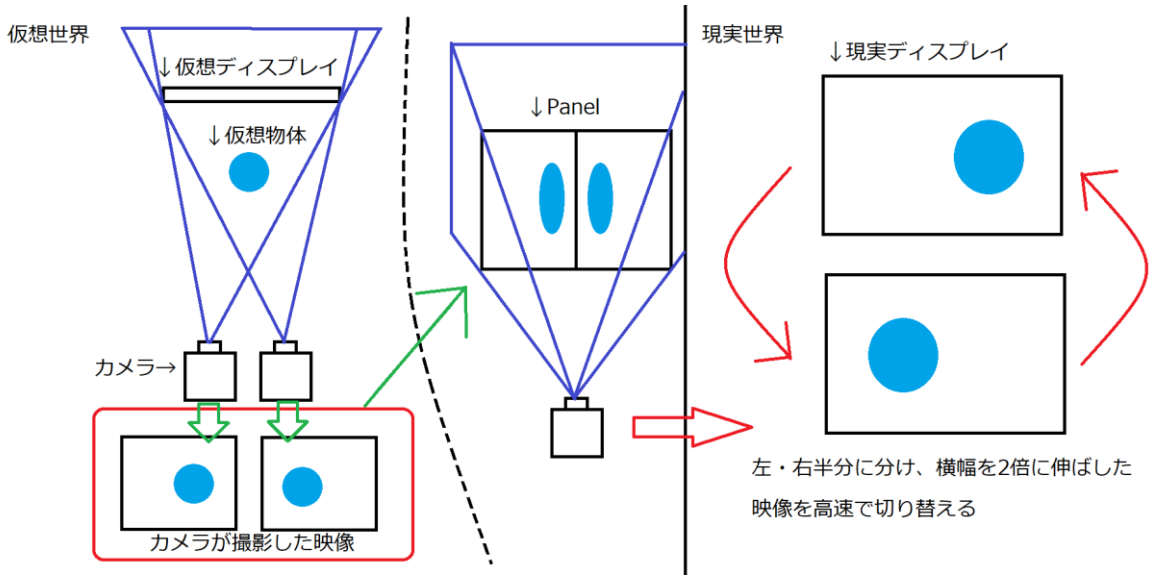

図 2.10　現実ディスプレイで表示する映像の作成

2.3　ユーザーの視点位置の取得

本取り組みにおいて，ユーザーの位置の取得は，HTC 社の HTC VIVE のコントローラー（図 2.11）を用いた．ここでは，始めに HTC VIVE の説明を行い，仮想世界で用いるための座標情報を得るために行ったことを説明する．尚，HTC VIVE のコントローラーの座標を取得する為に，SteamVR Plugin というアセットを用いた．

2.3.1　HTC VIVE

HTC VIVE（図 2.11）は，”全身での没入感”を生み出すことが可能である VR HMD である．2,160×1,200 [pixel]の高解像度と 90[fps]で描写されるハイスペックなヘッドセットに加え，二つの専用コントローラー，空間における正確な位置を全方位で追跡する二つのベースステーションにより，座った状態だけではなく，実際に空間を動きまわることが可能である VR を体験可能であるシステムである．

コントローラーには，ベースステーションが追跡に利用するセンサーが付いている．ベースステーションは赤外線を発射しており，ヘッドセットやコントローラーには受光部が用意されている．この光の到達時間や角度などから計算して位置を割り出している．

今回，実験に HTC VIVE を使用したのは，二つのベースステーションにより，コントローラーの空間座標を正確に取得することが可能であるためである．

図 2.11　HTC VIVE（左）と HTC VIVE コントローラー（右）

2.3.2　コントローラーとメガネの固定

瞳の位置を VIVE コントローラーで取得しつつ，3D メガネをかけてディスプレイを見る必要があるため, VIVE コントローラーと 3D メガネを一体にしたものを作成した．HMD のヘッドバンドのみを取り出したものに，養生テープやビニール紐を用いて，コントローラーとメガネを固定した．また，ヘッドバンドに充電器も取り付け，端子を用いてメガネと接続し，常にメガネに電力を供給するようにして，使いやすくしている（図 2.12）．

図 2.12　HMD コントローラー・３D メガネ一体型ヘッドギア

2.3.3　コントローラーから得られた座標の補正

コントローラーから得られる座標と現実世界における瞳の位置は同じでないため，補正を行う必要がある．行った補正は，コントローラーと両眼中心間の補正（2.3.3.1 節）とコントローラーと地面間の補正（2.3.3.2 節）の 2 つである．

2.3.3.1　コントローラーと両眼中心間の補正

コントローラーと瞳は離れているため，コントローラーと両眼の中心の間の距離を測定し，その分，コントローラーから得られた座標情報を補正した．コントローラーと両眼の中心の間の距離について，左右方向は，0[mm]，上下方向は，80[mm]，奥行き方向は，40[mm]だった（図 2.13）．

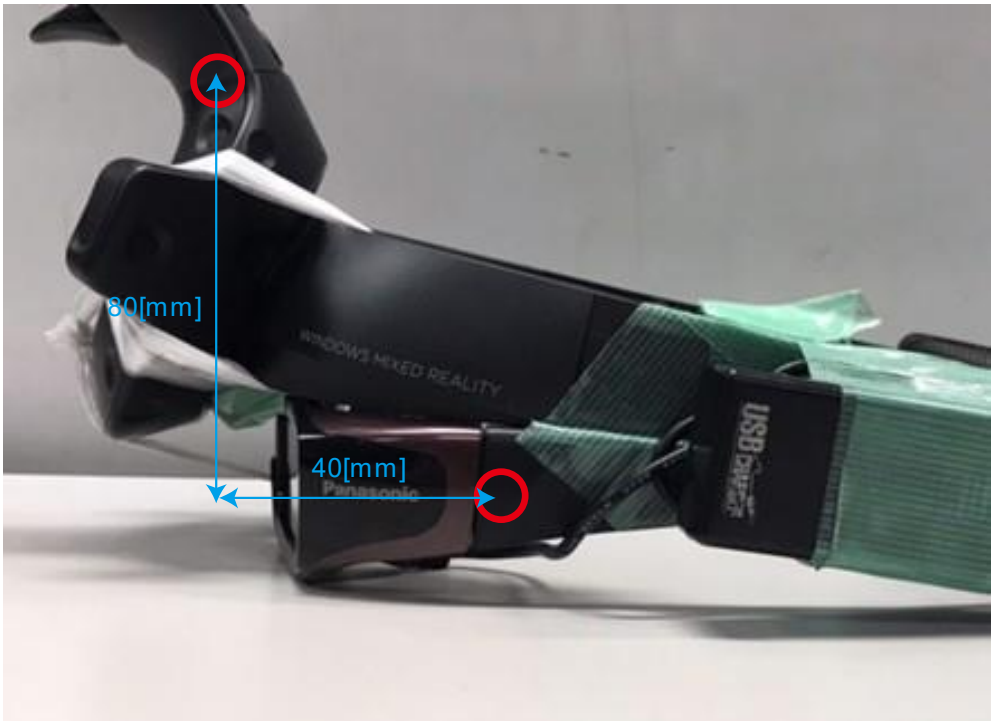

図 2.13　コントローラーと両眼中心の距離

＊コントローラーの座標の測定点の特定について

SteamVR Plugin[7]には，VIVE コントローラーのモデルがあり，アセットのコントローラーのモデルの座標の原点と，コントローラーの座標の測定点が等しいであるだろうと考えた．そのため，アセットのコントローラーのモデルの座標の原点を特定し，その点をコントローラーの座標の測定点とした．

Unity で，コントローラーのモデルと同じ階層に x 軸，y 軸，z 軸にそれぞれに棒を取り付け，その交点を調べた（図 2.14）．この交点に当たる部分を測定点とした．

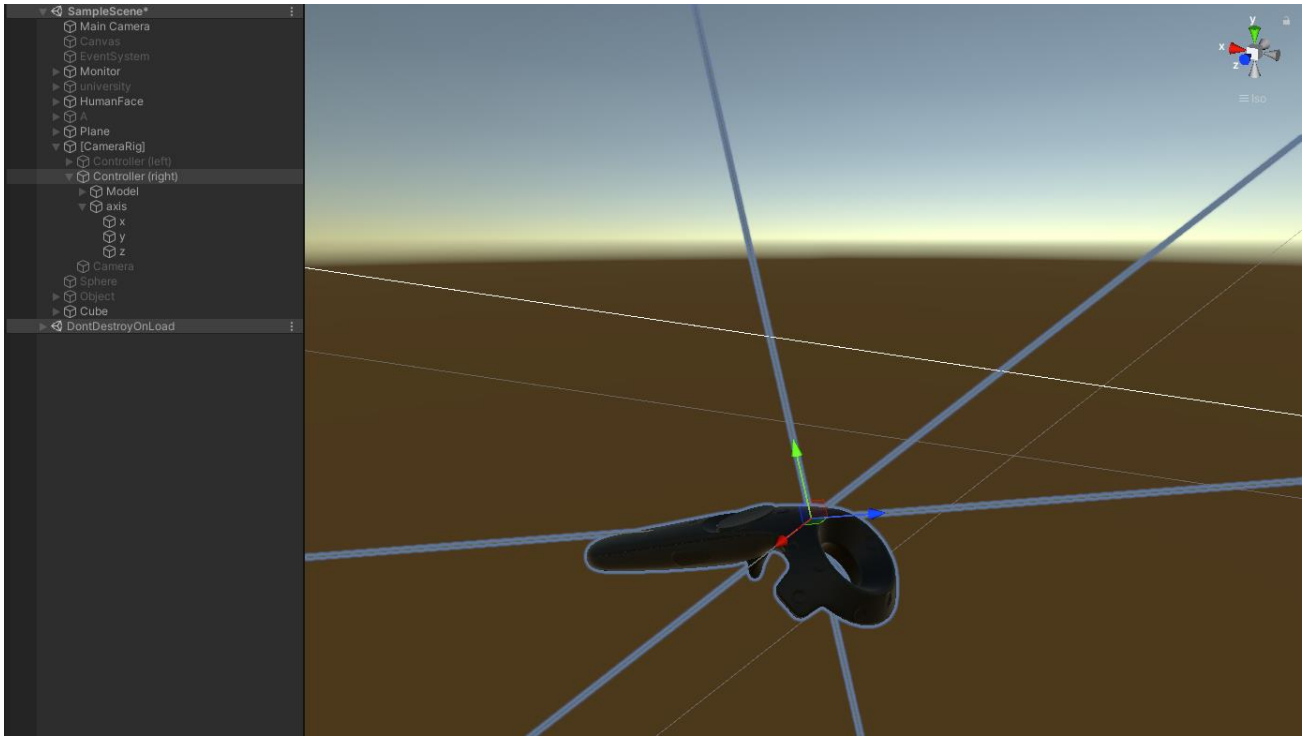

図 2.14　VIVE コントローラーのモデルの座標の原点の特定

2.3.3.2　コントローラーと地面間の補正

コントローラーから得られる座標情報を確認してみたところ，y 軸（上下方向）において，相対的な座標（ある位置からある位置までの座標の差）は正しかった．しかし，絶対的な座標（その位置の座標）が異なっており，床の位置の y 軸方向の座標が 0 でなかった．そのため，コントローラーを床に置いたときの y 軸の座標の値を，コントローラーから得られた y 軸の座標の値から引くことによって補正を行った．

2.3.3.3 総合的なコントローラーの補正

この節では，VIVE コントローラーから得られた座標からユーザーの視点位置を求めるために行った補正を合わせて，図や式を用いながら総合的な補正として説明する．

VIVE のプレイエリアの設定方法を用いて四隅の頂点を決めることで現実世界の原点を設定した．この原点からディスプレイの中心までの x, y, z 方向の距離を計測し，これと縮尺を合わせた仮想世界を作ることで現実世界と仮想世界の座標系を一致するようにした（図 2.15）．しかし，今回の実験の際に現実世界で確保したプレイエリアの地面の高さと仮想世界での高さ方向 0 の地点との間にずれがあった（2.3.3.2 節参照）ため，これを補正した．

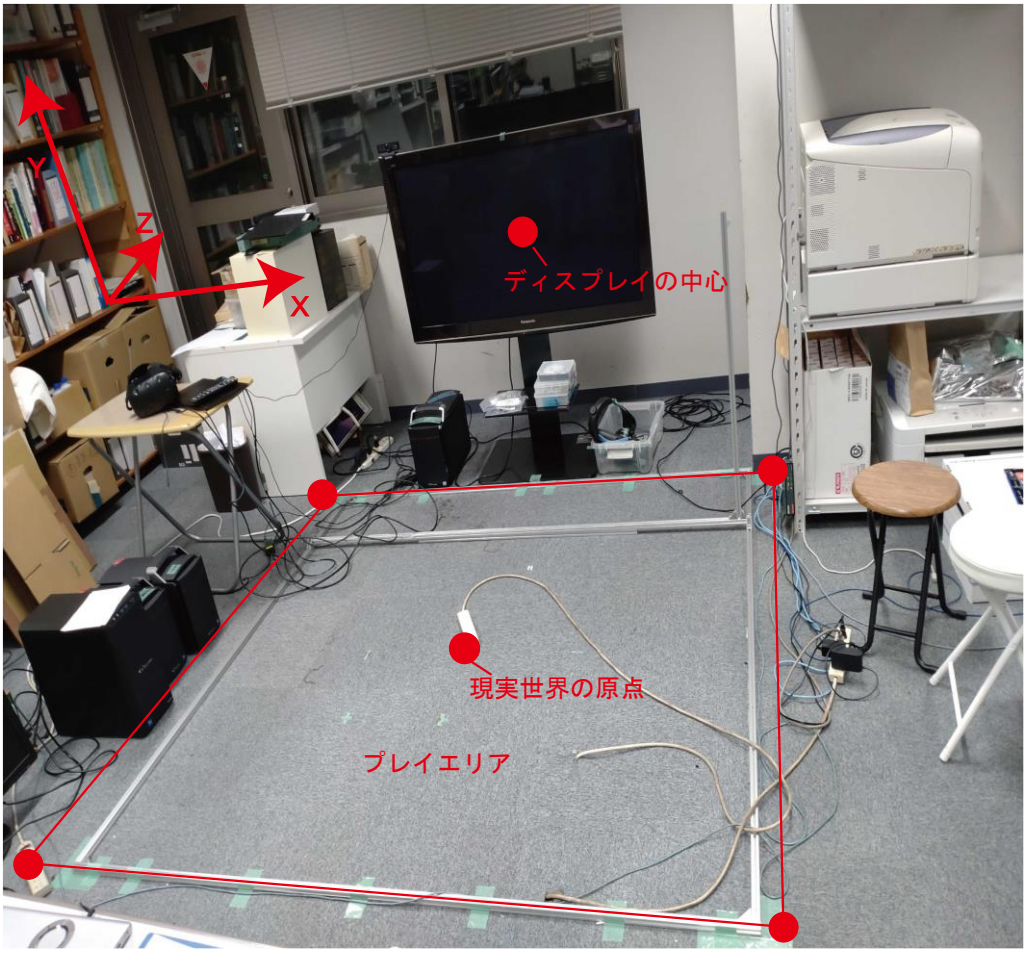

図 2.15　プレイエリアと現実世界の座標系

更にユーザーの眼とコントローラーの計測点のずれ（2.3.3.1 節参照）を補正した．これらを纏めたものを式と図を用いて以下に示す．

まず，以下の様なベクトルを定義する．

$\boldsymbol{k}$：仮想世界の原点から現実世界の原点までの補正ベクトル
$\boldsymbol{p}$：仮想世界（Unity の世界）の原点からコントローラーの座標の測定点への位置ベクトル
$\boldsymbol{d_1}$：センサーにおける局所座標系でのコントローラーの計測点からユーザーの両眼中心までの補正ベクトル
$\boldsymbol{d_r}$：センサーにおける局所座標系でのユーザーの両眼中心点から右目までの補正ベクトル
$\boldsymbol{d_l}$：センサーにおける局所座標系でのユーザーの両眼中心点から左目までの補正ベクトル
R：センサーにおける局所座標系から現実世界の座標系に変換するための回転行列

ユーザーの左目，右目の座標を以下の様に置く．これらは計測データから式 A，B と表せられる(図 2.16，図 2.17).

$\boldsymbol{e_l}$：現実世界の原点からユーザーの左目までの位置ベクトル
$\boldsymbol{e_r}$：現実世界の原点からユーザーの右目までの位置ベクトル

$$\boldsymbol{e_l} = \boldsymbol{k} + \boldsymbol{p} + \mathrm{R}(\boldsymbol{d_1} + \boldsymbol{d_l}) \quad \cdots\mathrm{A}$$
$$\boldsymbol{e_r} = \boldsymbol{k} + \boldsymbol{p} + \mathrm{R}(\boldsymbol{d_1} + \boldsymbol{d_r}) \quad \cdots\mathrm{B}$$

しかし，今回の実験ではユーザーが正面を向くことを条件とするため現実世界の座標系とセンサー座標系は常に一致する．そのため，R は単位行列として扱う．すなわち，以下の式（式 C，D）として近似することが可能である．これを最終的なユーザーの眼球の位置とした．

$$\boldsymbol{e_l} = \boldsymbol{k} + \boldsymbol{p} + \boldsymbol{d_1} + \boldsymbol{d_l} \quad \cdots\mathrm{C}$$
$$\boldsymbol{e_r} = \boldsymbol{k} + \boldsymbol{p} + \boldsymbol{d_1} + \boldsymbol{d_r} \quad \cdots\mathrm{D}$$

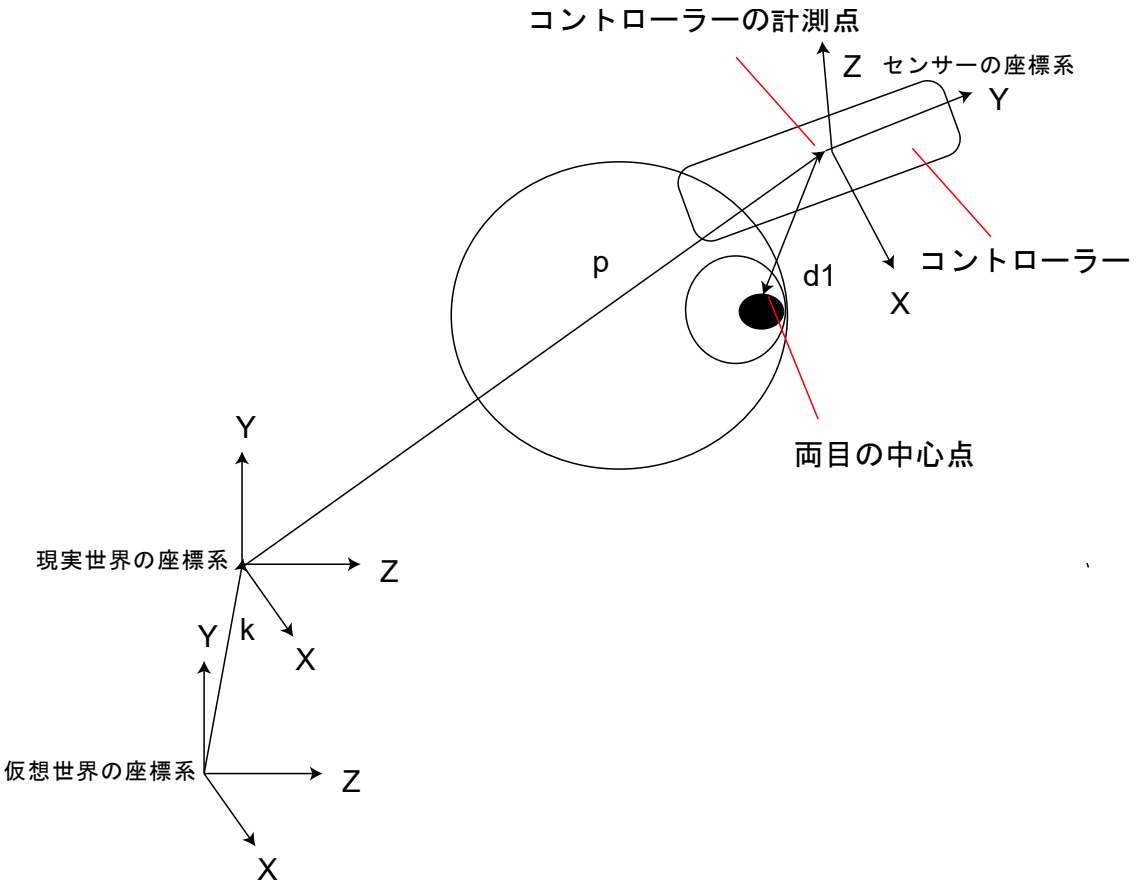

図 2.16　補正図（YZ 平面）

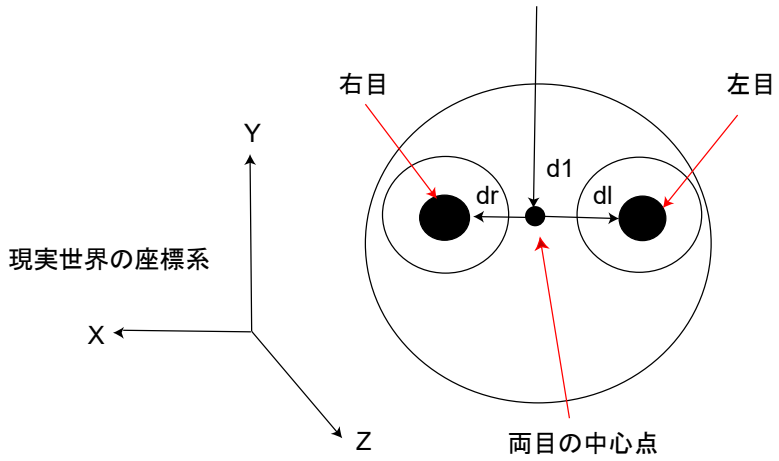

図 2.17　補正図（XY 平面）

第 3 章　誤差実験

第 2 章で述べた通りの環境をつくることで，理論上任意の場所に物体を見せることは可能である．しかし，以下に詳しく述べるが，現実世界での環境づくりは人間が行うことであるため，セットアップにおける誤差や測定誤差が生じることは避けられない．そこで，仮想世界の理想的な位置と現実世界での見える位置のずれを測定し，そのずれの大きさを本実験の精度評価とする．

3.1　実験方法

3.1.1　環境設計

Unity 上の仮想世界に仮想ディスプレイの中心から 1072[mm]離れ，地面から高さ 600[mm]の地点にごく小さな赤い球を配置した．
また，同様に現実世界の方でも現実ディスプレイから 1072[mm]離れたところにアルミフレームを立て，そこに先端が地面から高さ 600[mm]となるよう竹籤(A)を刺した．

被験者が観察する地点として，左，中央，右の 3 ヶ所に丸椅子を設置した．それらの椅子は指標から水平方向に 1000[mm]離れたところに設置し，また左右の椅子はそれぞれ中央から 350[mm]の距離にある（図 3.2）．

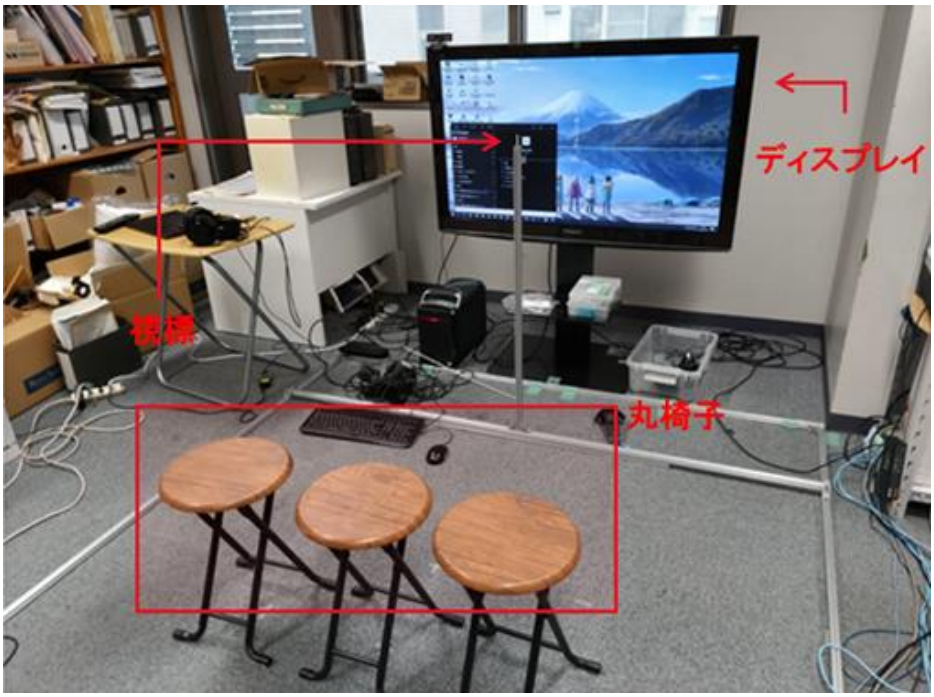

図 3.1　実験環境の写真

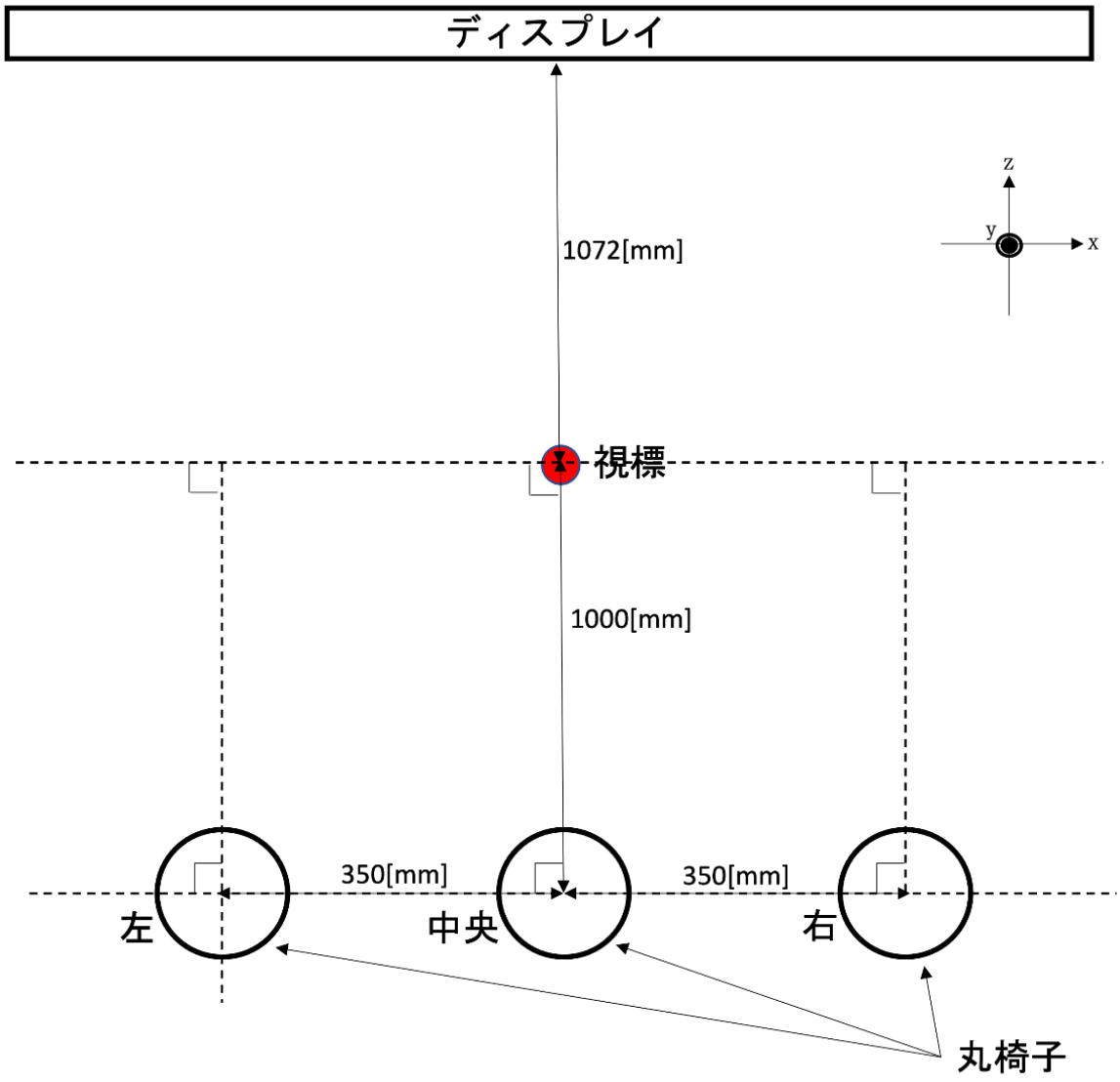

図 3.2　実験環境を上から見た図

3.1.2　測定方法

被験者は，図 2.12 のヘッドギアを頭に装着し，丸椅子に座る．被験者は，正面を向き，赤い球が見える場所に竹籤(B)を指す（図 3.3）．指した後の竹籤(B)の先端を実測点と呼ぶことにする．

また，仮想世界で設定した理想的な点，すなわちディスプレイの前に設置してある竹籤(A)の先端を目標点と呼ぶことにする．

この実測点と目標点との距離（実測点－目標点）を x 軸，y 軸，z 軸の 3 軸に分けて，定規を用いて計測した．定規は，最小目盛が 1[mm]のものを用いた．軸は，ディスプレイと床に平行な軸を x 軸，床と直交する軸を y 軸，ディスプレイと直交する軸を z 軸とした．また，被験者から見て右を x 軸の正，上を y 軸の正，ディスプレイ側を z 軸の正とした．

1 人につき，左，中央，右の 3 箇所での計測を 1 試行とし，これを 3 試行してもらった．被験者は，20 代男性の 10 人である．

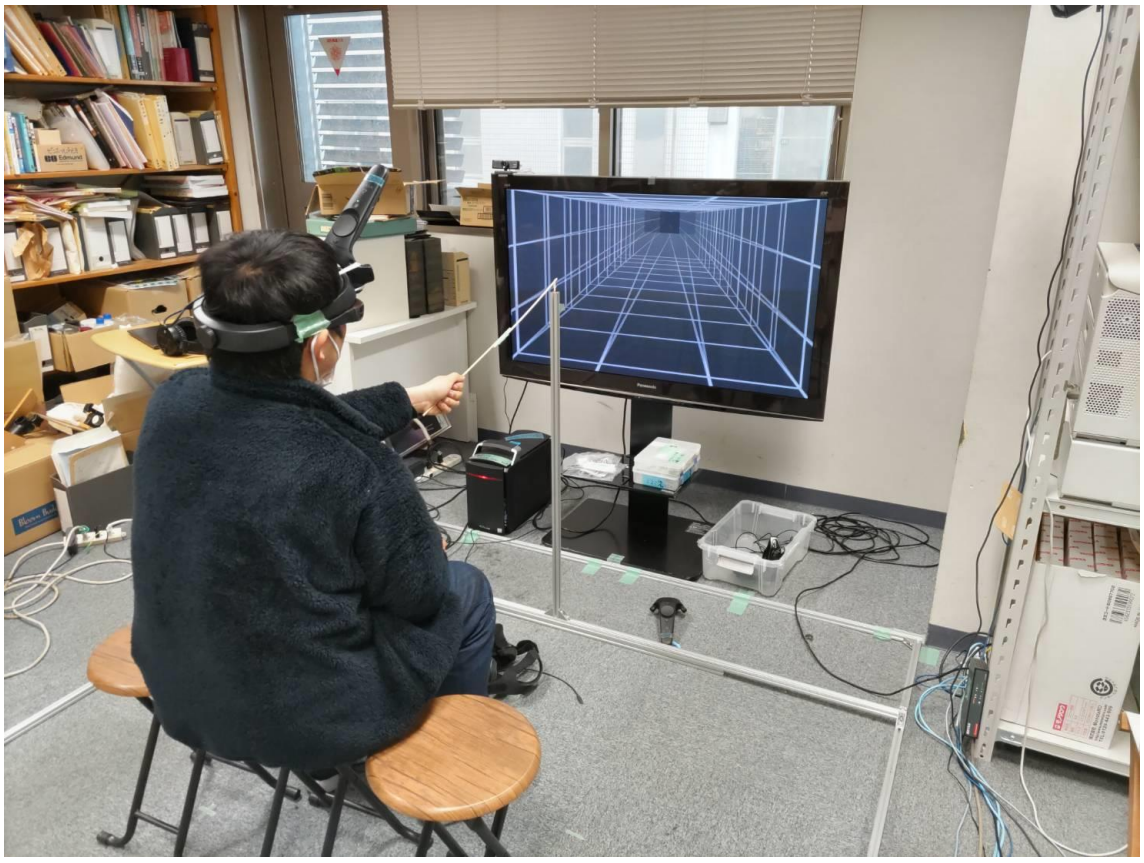

図 3.3　実験のイメージ

3.2　測定結果

全被験者の測定箇所毎の実測点と目標点との距離の平均値（距離，x 軸成分，y 軸成分，z 軸成分）を求め，棒グラフにした．エラーバーは，標準誤差である（図 3.4～図 3.7）．

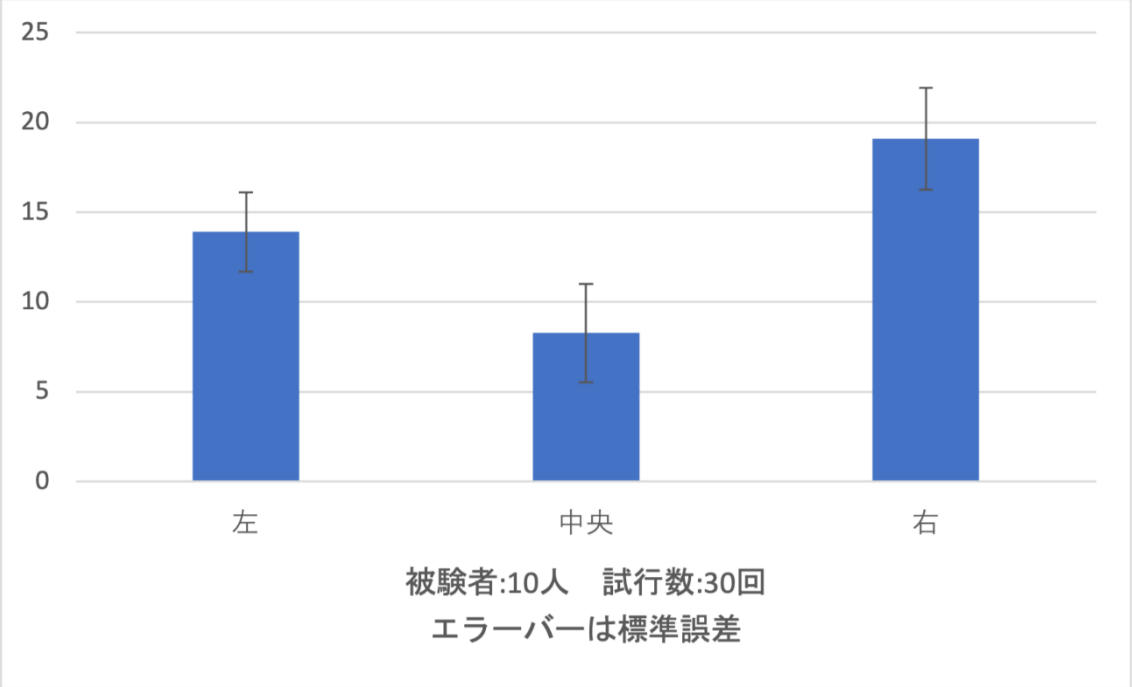

図 3.4　目標点と実測点とのずれ（距離）[mm]

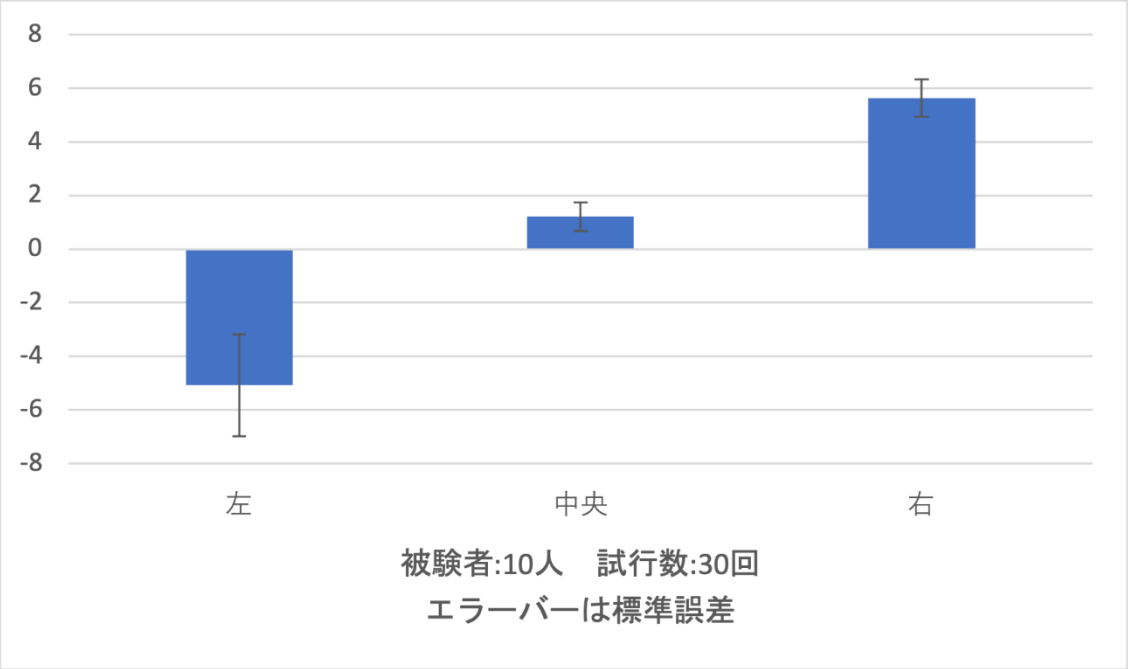

図 3.5　目標点と実測点とのずれ(x 軸方向)[mm]

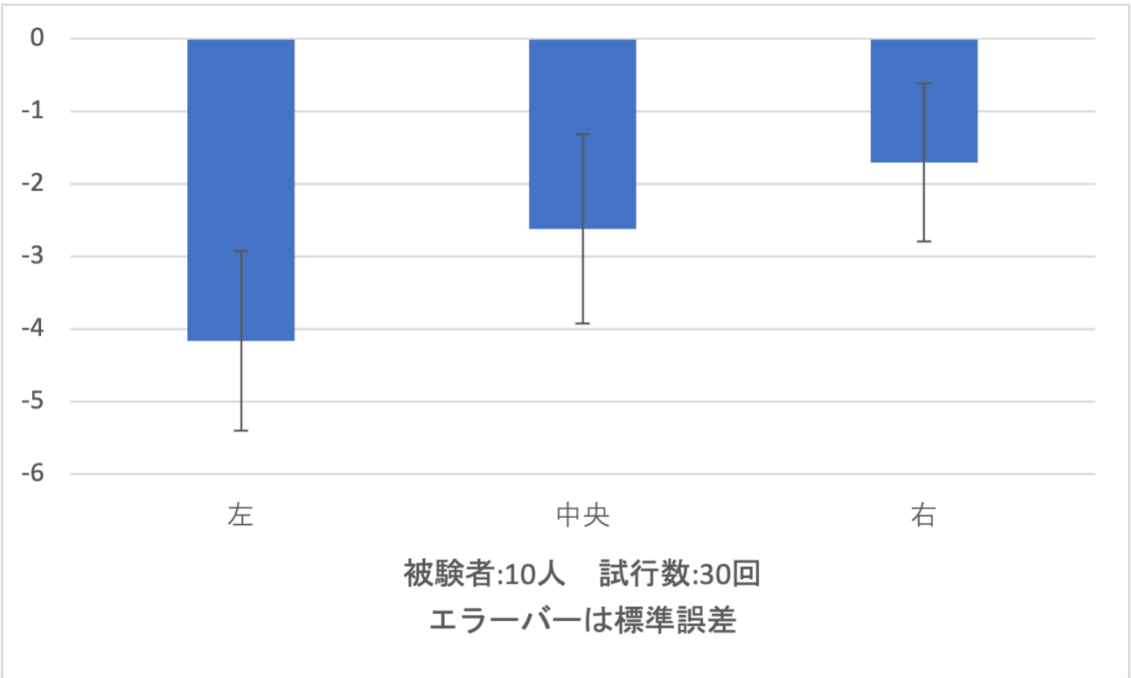

図 3.6　目標点と実測点とのずれ(y 軸方向)[mm]

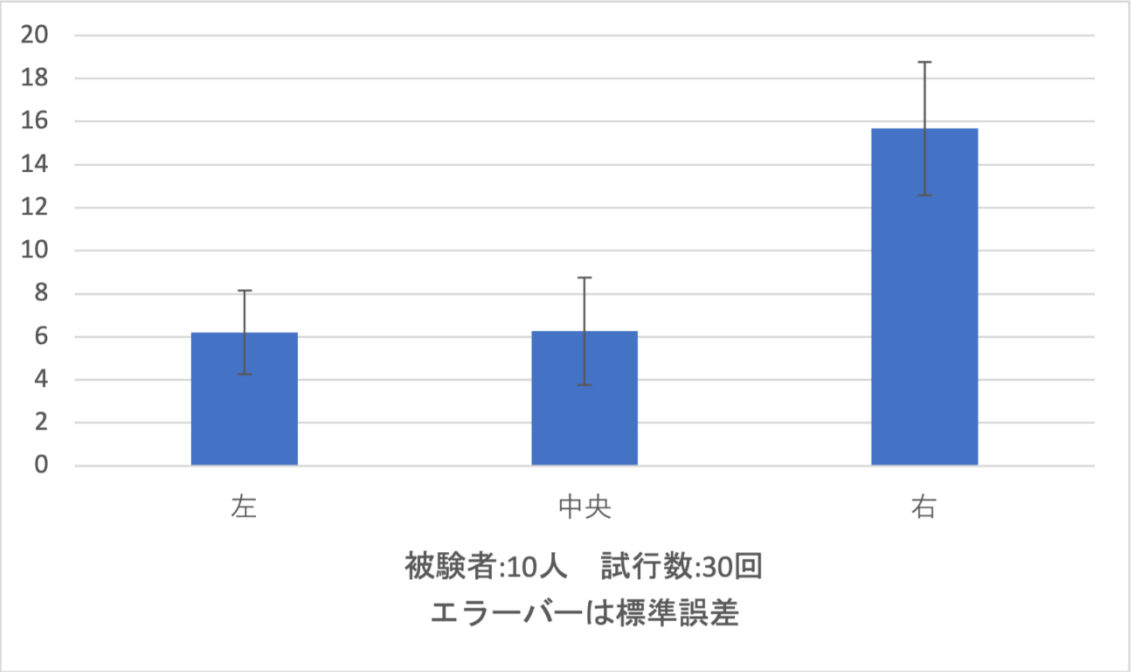

図 3.7　目標点と実測点とのずれ(z 軸方向)[mm]

3.3　考察

結果としては，我々が作成した環境の正確度（目標点と実測点のずれの大きさ）は，図 3.4 より約 5〜23[mm]であると言える．また，x 軸方向については，図 3.5 から約-8〜7[mm]，y 軸方向については，図 3.6 から約-6〜0[mm]，z 軸方向については，図 3.7 から約 1〜19[mm]であることがグラフから読み取れる．この実験結果から，各方向成分のずれの原因について考える．

y 軸方向のずれについて，全ての観測場所から見た時のずれが負となっているため，実測点は目標点よりも下に見えると言える．つまり，y 軸方向について，仮想世界と現実世界のオブジェクトの位置が完全に等しくなかったと考えられる．

また，z 軸方向のずれについて，全ての観測場所から見た時のずれが正となっているため，実測点は目標点より奥に見えると言える．つまり，z 軸方向についても，仮想世界と現実世界のオブジェクトの位置が完全に等しくなかったと考えられる．また，そのほかにも原因が 2 つあると考えている．1 つ目は，指標が小さな点であり，奥行きを知覚し難かったことである．2 つ目は，竹籤(A)と指標の隠蔽関係である．先ほど述べたように，実測点は目標点より下に観察されたため，竹籤(A)によって指標の下部が隠れてしまったと考えられる．

一方，x 軸方向のずれについて，観察場所が右の時は，右にずれて見え，観測場所が左の時は，左にずれて見えた．この原因は，z 軸方向のずれで述べたように，指標が実測点よりも奥に見えたことと，指標が極めて小さく奥行きを知覚し難かったためだと考えられる．また，中央から観察した時は少しであるが右にずれていることから，x 軸方向についても仮想世界と現実世界のオブジェクトの位置が完全に等しくなかったと考えられる．

第 4 章　纏めと今後の展望

今回の実験では 1.3 の目標で述べた，

・何処から見ても，同じ場所に物体が見える（自己運動視（1.3 節）が可能）.

・竹籤で同じ大きさの物体を作成し，手を伸ばすと物体がそこにあるように知覚する.

という事の実現に概ね成功した．被験者が，正面を向いた状態という限定条件の中ではあるが，3D 機能のあるテレビとアクティブシャッターメガネを使用し，仮想世界と現実世界での物体の位置を常に合わせることで，ディスプレイに表示される物体を現実世界で存在しているかのように見せることに成功した．これにより，仮想世界の自由な立体物を，多少の誤差は生じるが，現実世界にあるように知覚させることが可能となった.

しかし，本取り組みでは，現実世界の物体をただ観察する時と比べて，

・物体の方向に目線をずらす

・首を回転すること

の操作に関して，体験者が行うことが許されていないため，この 2 点を解消することが可能となれば，更に快適で現実的に利用可能である手段が増えるだろうと考えられる.

具体的な手段として，前者については，視線の動きを検出する機器を利用することによって可能となるだろうと考えられる．具体的には，視線検出装置を用いて目の動きを抽出し，その動きに合わせて Unity で作成した仮想世界のカメラを動かすことによって実現可能である．ただし，視線の移動速度とレイテンシ間の問題が考えられる.

後者については，VIVE のコントローラーが三次元の直行系を軸とする回転角の計測が可能であるため，現実世界の人の顔の向きと仮想世界のカメラの向きを等しくすることが可能であると考えられる.

この 2 点を実現できれば，体験者は，極めて自由な行動が可能となり，浮いている物体を更に現実的に知覚することが可能だろうと考えられる.

謝辞

本取り組みを進めるにあたり，御助言を承りました岐阜大学工学部電気電子・情報工学科の木島竜吾准教授に対し心から感謝の意を表する．また，誤差実験に協力頂き，日頃からお世話になっている木島研究室の先輩の方々に感謝の意を表する．

参考文献

[1] https://mcm-www.jwu.ac.jp/~physm/buturi16/holo/rittai.html

[2] https://direct.sanwa.co.jp/ItemPage/400-3DGS002_TSL

[3] https://www.nintendo.co.jp/kids/151216/about3d/

[4] 地上・BS・110 度 CS デジタルハイビジョンプラズマテレビ TH-P50VT2
https://panasonic.jp/viera/p-db/TH-P50VT2.html

[5] 3D グラス TY-EW3D2LW
https://panasonic.jp/viera/p-db/TY-EW3D2LW.html

[6] Unity 2020.2.5f1
https://unity3d.com/jp/unity/whats-new/2020.2.5

[7] SteamVR Plugin
https://assetstore.unity.com/packages/tools/integration/steamvr-plugin-32647?locale=ja-JP

[8] Unity Documentation 物理カメラの使用
https://docs.unity3d.com/ja/2019.4/Manual/PhysicalCameras.html

[9] Unity Documentation カメラ
https://docs.unity3d.com/ja/2019.4/Manual/class-Camera.html