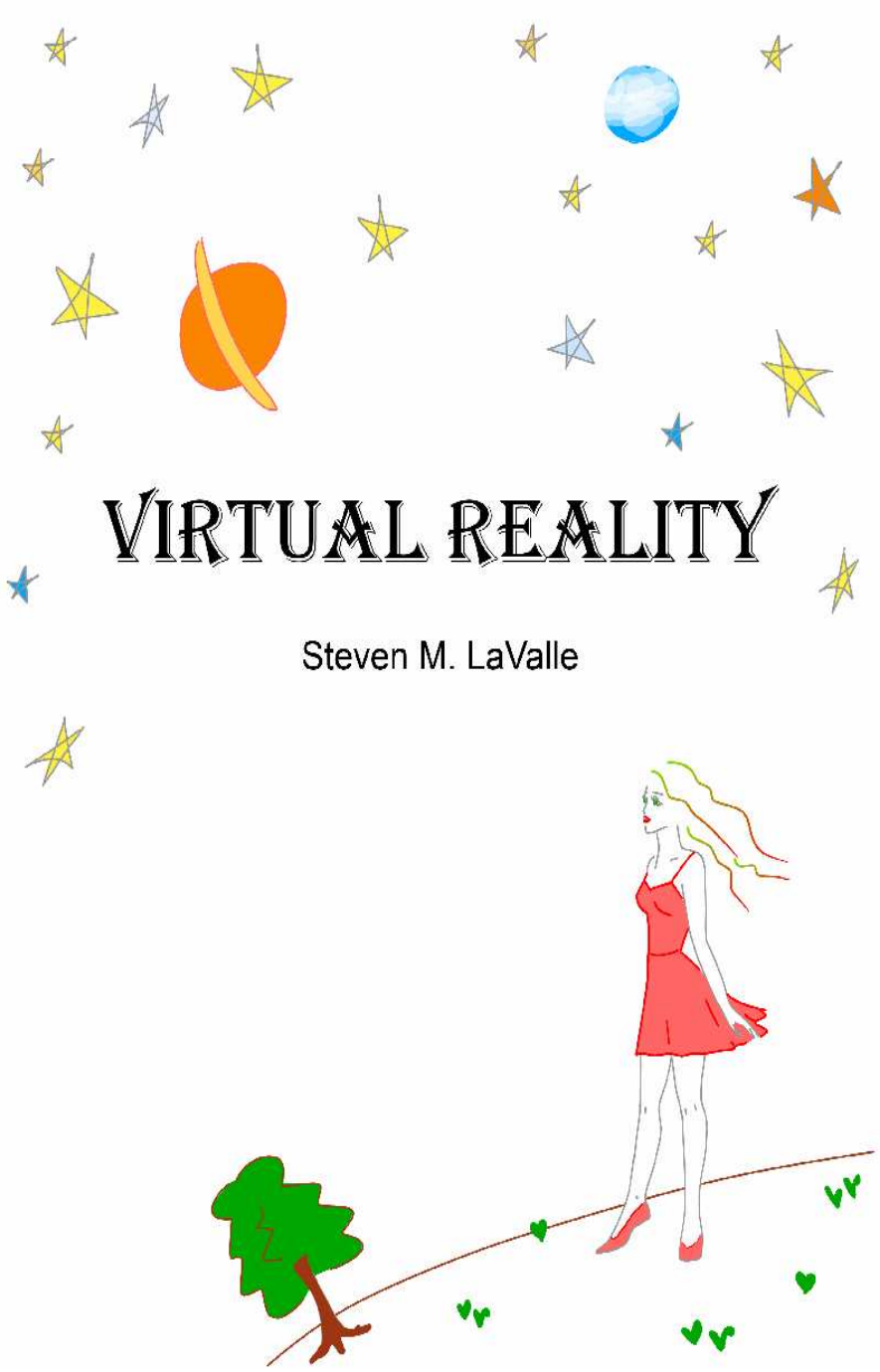



# Chapter 9

# Tracking

**Steven M. LaValle**

**University of Illinois**



Available for downloading at **http://vr.cs.uiuc.edu/**

# Chapter 9

# Tracking

| Chapter Status | Taken from *Virtual Reality*, S. M. LaValle |
|---|---|
|  | This online chapter is not the final version! Check **http://vr.cs.uiuc.edu/** for information on the latest draft version.<br><br>This draft was compiled on August 28, 2016. |

Keeping track of moving bodies in the physical world is a crucial part of any VR system. Tracking was one of the largest obstacles to overcome to bring VR headsets into consumer electronics, and it will remain a major challenge due to our desire to expand and improve VR experiences. Highly accurate tracking methods have been mostly enabled by commodity hardware components, such as inertial measurement units (IMUs) and cameras, that have plummeted in size and cost due to the smartphone industry.

Three categories of tracking may appear in VR systems:

1. **The user's sense organs:** Recall from Section 2.1 that sense organs, such as eyes and ears, have DOFs that are controlled by the body. If artificial stimuli are attached to a sense organ but they are meant to be perceived as attached to the surrounding world, then the position and orientation of the organ needs to be tracked. The inverse of the tracked transformation is applied to the stimulus to correctly "undo" these DOFs. Most of the focus is on *head tracking*, which is sufficient for visual and aural components of VR; however, the visual system may further require *eye tracking* if the rendering and display technology requires compensating for the eye movements discussed in Section 5.3.

2. **The user's other body parts:** If the user would like to see a compelling representation of his body in the virtual world, then its motion needs to be tracked so that it can be reproduced in the matched zone. Perhaps facial expressions or hand gestures are needed for interaction. Although perfect matching is ideal for tracking sense organs, it is not required for tracking other parts. Small movements in the real world could convert into larger virtual world motions so that the user exerts less energy. In the limiting case, the user could simply press a button to change the body configuration. For example, she might grasp an object in her virtual hand by a single click.

3. **The rest of the environment:** Physical objects in the real world that surrounds the user may need to be tracked. For objects that exist in the physical world but not the virtual world, it might be necessary to alert the user to their presence for safety reasons. Imagine the user is about to hit a wall, or trip over a toddler. In some VR applications, the tracked physical objects may be matched in VR so that the user receives touch feedback while interacting with them. In other applications, such as telepresence, a large part of the physical world could be "brought into" the virtual world through live capture.





Section 9.1 covers the easy case of tracking rotation around a single axis to prepare for Section 9.2, which extends the framework to tracking the 3-DOF orientation of a 3D rigid body. This relies mainly on the angular velocity readings of an IMU. The most common use is to track the head that wears a VR headset. Section 9.3 addresses the tracking of position and orientation together, which in most systems requires line-of-sight visibility between a fixed part of the physical world and the object being tracked. Section 9.4 discusses the case of tracking multiple bodies that are attached together by joints. Finally, Section 9.5 covers the case of using sensors to build a representation of the physical world so that it can be brought into the virtual world.

## 9.1 Tracking 2D Orientation

This section explains how the orientation of a rigid body is estimated using an IMU. The main application is determining the viewpoint orientation, $R_{eye}$ from Section 3.4 while the user is wearing a VR headset. Another application is estimating the orientation of a hand-held controller. For example, suppose we would like to make a laser pointer that works in the virtual world, based on a direction indicated by the user. The location of a bright red dot in the scene would be determined by the estimated orientation of a controller. More generally, the orientation of any human body part or moving object in the physical world can be determined if it has an attached IMU.

To estimate orientation, we first consider the 2D case by closely following the merry-go-round model of Section 8.1.2. The main issues are easy to visualize in this case, and extend to the more important case of 3D rotations. Thus, imagine that we mount a gyroscope on a spinning merry-go-round. Its job is to measure the angular velocity as the merry-to-round spins. It will be convenient throughout this chapter to distinguish a true parameter value from an estimate. To accomplish

this, a "hat" will be placed over estimates. Thus, let $\hat{\omega}$ correspond to the estimated or measured angular velocity, which may not be the same as $\omega$, the *true* value.

How are $\hat{\omega}$ and $\omega$ related? If the gyroscope were functioning perfectly, then $\hat{\omega}$ would equal $\omega$; however, in the real world this cannot be achieved. The main contributor to the discrepancy between $\hat{\omega}$ and $\omega$ is *calibration error*. The quality of calibration is the largest differentiator between an expensive (thousands of dollars) and cheap (one dollar) IMU.

We now define a simple model of calibration error. The following *sensor mapping* indicates how the sensor output is related to true angular velocity:

$$\hat{\omega} = a + b\,\omega. \tag{9.1}$$

Above, $a$ and $b$ are called the *offset* and *scale*, respectively. They are unknown constants that interfere with the measurement. If $\omega$ were perfectly measured, then we would have $a = 0$ and $b = 1$.

Consider the effect of calibration error. Comparing the measured and true angular velocities yields:

$$\hat{\omega} - \omega = a + b\,\omega - \omega = a + \omega(b - 1). \tag{9.2}$$

Now imagine using the sensor to estimate the orientation of the merry-go-round. We would like to understand the difference between the true orientation $\theta$ and an estimate $\hat{\theta}$ computed using the sensor output. Let $d(t)$ denote a function of time called the *drift error*:

$$d(t) = \theta(t) - \hat{\theta}(t). \tag{9.3}$$

Suppose it is initially given that $\theta(0) = 0$, and to keep it simple, the angular velocity $\omega$ is constant. By integrating (9.2) over time, drift error is

$$d(t) = (\hat{\omega} - \omega)t = (a + b\,\omega - \omega)t = (a + \omega(b - 1))t. \tag{9.4}$$

Of course, the drift error grows as $a$ is larger or $b$ deviates from one; however, note that the second component is proportional to $\omega$. Ignoring $a$, this means that the drift error is proportional to the speed of the merry-go-round. In terms of tracking a VR headset using a gyroscope, this means that tracking error increases at a faster rate as the head rotates more quickly [21].

At this point, four general problems must be solved to make an effective tracking system, even for this simple case:

1. **Calibration:** If a better sensor is available, then the two can be closely paired so that the outputs of the worse sensor are transformed to behave as close to the better sensor as possible.

2. **Integration:** Every sensor provides measurements at discrete points in time, resulting in a *sampling rate*. The orientation is estimated by aggregating or integrating the measurements.



3. **Registration:** The initial orientation must somehow be determined, either by an additional sensor or a clever default assumption or startup procedure.

4. **Drift correction:** Other sensors directly estimate the drift error. The resulting information is used to gradually eliminate it.

All of these issues remain throughout this chapter for the more complicated settings. The process of combining information from multiple sensor readings is often called *sensor fusion* or *filtering*.

We discuss each of these for the 2D case, before extending the ideas to the 3D case in Section 9.2.

**Calibration** You could buy a sensor and start using it with the assumption that it is already well calibrated. For a cheaper sensor, however, the calibration is often unreliable. Suppose we have an expensive, well-calibrated sensor that reports angular velocities with very little error. Let $\hat{\omega}'$ denote its output, to distinguish it from the forever unknown true value $\omega$. This measurement could be provided by an expensive turntable, calibration rig, or robot.

Calibration involves taking many samples, typically thousands, and comparing $\hat{\omega}'$ to $\hat{\omega}$. A common criterion is the *sum of squares error*, which is given by

$$\sum_{i=1}^{n} (\hat{\omega}_i - \hat{\omega}'_i)^2 \tag{9.5}$$

for $n$ samples of the angular velocity. The task is to determine a transformation to apply to the sensor outputs $\hat{\omega}$ so that it behaves as closely as possible to the expensive sensor which produced $\hat{\omega}'$. Thus, select constants $c_1$ and $c_2$ that optimize the error:

$$\sum_{i=1}^{n} (c_1 + c_2\,\hat{\omega} - \hat{\omega}')^2. \tag{9.6}$$

This is a classical regression problem referred to as *linear least-squares*. It is typically solved by calculating the Moore-Penrose pseudoinverse of an non-square matrix that contains the sampled data [29].

Once $c_1$ and $c_2$ are calculated, every future sensor reading is transformed as

$$\hat{\omega}_{cal} = c_1 + c_2\,\hat{\omega}, \tag{9.7}$$

in which $\hat{\omega}$ is the original, raw sensor output, and $\hat{\omega}_{cal}$ is the calibrated output. Thus, the calibration produces a kind of invisible wrapper around the original sensor outputs so that the better sensor is simulated. The raw sensor outputs are no longer visible to outside processes. The calibrated outputs will therefore simply be referred to as $\hat{\omega}$ in future discussions.

**Integration** Sensor outputs usually arrive at a regular sampling rate. For example, the Oculus Rift CV1 gyroscope provides a measurement every 1ms (yielding a 1000Hz sampling rate). Let $\hat{\omega}[k]$ refer to the $k$th sample, which arrives at time $k\Delta t$.

The orientation $\theta(t)$ at time $t = k\Delta t$ can be estimated by integration as:

$$\hat{\theta}[k] = \theta(0) + \sum_{i=1}^{k} \hat{\omega}[i]\Delta t. \tag{9.8}$$

Each output $\hat{\omega}[i]$ causes a rotation of $\Delta\theta[i] = \hat{\omega}[i]\Delta t$. It is sometimes more convient to write (9.8) in an incremental form, which indicates the update to $\hat{\theta}$ after each new sensor output arrives:

$$\hat{\theta}[k] = \hat{\omega}[k]\Delta t + \hat{\theta}[k-1]. \tag{9.9}$$

For the first case, $\hat{\theta}[0] = \theta(0)$.

If $\omega(t)$ varies substantially between $\theta(k\Delta t)$ and $\theta((k+1)\Delta t)$, then it is helpful to know what $\hat{\omega}[k]$ corresponds to. It could be angular velocity at the start of the interval $\Delta t$, the end of the interval, or an average over the interval. If it is the start or end, then a trapezoidal approximation to the integral may yield less error over time [].

**Registration** In (9.8), the initial orientation $\theta(0)$ was assumed to be known. In practice, this corresponds to a *registration* problem, which is the initial alignment between the real and virtual worlds. To understand the issue, suppose that $\theta$ represents the yaw direction for a VR headset. One possibility is to assign $\theta(0) = 0$, which corresponds to whichever direction the headset is facing when the tracking system is turned on. This might be when the system is booted, or if the headset has an "on head" sensor, then it could start when the user attaches the headset to his head. Often times, the forward direction could be unintentionally set in a bad way. For example, if one person starts a VR demo and hands the headset to someone else, who is facing another direction, then in VR the user would not be facing in the intended forward direction. This could be fixed by a simple option that causes "forward" (and hance $\theta(t)$) to be redefined as whichever direction the user is facing at present.

An alternative to this entire problem is to declare $\theta(0) = 0$ to correspond to a direction that is fixed in the physical world. For example, if the user is sitting at a desk in front of a computer monitor, then the forward direction could be defined as the yaw angle for which the user and headset are facing the monitor. Implementing this solution requires a sensor that can measure the yaw orientation with respect to the surrounding physical world. For example, with the Oculus Rift CV1, the user faces a stationary camera, which correponds to the forward direction.



**Drift correction** To make a useful tracking system, the drift error (9.3) cannot be allowed to accumulate. Even if the sensor were perfectly calibrated, drift error would nevertheless grow due to other factors such as quantized output values, sampling rate limitations, and unmodeled noise. The first problem is to estimate the drift error, which is usually accomplished with an additional sensor. Practical examples of this will be given in Section 9.2. For the simple merry-go-round example, imagine that an overhead camera takes a picture once in a while to measure the orientation. Based on this, let $\hat{d}[k]$ denote the estimated drift error.

The problem is that there are now two conflicting sources of information. A classic way to blend these two sources is via a *complementary filter*. This gives

$$\hat{\theta}_c[k] = \alpha\hat{d}[k] + (1-\alpha)\hat{\theta}[k], \tag{9.10}$$

in which $\alpha$ is a *gain* parameter, with $0 < \alpha < 1$. Above, $\hat{\theta}_c$ denotes the corrected estimate. The filter essentially interpolates between the two sources. Since the gyroscope is usually accrurate over short times but gradually drifts, $\alpha$ is chosen to be close to zero (for example, $\alpha = 0.001$). This causes the drift correction term $\hat{d}[k]$ to be gradually applied. It is important to select the gain $\alpha$ to be high enough so that the drift is correct, but low enough so that the user does not perceive the corrections. The gain could be selected "optimally" by employing a Kalman filter [4, 13, 16]; however, the optimality only holds if we have a linear stochastic system, which is not the case in rigid body tracking. The relationship between Kalman and complementary filters, for the exact models used in this chapter, appears in [12].

## 9.2 Tracking 3D Orientation

**IMUs** Recall from Section 2.1 (Figure 2.9) that IMUs have recently gone from large, heavy mechanical systems to cheap, microscopic MEMS circuits. This progression was a key enabler to high-quality orientation tracking. The gyroscope measures angular velocity along three orthogonal axes, to obtain $\hat{\omega}_x$, $\hat{\omega}_y$, and $\hat{\omega}_z$. For each axis, the sensing elements lie in the perpendicular plane, much like the semicircular canals in the vestibular organ (Section 8.2). The sensing elements in each case are micromachined mechanical elements that vibrate and operate like a tuning fork. If the sensor rotates in its direction of sensitivity, the elements experience Coriolis forces, which are converted into electrical signals that are calibrated to produce an output in degrees or radians per second; see Figure 9.1.

IMUs usually contain additional sensors that are useful for detecting drift errors. Most commonly, accelerometers measure linear acceleration along three axes to obtain $\hat{a}_x$, $\hat{a}_y$, and $\hat{a}_z$. The principle of their operation is shown in Figure 9.2. MEMS magnetometers also appear on many modern IMUs, which measure magnetic field strength along the three perpendicular axis. This is often accomplished by the mechanical motion of a MEMS structure that is subject to Lorentz force as it conducts inside of a magnetic field.

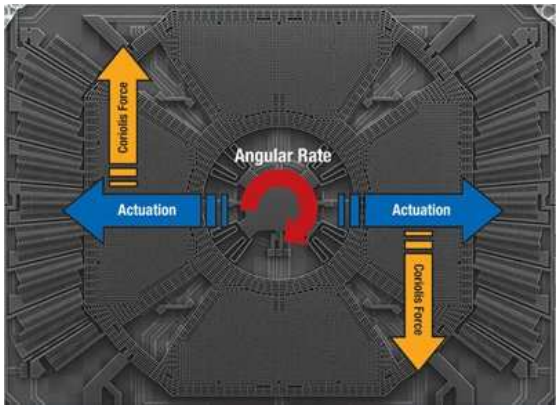



Figure 9.1: The vibrating MEMS elements respond to Coriolis forces during rotation, which are converted into an electrical signal. (Figure by Fabio Pasolini.)

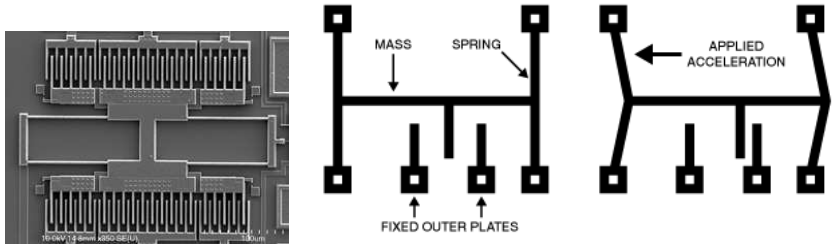
**Calibration** Recall from Section 9.1 that the sensor outputs are distorted due to calibration issues. In the one-dimensional angular velocity case, there were only two parameters, for scale and offset, which appear in (9.1). In the 3D setting, this would naturally extend to 3 scale and 3 offset parameters; however, the situation is worse because there may also be errors due to non-orthogonality of the MEMS elements. All of these can be accounted for by 12 parameters arranged in a homogeneous transform matrix:

$$\begin{bmatrix} \hat{\omega}_x \\ \hat{\omega}_y \\ \hat{\omega}_z \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c & j \\ d & e & f & k \\ g & h & i & \ell \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \\ 1 \end{bmatrix} \tag{9.11}$$

There are 12 and not 6 DOFs because the upper left, 3-by-3, matrix is not constrained to be a rotation matrix. The $j$, $k$, and $\ell$ parameters correspond to offset, whereas all others handle scale and non-orthogonality. Following the same methodology as in Section 9.1, the inverse of this transform can be estimated by minimizing the least squares error with respect to a better sensor that provides ground truth. The outputs of the MEMS sensor are then adjusted by applying the estimated homogeneous transform to improve performance (this is an extension of (9.7) to the 12-parameter case. This general methodology applies to calibrating gyroscopes and accelerometers. Magnetometers may also be calibrated in this way, but have further complications such as *soft iron bias*.

An additional challenge with MEMS sensors is dealing with other subtle dependencies. For example, the outputs are sensitive to the particular temperature of the MEMS elements. If a headset heats up during use, then calibration parameters are needed for every temperature that might arise in practice. Fortunately,






(a) (b)

Figure 9.2: (a) A MEMS element for sensing linear acceleration. (b) Due to linear acceleration in one direction, the plates shift and cause a change in capacitance as measured between the outer plates. (Figure by David Askew.)

IMUs usually contain a temperature sensor that can be used to associate the calibration parameters with the corresponding temperatures. Finally, MEMS elements may be sensitive to forces acting on the circuit board, which could be changed, for example, by a dangling connector. Care must be given to isolate external board forces from the MEMS circuit.

**Integration** Now consider the problem converting the sequence of sensor outputs into an estimate of the 3D orientation. At each stage $k$ a vector

$$\hat{\omega}[k] = (\hat{\omega}_x[k], \hat{\omega}_y[k], \hat{\omega}_z[k]) \tag{9.12}$$

arrives from the sensor. In Section 9.1, the sensor output $\hat{\omega}[k]$ was converted to a change $\Delta\theta[k]$ in orientation. For the 3D case, the change in orientation is expressed as a *quaternion*.

Let $q(v, \theta)$ be the quaternion obtained by the axis-angle conversion formula (3.29). Recall from Section 8.1.2 that the instantaneous axis of rotation is the magnitude of the angular velocity. Thus, if $\hat{\omega}[k]$ is the sensor output at stage $k$, then the estimated rotation axis is

$$\hat{v}[k] = \hat{\omega}[k] / \|\hat{\omega}[k]\|. \tag{9.13}$$

Furthermore, the estimated amount of rotation that occurs during time $\Delta t$ is

$$\Delta\hat{\theta}[k] = \|\hat{\omega}[k]\| \Delta t. \tag{9.14}$$

Using the estimated rotation axis (9.13) and amount (9.14), the orientation change over time $\Delta t$ is estimated to be

$$\Delta\hat{q}[k] = q(\hat{v}[k], \Delta\hat{\theta}[k]). \tag{9.15}$$

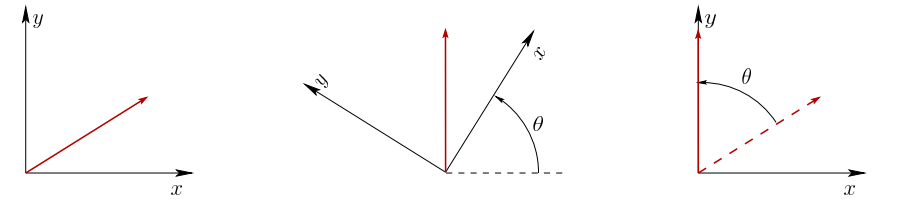



Figure 9.3: If "up" is perfectly sensed by a sensor that is rotated by $\theta$, then its output needs to be rotated by $\theta$ to view it from the world frame.

Using (9.15) at each stage, the estimated orientation $\hat{q}[k]$ after obtaining the latest sensor output is calculated incrementally from $\hat{q}[k-1]$ as

$$\hat{q}[k] = \Delta\hat{q}[k] * \hat{q}[k-1], \tag{9.16}$$

in which $*$ denotes quaternion multiplication. This is the 3D generalization of (9.9), at which point simple addition could be used to combine rotations in the 2D case. In (9.16), quaternion multiplication is needed to aggregate the change in orientation (simple addition is commutative, which is inappropriate for 3D rotations).

**Registration** The registration problem for the yaw component is the same as in Section 9.2. The forward direction may be chosen from the initial orientation of the rigid body or it could be determined with respect to a fixed direction in the world. The pitch and roll components should be determined so that they align with gravity. The virtual world should not appear to be tilted with respect to the real world (unless that is the desired effect, which is rarely the case).

**Tilt correction** The drift error $d(t)$ in (9.3) was a single angle, which could be positive or negative. If added to the estimate $\hat{\theta}(t)$, the true orientation $\theta(t)$ would be obtained. It is similar for the 3D case, but with quaternion algebra. The 3D drift error is expressed as

$$d(t) = q(t) * \hat{q}^{-1}(t), \tag{9.17}$$

which is equal to the identity rotation if $q(t) = \hat{q}(t)$. Furthermore, note that applying the drift error to the estimate yields $q(t) = d(t) * \hat{q}(t)$.

Since the drift error is 3D rotation, it could be constructed as the product of a yaw, pitch, and a roll. Let *tilt error* refer to the part of the drift error that corresponds to pitch and roll. This will be detected using an "up" sensor. Let *yaw error* refer to the remaining part of the drift error, which will be detecting using a "compass". In reality, there do not exist perfect "up" and "compass" sensors, which will be addressed later.

Suppose that a sensor attached to the rigid body always reports an "up" vector that is parallel to $y$ axis in the fixed, world coordinate frame. In other words, it would be parallel to gravity. Since the sensor is mounted to the body, it reports its values in the coordinate frame of the body. For example, if the body were rolled 90 degrees so that its $x$ axis is pointing straight up, then the "up" vector would be reported as $(0, 0, 1)$, instead of $(0, 1, 0)$. To fix this, it would be convenient to transform the sensor output into the world frame. This involves rotating it by $q$, the body orientation. For our example, this roll rotation would transform $(0, 0, 1)$ into $(0, 1, 0)$. Figure 9.3 shows a 2D example of this problem.



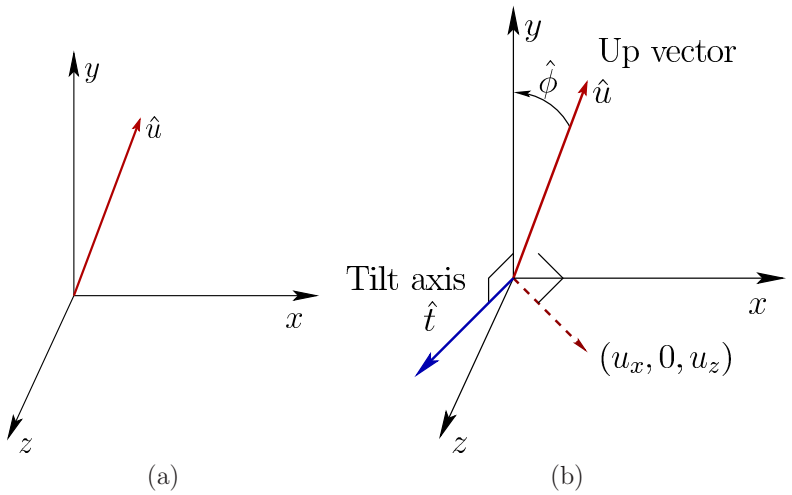



Figure 9.4: (a) Tilt error causes a discrepancy between the $y$ axis and the sensed up vector that is rotated using the estimate $\hat{q}[k]$ to obtain $\hat{u}$. (b) The *tilt axis* is normal to $\hat{u}$; a rotation of $-\phi$ about the tilt axis would bring them into alignment, thereby eliminating the tilt error.

Now suppose that drift error has occured and that $\hat{q}[k]$ is the estimated orientation. If this transform is applied to the "up" vector, then due to drift error, it might not be aligned with the $y$ axis, as shown Figure 9.4. The up vector $\hat{u}$ is projected into the $xz$ plane to obtain $(\hat{u}_x, 0, \hat{u}_z)$. The *tilt axis* lies in the $xz$ plane and is constructed as the normal to the projected up vector: $\hat{t} = (\hat{u}_z, 0, -\hat{u}_x)$. Performing a rotation of $\phi$ about the axis $\hat{t}$ would move the up vector into alignment with the $y$ axis. Thus, the tilt error portion of the drift error is the quaternion $q(\hat{t}, \phi)$.

Unfortunately, there is no "up" sensor. In practice, the accelerometer is used to measure the "up" direction because gravity acts on the sensor, causing the sensation of upward acceleration at roughly $9.8\text{m}/s^2$. The problem is that it also responds to true linear acceleration of the rigid body, and this cannot be separated from gravity due to the Einstein equivalence principle. It measures the vector sum of gravity and true linear acceleration, as shown in Figure 9.5. A simple

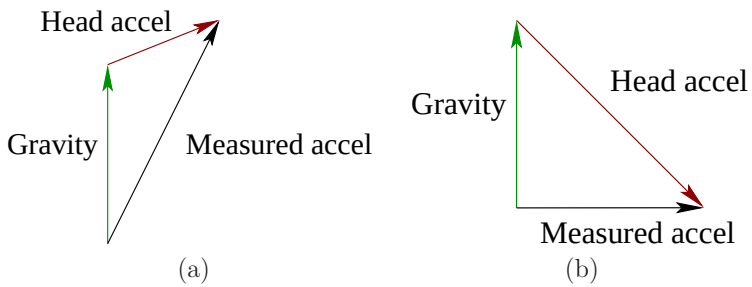



Figure 9.5: (a) There is no gravity sensor; the accelerometer measures the vector sum of apparent acceleration due to gravity and the true acceleration of the body. (b) A simple heuristic of accepting the reading as gravity only if the magnitude is appromimately 9.8m$^2$ will fail in some cases.

heuristic is to trust accelometer outputs as an estimate of the "up" direction only if its magnitude is close to 9.8m$^2$ [7]. This could correspond to the common case in which the rigid body is stationary. However, this assumption is unreliable because downward and lateral linear accelerations can be combined to provide an output that is close to 9.8m$^2$, but with a direction that is far from gravity. Better heuristics may be built from simultaneously considering the outputs of other sensors or the rate at which "up" appears to change.

Assuming the accelerometer is producing a reliable estimate of the gravity direction, the up vector $\hat{u}$ is calculated from the accelerometer output $\hat{a}$ by using (3.33), to obtain

$$\hat{u} = \hat{q}[k] * \hat{a} * \hat{q}^{-1}[k]^{-1}. \tag{9.18}$$

**Yaw correction** The remaining drift error component is detected by a "compass", which outputs a vector that lies in the world $xz$ plane and always points "north". Suppose this is $\hat{n} = (0, 0, -1)$. Once again, the sensor output occurs in the coordinate frame of the body, and needs to be transformed by $\hat{q}[k]$. The difference between $\hat{n}$ and the $-z$ axis is the resulting yaw drift error.

As in the case of the "up" sensor, there is no "compass" in the real world. Instead, there is a magnetometer, which measures a 3D magnetic field vector: $(\hat{m}_x, \hat{m}_y, \hat{m}_z)$. Suppose this is used to measure the Earth's magnetic field. It turns out that the field vectors do not "point" to the North pole. The Earth's magnetic field produces 3D vectors that generally do not lie in the horizontal plane, resulting in an *inclination angle*. Thus, the first problem is that the sensor output must be projected into the $xz$ plane. Residents of Ecuador may enjoy magnetic field vectors that are nearly horizontal; however, in Finland they are closer to vertical; see Figure 9.6. If the magnetic field vector is close to vertical, then the horizontal component may become too small to be useful.

Another issue is that the projected vector in the horizontal plane does not



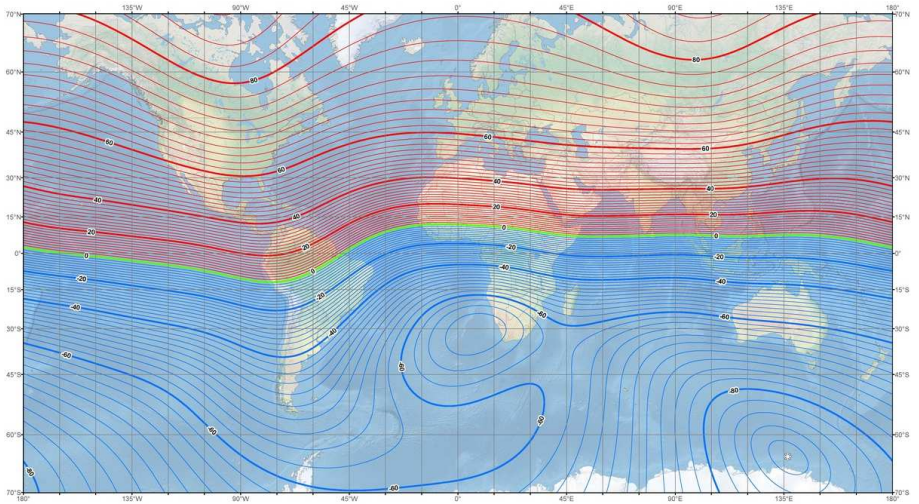


Figure 9.6: The inclination angle of the Earth's magnetic field vector varies greatly over the Earth. (Map developed by NOAA/NGDC and CIRES.)

point north, resulting in a *declination angle*; this is the deviation from north. Fortuntely, reference to the true north is not important. It only matters that the sensor output is recorded in the registration stage to provide a fixed yaw reference.

The most significant problem is that the magnetometer measures the vector sum of *all* magnetic field sources. In addition to the Earth's field, a building generates its own field due to ferromagnetic metals. Furthermore, such materials usually exist on the circuit board that contains the sensor. For this case, the field moves with the sensor, generating a constant vector offset. Materials that serve as a source of magnetic fields are called *hard iron*. Other materials distort magnetic fields that pass through them; these are called *soft iron*. Magnetometer calibration methods mainly take into account offsets due to hard-iron bias and eccentricities due to soft-iron bias [9, 15].

After all calibrations have been performed, the yaw drift error can be estimated from most locations with a few degrees of accuracy, which is sufficient to keep yaw errors from gradually accumulating. There are still problems. If a strong field is placed near the sensor, then it will be non-constant. This could cause the measured direction to change as the rigid body translates back and forth. Another problem is that in some building locations, vector sum of the Earth's magnetic field and the field generated by the building could by approximately zero (they are of similar magnitude and pointing in opposite directions). In this unfortunate case, the magnetometer cannot produce useful outputs for yaw drift error detection.

| Yaw | Pitch | Roll | Error |
|---|---|---|---|
| + | + | + | None |
| - | + | + | L/R mix, flipped $x$ |
| + | - | + | L/R mix, flipped $y$ |
| + | + | - | L/R mix, flipped $z$ |
| + | - | - | Inverse and L/R mix, flipped $x$ |
| - | + | - | Inverse and L/R mix, flipped $y$ |
| - | - | + | Inverse and L/R mix, flipped $z$ |
| - | - | - | Inverse |

Figure 9.7: A table to help debug common viewpoint transform errors. Each + means that the virtual world appears to move the correct way when performing the yaw, pitch, or roll. Each − means it moves in the opposite way. The first case is correct. All others are bugs. "L/R mix" means that left and right-handed coordinate systems are getting mixed; the axis that was flipped is indicated.

**Filtering** Using the detected drift error, filtering works in the same way as described in Section 9.1. The complementay filter (9.10) is upgraded to work with quaternions:

$$\hat{q}_c[k] = \alpha \hat{d}[k] * (1 - \alpha)\hat{q}[k]. \tag{9.19}$$

The estimated drift error $\hat{d}[k]$ is obtained by multiplying the estimated tilt and yaw errors. Alternatively, they could contribute separately, with different gains for each, and even combined with drift error estimates from more sources [22].

**Setting the viewpoint** The viewpoint is set using the estimated orientation $\hat{q}[k]$, although it might be adjusted to account for alternative timings, for the purpose of prediction or image warping, as discussed in Section 7.4. Let $\hat{q}(t)$ denote the estimated orientation for time $t$. In terms of Section 3.4, we have just estimated $R_{eye}$. To calculate the correct viewpoint, the inverse is needed. Thus, $\hat{q}^{-1}(t)$ would correctly transform models to take the estimated viewpoint into account.

**A debugging tip** Programmers often make mistakes when connecting the tracked orientation to the viewpoint. Figure 9.7 shows a table of the common mistakes. To determine whether the transform has been correctly applied, one should put on the headset and try rotating about the three canonical axes: A pure yaw, a pure pitch, and a pure roll. Let + denote that the world is moving corrected with respect to a head rotation. Let − denote that it seems to move in the opposite direction. Figure 9.7 shows a table of the eight possible outcomes and the most likely cause of each problem.



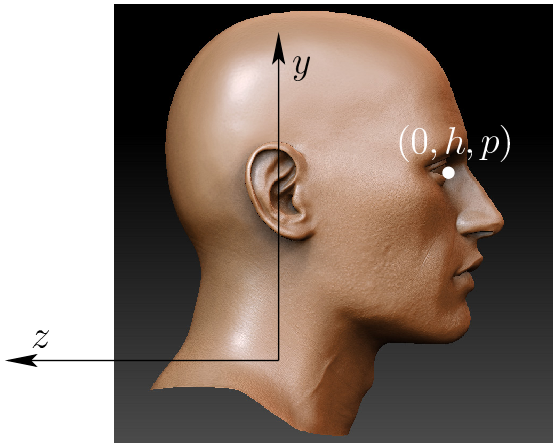



Figure 9.8: To obtain a *head model*, the rotation center is moved so that orientation changes induce a plausible translation of the eyes. The height $h$ is along the $y$ axis, and the protrusion $p$ is along the $z$ axis (which leads a negative number).

**A head model** The translation part of the head motion has not been addressed. Ideally, the head should be the same height in the virtual world as in the real world. This can be handled by the translation part of the $T_{eye}$ matrix (3.35).

We must also account for the fact that as the head rotates, the eyes change their positions. For example, in a yaw head movement (nodding "no"), the pupils displace a few centimeters in the $x$ direction. More accurately, they travel along a circular arc in a horizontal plane. To more closely mimic the real world, the movements of the eyes through space can be simulated by changing the center of rotation according to a fictitious *head model*. This trick is needed until Section 9.3, where position is instead estimated from sensors.

Recall from Section 3.5 that the cyclopean viewpoint was first considered and then modified to handle left and right eyes by applying horizontal offsets by inserting $T_{left}$ (3.49) and $T_{right}$ (3.51). In a similar way, offsets in the $y$ and $z$ directions can be added to account for displacement that would come from a rotating head. The result is to insert the following before or after $T_{right}$ and $T_{left}$:

$$T_{head} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & h \\ 0 & 0 & 1 & p \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{9.20}$$

in which $h$ is a height parameter and $p$ is a protrusion parameter. The idea is to choose $h$ and $p$ that would correspond to the center of rotation of the head. The parameter $h$ is the distance from the rotation center to the eye height, along the $y$ axis. A typical value is $h = 0.15$m. The protrusion $p$ is the distance from

the rotation center to the cyclopean eye. A typical value is $p = -0.10$m, which is negative because it extends opposite to the $z$ axis. Using a fake head model approximates the eye locations as the user rotates her head; however, it is far from perfect. If the torso moves, then this model completely breaks, resulting in a large mismatch between the real and virtual world head motions. Nevertheless, this head model is currently used in popular headsets, such as Samsung Gear VR.

An issue also exists with the $y$ height of the head center. They user may be seated in the real world, but standing in the virtual world. This mismatch might be uncomfortable. The brain knows that the body is seated because of proprioception, regardless of the visual stimuli provided by VR. If the user is standing, then the head-center height could be set so that the eyes are at the same height as in the real world. This issue even exists for the case of full six-DOF tracking, which is covered next; the user might be sitting, and a vertical offset is added to make him appear to be standing in VR.

## 9.3 Tracking Position and Orientation

This section covers tracking of all 6 DOFs for a moving rigid body, with the most important case being head tracking. For convenience, we will refer to the position and orientation of a body as its *pose*. Six-DOF tracking enables $T_{eye}$ from 3.4 to be fully derived from sensor data, rather than inventing positions from plausible head models, as in (9.20). By estimating the position, the powerful depth cue of parallax becomes much stronger as the user moves his head from side to side. He could even approach a small object and look at it from any viewpoint, such as from above, below, or the sides. The methods in this section are also useful for tracking hands in space or objects that are manipulated during a VR experience.

**Why not just integrate the accelerometer?** It seems natural to try to accomplish 6-DOF tracking with an IMU alone. Recall from Figure 9.5 that the accelerometer measures the vector sum of true linear acceleration and acceleration due to gravity. If the gravity component is subtracted away from the output, as is heuristically accomplished for tilt correction, then it seems that the residual part is pure body acceleration. Why not simply integrate this acceleration twice to obtain position estimates? The trouble is that the drift error rate is much larger than in the case of a gyroscope. A simple calibration error leads to linear drift in the gyroscope case because it is the result of a single integration. After a double integration, a calibration error leads to quadratic drift error. This becomes unbearable in practice after a fraction of a second. Furthermore, the true body acceleration cannot be accurately extracted, especially when the body quickly rotates. Finally, as drift accumulates, what sensors can be used to detect the positional drift error? The IMU alone cannot help. Note that it cannot even distinguish motions at constant velocity, including zero motion; this is the same



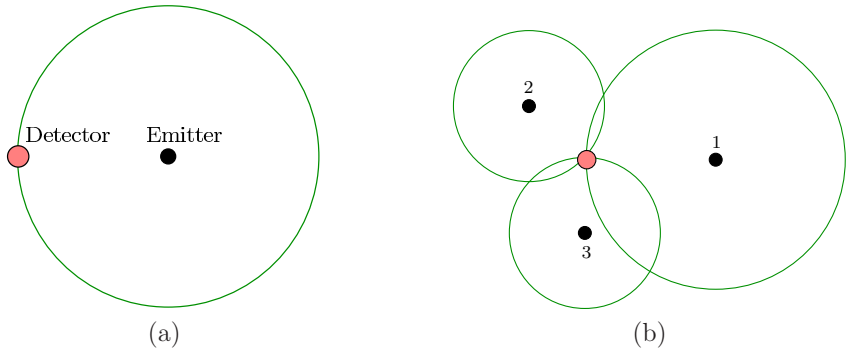



Figure 9.9: The principle of trilateration enables the detector location to be determined from estimates of distances to known emitter. A 2D example is shown: (a) from a single emitter, the detector could be anywhere along a circle; (b) using three emitters, the position is uniquely determined.

as our vestibular organs. Despite its shortcomings, modern IMUs remain an important part of 6-DOF tracking systems because of their high sampling rates and ability to accurately handle the rotational component.

**Make your own waves** The IMU approach was *passive* in the sense that it relied on sources of information that already exist in the environment. Instead, an *active* approach can be taken by transmitting waves into the environment. Since humans operate in the same environment, waves that are perceptible, such as light and sound, are not preferred. Instead, common energy sources in active tracking systems include infrared, ultrasound, and electromagnetic fields.

Consider transmitting an ultrasound pulse (above $20,000$ Hz) from a speaker and using a microphone to listen for its arrival. This is an example of an *emitter-detector pair*. The speaker is the emitter, and the microphone is the detector. If time measurement is synchronized between source and destination, then the *time of arrival* (*TOA* or *time of flight*) can be calculated. This is the time that it took for the pulse to travel the distance $d$ between the emitter and detector. Based on the known propagation speed in the medium (330 m/s for ultrasound), the distance $\hat{d}$ is estimated. One frustrating limitation of ultrasound systems is reverberation between surfaces, causing the pulse to be received multiple times at each detector.

When functioning correctly, the position of the detector could then be narrowed down to a sphere of radius $\hat{d}$, centered at the transmitter; see Figure 9.9(a). By using two transmitters and one microphone, the position is narrowed down to the intersection of two spheres, resulting in a circle (assuming the transmitter locations are known). With three transmitters, the position is narrowed down to

(a) (b)

Figure 9.10: (a) A magnetic dipole offers a field that varies its magnitude and direction as the position changes. (b) The Razer Hydra, a game controller that generated a weak magnetic field using a base station, enabling it to track its position.

two points, and with four or more transmitters, the position is uniquely determined.[1] The emitter and detector roles could easily be reversed so that the object being tracked carries the emitter, and several receivers are placed around it. The method of combining these measurements to determine position is called *trilateration*. If electromagnetic waves, such as radio, light, or infrared, are used instead of ultrasound, trilateration could still be applied even though the impossible to measure the propagation time directly. If the transmitter amplitude is known then distance can be estimated based on power degradation, rather than TOA. Alternatively, a time-varying signal can be emitted and its reflected phase shift can be estimated when the received signal is superimposed onto the transmitted signal.

If the detectors do not know the precise time that the pulse started, then they could compare differences in arrival times between themselves; this is called *time difference of arrival* (*TDOA*). The set of possible locations is a hyperboloid instead of a sphere. Nevertheless, the hyperboloid sheets can be intersected for multiple emitter-detector pairs to obtain the method of *multilateration*. This was used in the Decca Navigation System in World War II to locate ships and aircraft. This principle is also used by our ears to localize the source of sounds, which will be covered in Section **??**.

Finally, some methods could track position by emitting a complicated field that varies over the tracking area. For example, by creating a magnetic dipole,

[1]Global positioning systems (*GPS*) work in this way, but using radio signals, the Earth surface constraint, and at least one more satellite eliminate time synchronization errors.



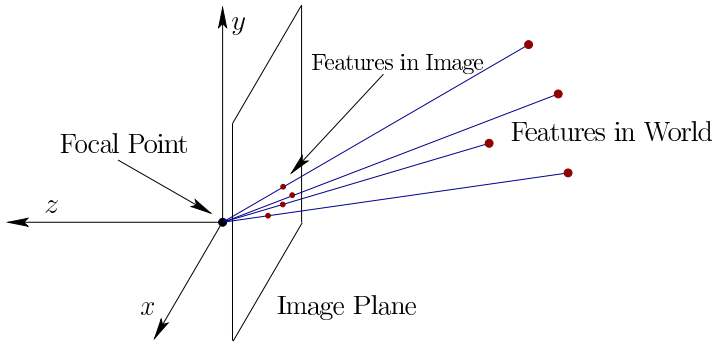



Figure 9.11: The real world contains special *features*, which are determined to lie along a line segment via perspective projection.

perhaps coded with a signal to distinguish it from background fields, the position and orientation of a body in the field could be distinguished in the field; see Figure 9.10(a). This principle was used for video games in the Razer Hydra tracking system in a base station that generated a magnetic field; see Figure 9.10(b). One drawback is that the field may become unpredictably warped in each environment, causing straight-line motions to be estimated as curved. Note that the requirements are the opposite of what was needed to use a magnetometer for yaw correction in Section 9.2; in that setting the field needed to be constant over the tracking area. For estimating position, the field should vary greatly in different locations.

**The power of visibility** The most powerful paradigm for 6-DOF tracking is *visibility*. The idea is to identify special parts of the physical world called *features* and calculate their position along a line-of-sight ray to a known location. Figure 9.11 shows an example inspired by a camera, but other methods could be used. One crucial aspect for tracking is *distinguishability*. If all features appear the same, then it may become difficult to determine and maintain "which is which" during the tracking process. Each feature should be assigned a unique label that is invariant over time, as rigid bodies in the world move. Confusing features with each other could cause catastrophically bad estimates to be made regarding the body pose.

The most common sensor used to detect features is a digital camera. Detecting, labeling, and tracking features are common tasks in computer vision or image processing. There are two options for features:

1. **Natural:** The features are automatically discovered, assigned labels, and maintained during the tracking process.

2. **Artificial:** The features are engineered and placed into the environment so

Figure 9.12: A sample QR code, which could be printed and used as an artificial feature. (Picture from Wikipedia.)

that they can be easily detected, matched to preassigned labels, and tracked.

Natural features are advantageous because there are no setup costs. The environment does not need to be engineered. Unfortunately, they are also much more unreliable. Using a camera, this is considered to be a hard computer vision problem because it may be as challenging as it is for the human visual system. For some objects, textures, and lighting conditions, it could work well, but it is extremely hard to make it work reliably for *all* possible settings. Therefore, artificial features are much more common.

For artificial features, one of the simplest solutions is to print a special tag onto the object to be tracked. For example, we could print bright red dots onto the object and easily scan for their appearance as red blobs in the image. To solve the distinguishability problem, we might have to use multiple colors, such as red, green, blue, and yellow dots. Trouble may occur if these colors exist naturally in other parts of the image. A more reliable method is to design a specific *tag* that is clearly distinct from the rest of the image. Such tags can be coded to contain large amounts of information, including a unique identification number. One of the most common coded tags is the *QR code*, an example of which is shown in Figure 9.12.

The features described so far are called *passive* because they do not emit energy. The hope is that sufficient light is in the world so that enough reflects off of the feature and enters the camera sensor. A more reliable alternative is to engineer *active* features that emit their own light. For example, colored LEDs can be mounted on the surface of a headset or controller. This comes at the expense of requiring a power source and increasing overall object cost and weight. Furthermore, its industrial design may be compromised because it would be lit up like a Christmas tree.

Fortunately, all of these tricks can be moved to the infrared (IR) part of the spectrum so that features are visible to cameras, but not to humans. Patterns can be painted onto objects that highly reflect IR energy. Alternatively, IR LEDs can be mounted onto devices. This is the case for Oculus Rift DK2 and later models, and the IR LEDs are even hidden behind plastic that is transparent for IR energy,




Figure 9.13: The Oculus Rift DK2 headset contained IR LEDs hidden behind IR-transparent plastic. (Photo by Paul Dempsey.)

but appears black to humans; see Figure 9.13.

In some settings, it might be difficult to mount LEDs on the objects, as in the case of tracking the subtle motions of an entire human body. This is called *MOCAP* or *motion capture*, which is described in Section 9.4. In MOCAP systems, powerful IR LEDs are positioned around the camera so that they illuminate *retroreflective markers* that are placed in the scene. Each marker can be imagined as a spherical mirror in the IR part of the spectrum. One unfortunate drawback is that the range is limited because IR energy must travel from the camera location to the target and back again. Since energy dissipates quadratically as function of distance, doubling the distance results on one-fourth of the energy level arriving at the camera.

At this point, it is natural to wonder why an entire image is being captured if the resulting image processing problem is trivial. The main reason is the proliferation of low-cost digital cameras and image processing software. Why not simply design an emitter-detector pair that produces a binary reading, indicating whether the visibility beam is occluded? This is precisely how the detection beam works in an automatic garage door system to ensure the door does not close on someone: An IR LED emits energy to a detection *photodiode*, which is essentially a switch that activate when it receives a sufficient level of energy for its target wavelength (in this case IR). To reduce the amount of energy dissipation, mirrors or lenses could be used to focus the energy.

Even better, an IR laser can be aimed directly at the detector. The next task is to use lenses and moving mirrors so that every detector that is visible from a fixed location will become illuminated at some point. The beam can be spread from a dot to a line using a lens, and then the line is moved through space using a spinning mirror. This is the basis of the *lighthouse tracking* system for the HTC

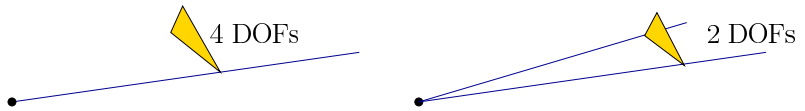



Figure 9.14: Each feature that is visible eliminates 2 DOFs. On the left, a single feature is visible, and the resulting rigid body has only 4 DOFs remaining. On the right, two features are visible, resulting in only 2 DOFs. This can be visualized as follows. The edge that touches both segments can be moved back and forth while preserving its length if some rotation is also applied. Rotation about an axis common to the edge provides the second DOF.

Vive VR headset, which is covered later in this section.

**The Perspective-n-Point (PnP) problem** A moving rigid body needs to be "pinned down" using $n$ observed features. This is called the *Perspective-n-Point* (or *PnP*) problem. We can borrow much of the math from Chapter 3; however, here we consider the placement of bodies in the *real* world, rather than the virtual world. Furthermore, we have an *inverse problem*, which is to determine the body placement based on points in the image. Up until now, the opposite problem was considered. For visual rendering in Chapter 7, an image was produced based on the known body placement in the (virtual) world.

The features could be placed on the body or in the surrounding world, depending on the sensing method. Suppose for now that they are on the body. Each feature corresponds to a point $p = (x, y, z)$ with coordinates defined in the frame of the body. Let $T_{rb}$ be a homogeneous transformation matrix that contains pose parameters, which are assumed to be unknown. Applying the transform $T_{rb}$ to the point $p_i$ as in (3.22) could place it anywhere in the real would. Recall the chain of transformations (3.40), which furthermore determines where each point on the body would appear in an image. The matrix $T_{eye}$ held the camera pose, whereas $T_{vp}$ and $T_{can}$ contained the perspective projection and transformed the projected point into image coordinates.

Now suppose that a feature has been observed to be at location $(i, j)$ in image coordinates. If $T_{rb}$ is unknown, but all other transforms are given, then there would be six independent parameters to estimate, corresponding to the 6 DOFs. Observing $(i, j)$ provides two independent constraints to the chain of transforms (3.40), one $i$ and one for $j$. The rigid body therefore loses 2 DOFs, as shown in Figure 9.14. This was the P1P problem because $n$, the number of features, is one.

The P2P problem corresponds to observing two features in the image and results in four constraints. In this case, each constraint eliminates two DOFs, resulting in only two remaining DOFs; see Figure 9.14. Continuing further, if three features are observed, then for the P3P problem, zero DOFs remain (except for the case in which collinear features are chosen on the body). It may seem that the problem is completely solved; however, zero DOFs allows for isolated point



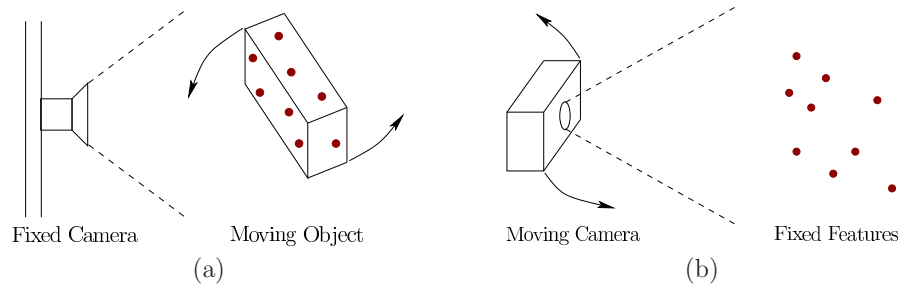



Figure 9.15: Two cases for camera placement: (a) A *world-fixed camera* is stationary, and the motions of objects relative to it are estimated using features on the objects. (b) An *object-fixed camera* is frequently under motion and features are ideally fixed to the world coordinate frame.

solutions. The P3P problem corresponds to trying to place a given triangle into a pyramid formed by rays so that each triangle vertex touches a different ray. This can be generally accomplished in four ways, which are hard to visualize. Imagine trying to slice a tall, thin pyramid (simplex) made of cheese so that four different slices have the exact same triangular size and shape. The cases of P4P and P5P also result in ambiguous solutions. Finally, in the case of P6P, unique solutions are always obtained if no four features are coplanar. All of the mathematical details are worked out in [31].

The PnP problem has been described in the ideal case of having perfect coordinate assignments to the feature points on the body and the perfect observation of those through the imaging process. In practice, small errors are made due to factors such as sensor noise, image quantization, and manufacturing tolerances. This results in ambiguities and errors in the estimated pose, which could deviate substantially from the correct answer [23]. Therefore, many more features may be used in practice to improve accuracy. Furthermore, a calibration procedure, such as *bundle adjustment* [11], may be applied before the device is used so that the feature point locations can be more accurately assigned before pose estimation is performed.

**Camera-based implementation** The visibility problem may be solved using a camera in two general ways, as indicated in Figure 9.15. Consider the *camera frame*, which is analogous to the eye frame from Figure 3.14 in Chapter 3. A *world-fixed camera* is usually stationary, meaning that the camera frame does not move relative to the world. A single transformation may be used to convert an object pose as estimated from the camera frame into a convenient world frame. For example, in the case of the Oculus Rift CV1, the head pose could be converted to a world frame in which the $-z$ direction is pointing at the camera, $y$ is "up", and

the position is in the center of the camera's tracking region or a suitable default based on the user's initial head position. For an *object-fixed camera*, the estimated pose, derived from features that remain fixed in the world, is the transformation from the camera frame to the world frame. This case would be obtained, for example, if we placed QR codes on the walls.

As in the case of an IMU, calibration is important for improving sensing accuracy. The following homogeneous transformation matrix can be applied to the image produced by a camera:

$$\begin{bmatrix} \alpha_x & \gamma & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{9.21}$$

The five variables appearing in the matrix are called *intrinsic parameters* of the camera. The $\alpha_x$ and $\alpha_y$ parameters handle scaling, $\gamma$ handles shearing, and $u_0$ and $v_0$ handle offset of the optical axis. These parameters are typically estimated by taking images of an object for which all dimensions and distances have been carefully measured, and performing least-squares estimation to select the parameters that reduce the sum-of-squares error (as described in Section 9.1). For a wide-angle lens, further calibration may be needed to overcome optical distortions (recall Section 7.3).

Now suppose that a feature as been observed in the image, perhaps using some form of *blob detection* to extract the pixels its image covers from the rest of the image []. This is easiest for a global shutter camera because all pixels will correspond to the same instant of time. In the case of a rolling shutter, the image may need to be transformed to undo the effects of motion (recall Figure 4.33). The location of the observed feature is calculated as a statistic of the blob pixel locations. Most commonly, the average over all blob pixels is used, resulting in non-integer image coordinates. Many issues affect performance: 1) quantization errors arise due to image coordinates for each blob pixel being integers; 2) if the feature does not cover enough pixels, then the quantization errors are worse; 3) changing in lighting conditions may make it difficult to extract the feature, especially in the case of natural features; 4) at some angles, two or more features may become close in the image, making it difficult to separate their corresponding blobs; 5) as various features enter or leave the camera view, the resulting estimated pose may jump.

**Laser-based implementation** By designing a special emitter-detector pair, the visibility problem can be accurately solved over great distances. This was accomplished by the *lighthouse tracking* system of the 2016 HTC Vive headset, and the *Minnesota scanner* from 1989 [24]. Figure 9.16 shows the lighthouse tracking hardware for the HTC Vive. The operation of a camera is effectively simulated, as shown in Figure 9.17(a).

If the base station were a camera, then the sweeping vertical stripe would correspond to estimating the row of the pixel that corresponds to the feature;



(a) (b)

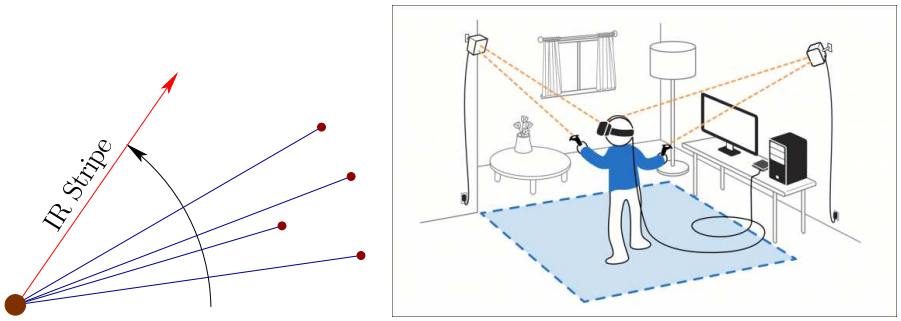
Figure 9.16: The laser-based tracking approach used in the HTC Vive VR headset: (a) A base station contains spinning drums that emit horizontal and vertical sheets of IR light. An array of IR LEDs appears in the upper left, which provide a synchronization flash. (b) Photodiodes in pockets on the front of the headset detect the incident IR light.

see Figure 9.17(a). Likewise, the sweeping horizontal stripe corresponds to the pixel column. The rotation rate of the spinning drum is known and is analogous to the camera frame rate. The precise timing is recorded as the beam hits each photodiode.

Think about polar coordinates (distance and angle) relative to the base station. Using the angular velocity of the sweep and the relative timing differences, the angle between the features as "observed" from the base station can be easily estimated. Although the angle between features is easily determined, their angles relative to some fixed direction from the base station must be determined. This is accomplished by an array of IR LEDs that are pulsed on simultaneously so that all photodiodes detect the flash (visible in Figure 9.17(a)). This could correspond, for example, to the instant of time at which each beam is at the 0 orientation. Based on the time from the flash until the beam hits a photodiode, and the known angular velocity, the angle of the observed feature is determined. To reduce temporal drift error, the flash may be periodically used during operation.

As in the case of the camera, the distances from the base station to the features are not known, but can be determined by solving the PnP problem. Multiple base stations can be used as well, in a way that is comparable to using multiple cameras or multiple eyes to infer depth. The result is accurate tracking over a large area, as shown in Figure 9.17(b).

**Filtering** As in Section 9.2, outputs from sensors are combined over time by a filtering method to maintain the estimate. In the current setting, the pose can be maintained by combining both visibility information and outputs of an IMU.




(a) (b)

Figure 9.17: (a) This is a 2D view of the angular sweep of the IR stripe in the laser-based tracking approach (as in HTC Vive). This could correspond to a top-down view, in which a vertical stripe spins with a yaw rotation about the base. In this case, the angular locations in the horizontal direction are observed, similar to column coordinates of a camera image. This could also correspond to a side view, in which case the vertical stripe spins with a pitch rotation and the angular locations in the vertical direction. As the beam hits the features, which are photodiodes, the direction is known because of the spinning rate and time since the synchronization flash. (b) By putting two case stations on top of poles at the corners of the tracking area, a large region can be accurately tracked for a headset and controllers. (Drawing by Chris Stobing.)

For the orientation component of the pose, the complementary filter from (9.10) could be used. The camera provides an additional source for detecting orientation drift error. The camera optical axis is a straightforward reference for yaw error estimation detection, which makes it a clear replacement for the magnetometer. If the camera tilt is known, then the camera can also provide accurate tilt error estimation.

The IMU was crucial for obtaining highly accurate orientation tracking because of accurate, high-frequency estimates of angular velocity provided by the gyroscope. If the frame rate for a camera or lighthouse system is very high, then sufficient sensor data may exist for accurate position tracking; however, it is preferable to directly measure derivatives. Unfortunately, IMUs do not measure linear velocity. However, the output of the linear accelerometer could be used as suggested in the beginning of this section. Suppose that the accelerometer estimates the body acceleration as

$$\hat{a}[k] = (\hat{a}_x[k], \hat{a}_y[k], \hat{a}_z[k]) \tag{9.22}$$

in the world frame (this assumes the gravity component has been subtracted from the accelerometer output).



By numerical integration, the velocity $\hat{v}[k]$ can be estimated from $\hat{a}[k]$. The position $\hat{p}[k]$ is estimated by integrating the velocity estimate. The update equations using simple Euler integration are

$$\begin{aligned} \hat{v}[k] &= \hat{a}[k]\Delta t + \hat{v}[k-1] \\ \hat{p}[k] &= \hat{v}[k]\Delta t + \hat{p}[k-1]. \end{aligned} \tag{9.23}$$

Note that each equation actually handles three components, $x$, $y$, and $z$, at the same time. The accuracy of the second equation can be further improved by adding $\frac{1}{2}\hat{a}[k]\Delta t^2$ to the right side.

As stated earlier, double integration of the acceleration leads to rapidly growing position drift error, denoted by $\hat{d}_p[k]$ . The error detected from PnP solutions provide an estimate of $\hat{d}_p[k]$, but perhaps at a much lower rate than the IMU produces observations. For example, a camera might take pictures at 60 FPS and the IMU might report accelerations at 1000 FPS.

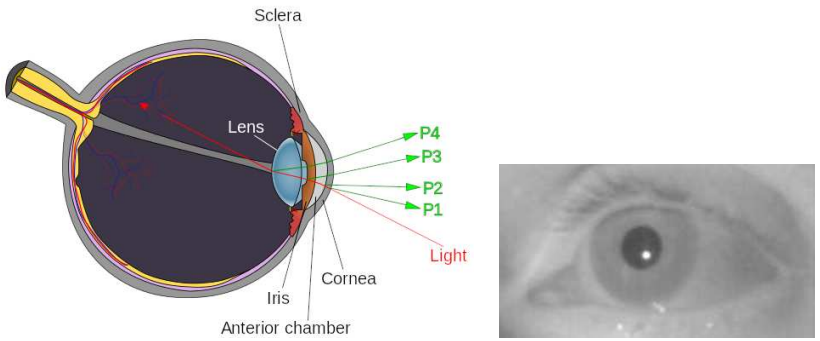
The complementary filter from (9.10) can be extended to the case of double integration to obtain

$$\begin{aligned} p_c[k] &= \alpha_p \hat{d}_p[k] + (1-\alpha_p)\hat{p}[k] \\ v_c[k] &= \alpha_v \hat{d}_p[k] + (1-\alpha_v)\hat{v}[k]. \end{aligned} \tag{9.24}$$

Above, $p_c[k]$ and $v_c[k]$ are the corrected position and velocity, respectively, which are each calculated by a complementary filter. The estimates $\hat{p}[k]$ and $\hat{v}[k]$ are calculated using (9.23). The parameters $\alpha_p$ and $\alpha_v$ control the amount of importance given to the drift error estimate in comparison to IMU updates.

Equation (9.24) is actually equivalent to a *Kalman filter*, which is the optimal filter (providing the most accurate estimates possible) for the case of a linear dynamical system with Gaussian noise, and sensors that also suffer from Gaussian noise. Let $\omega_d^2$ represent the variance of the estimated Gaussian noise in the dynamical system, and let $\omega_s^2$ repesent the sensor noise variance. The complementary filter (9.24) is equivalent to the Kalman filter if the parameters are chosen as $\alpha_1 = \sqrt{2\omega_d/\omega_s}$ and $\alpha_2 = \omega_d/\omega_s$ [12]. A large variety of alternative filtering methods exist; however, the impact of using different filtering methods is usually small relative to calibration, sensor error models, and dynamical system models that are particular to the setup. Furthermore, the performance requirements are mainly *perceptually* based, which could be different than the classical criteria around which filtering methods were designed.

Once the filter is running, its pose estimates can be used to aid the PnP problem. The PnP problem can be solved incrementally by perturbing the pose estimated by the filter, using the most recent accelerater outputs, so that the observed features are perfectly matched. Small adjustments can be made to the pose so that the sum-of-squares error is reduced to an acceptable level. In most case, this improves reliability when there are so few features visible that the PnP problem has ambiguous solutions. Without determining the pose incrementally, a catastrophic jump to another PnP solution might occur.




(a) (b)

Figure 9.18: (a) The first and sometimes the fourth Purkinje images of an IR light source are used for eye tracking. (Figure from Wikipedia.) (b) The first Purkinje image generates a bright reflection as shown. (Picture from Massimo Gneo, Maurizio Schmid, Silvia Conforto, and Tomasso D'Alessio.)

## 9.4 Tracking Attached Bodies

Many tracking problems involve estimating the motion of one body relative to another. For example, an eye rotates inside of its socket, which is part of the skull. Although the eye may have six DOFs when treated as a rigid body in space, its position and orientation are sufficiently characterized with two or three parameters once the head pose is given. Other examples include the head relative to the torso, a hand relative to the wrist, and the tip of a finger relative to its middle bone. The entire human body can even be arranged into a tree of attached bodies, based on a skeleton. Furthermore, bodies may be attached in a similar way for other organisms, such as dogs or monkeys, and machinery, such as robots or cars. In the case of a car, the wheels rotate relative to the body. The result is a *multibody system*. The mathematical characterization of the poses of bodies relative to each other is called *multibody kinematics*, and the full determination of their velocities and accelerations is called *multibody dynamics*.

**Eye tracking** Eye tracking systems been used by vision scientists for over a century to study eye movements. Three main uses for VR are: 1) To accomplish foveated rendering, as mentioned in Section 5.4, so that high-resolution rendering need only be performed for the part of the image that lands on the fovea. 2) To study human behavior by recording tracking data so that insights may be gained into VR sickness, attention, and effectiveness of experiences. 3) To render the eye orientations in VR so that social interaction may be improved by offering eye-contact and indicating someone's focus of attention; see Section 10.4.

Three general categories of eye-tracking approaches have been developed. The



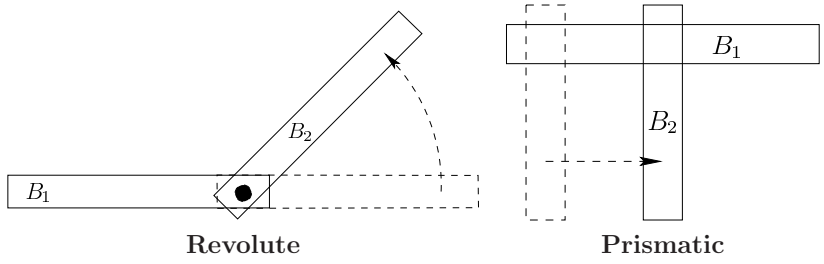



Figure 9.19: Two types of 2D joints: a revolute joint allows one link to rotate with respect to the other, and a prismatic joint allows one link to translate with respect to the other.

first is *electro-oculography* (*EOG*), which obtains measurements from several electrodes placed on the facial skin around each eye. The recorded potentials correspond to eye muscle activity, from which the eye orientation relative to the head is determined through filtering. The second approach uses a contact lens, which contains a tiny magnetic coil that causes a potential change in a surrounding electromagnetic field. The third approach is called *video-oculography* (*VOG*), which shines IR light onto the eye and senses its *corneal reflection* using a camera or photodiodes. The reflection is based on *Purkinje images*, as shown in Figure 9.18. Because of its low cost and minimal invasiveness, this is the most commonly used method today. The contact lens approach is the most accurate; however, it is also the most uncomfortable.

**Forward kinematics** Suppose that an eye tracking method has estimated the eye orientation relative to the human skull and it needs to be placed accordingly in the virtual world. This transformation must involve a combination of the head and eye transforms. For a more complicated problem, consider placing the right index finger in the world by using pose of the torso along with all of the angles formed between bones at each joint. To understand how these and other related problems are solved, it is helpful to first consider 2D examples.

Each body of a multibody system is called a *link*, and a pair of bodies are attached at a *joint*, which allows one or more DOFs of motion between them. Figure 9.19 shows two common ways that one planar body might move while attached to another. The *revolute joint* is most common and characterizes the motion allowed by a human elbow.

Consider defining a chain of $m$ links, $B_1$ to $B_m$, and determining the location of a point on the last link. The points on each link are defined using coordinates of its own *body frame*. In this frame, the body appears as shown for $B_{i-1}$ in Figure 9.20, with the origin at the joint that connects $B_{i-1}$ to $B_{i-2}$ and the $x$ axis pointing through the joint that connects $B_{i-1}$ to $B_i$. To move the points on $B_i$ to

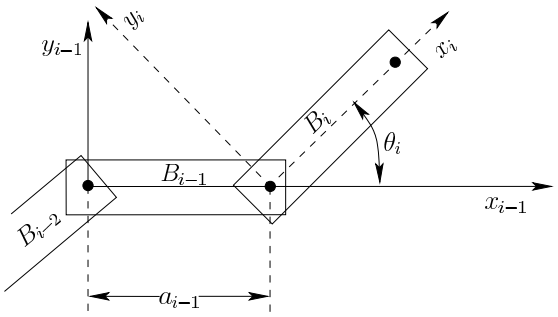



Figure 9.20: The body frame of each $B_i$, for $1 < i < m$, is based on the joints that connect $B_i$ to $B_{i-1}$ and $B_{i+1}$.

the proper location in the body frame of $B_{i-1}$, the homogeneous transform

$$T_i = \begin{pmatrix} \cos\theta_i & -\sin\theta_i & a_{i-1} \\ \sin\theta_i & \cos\theta_i & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{9.25}$$

is applied. This rotates $B_i$ by $\theta_i$, and then translates it along the $x$ axis by $a_{i-1}$. For a revolute joint, $\theta_i$ is a variable, and $a_{i-1}$ is a constant. For a prismatic joint, $\theta_i$ is constant and $a_{i-1}$ is a variable.

Points on $B_i$ are moved into the body frame for $B_1$ by applying the product $T_2 \cdots T_i$. A three-link example is shown in Figure 9.21. To move the first link $B_1$ into the world frame, a general 2D homogeneous transform can be applied:

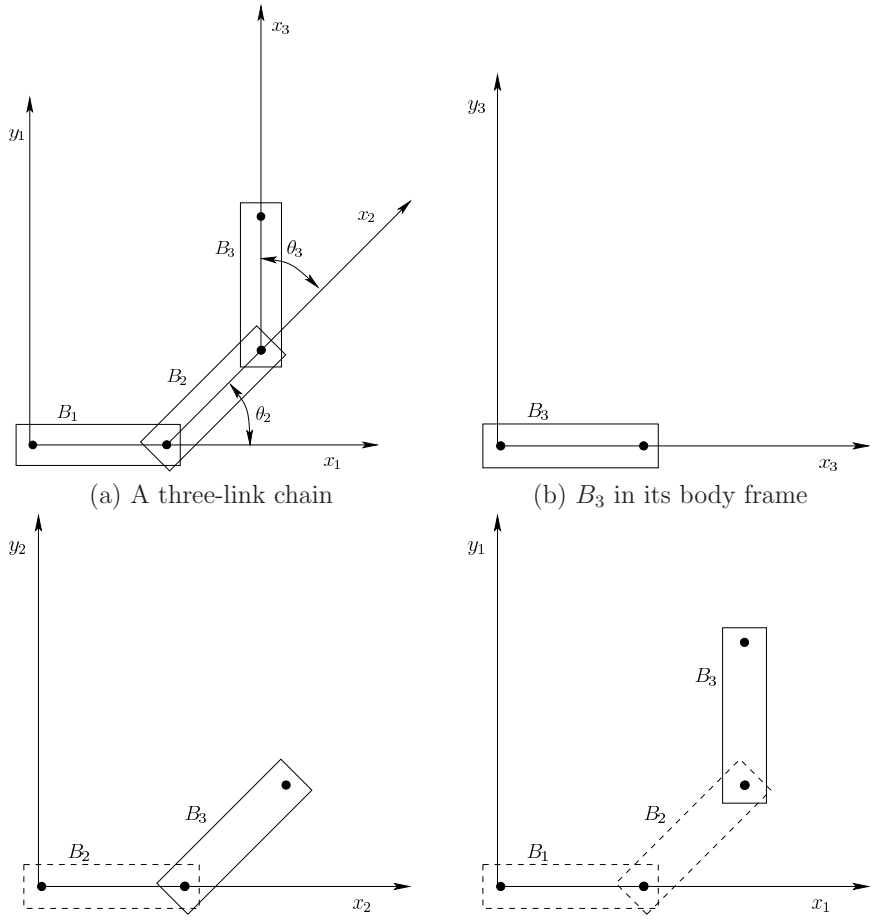
$$T_1 = \begin{pmatrix} \cos\theta_i & -\sin\theta_i & x_t \\ \sin\theta_i & \cos\theta_i & y_t \\ 0 & 0 & 1 \end{pmatrix}. \tag{9.26}$$

This transform is simply added to the matrix product to move each $B_i$ by applying $T_1 T_2 \cdots T_i$.

A chain of 3D links is handled in the same way conceptually, but the algebra becomes more complicated. See Section 3.3 of [17] for more details. Figure 9.22 shows six different kinds of joints that are obtained by allowing a pair of 3D links to slide against each other. Each link is assigned a convenient coordinate frame based on the joints. Each homogeneous transform $T_i$ contains a mixture of constants and variables in which the variables correspond to the freedom allowed by the joint. The most common assignment scheme is called *Denavit-Hartenberg parameters* [10]. In some settings, it might be preferable to replace each $T_i$ by a parameterized quaternion that rotates the body, followed by a simple addition that translates the body.

A tree of links may also be considered; a common example is a human torso serving as the root, with a head, two arms, and two legs being chains that extend






(a) A three-link chain

(b) $B_3$ in its body frame

(c) $T_3$ puts $B_3$ in $B_2$'s body frame

(d) $T_2T_3$ puts $B_3$ in $B_1$'s body frame

Figure 9.21: Applying the transformation $T_2T_3$ to the model of $B_3$. If $T_1$ is the identity matrix, then this yields the location in the virtual world of points in $B_3$.

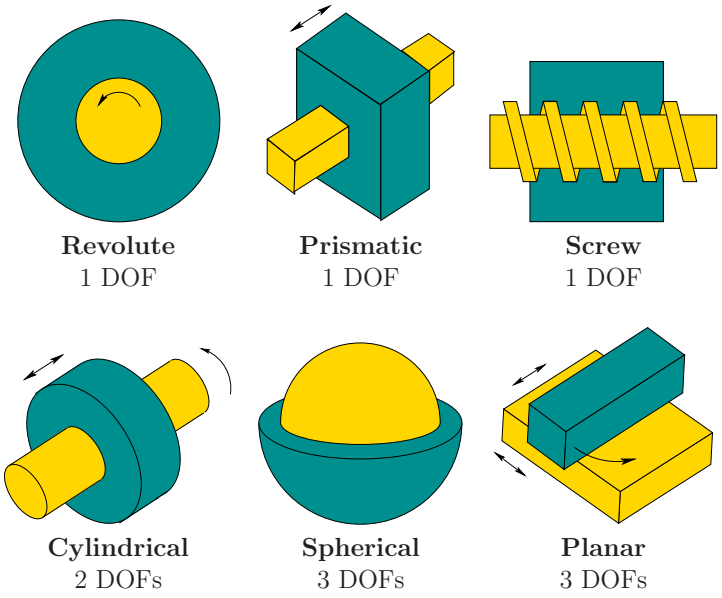



Figure 9.22: Types of 3D joints arising from the 2D surface contact between two bodies.

from it. The human hand is another example. Coordinate frames in this case are often assigned using *Kleinfinger-Khalil parameters* [14].

**Inverse kinematics** Recall the PnP problem from Section 9.3, which involved calculating the pose of a body based on some observed constraints. A similar problem is to determine the joint parameters for a chain of bodies by observing constraints on the last body. A common example is to calculate the poses of the arm links by using only the pose of the hand. This is generally called the *inverse kinematics problem* (see [1] and Section 4.4 of [17]). As in the case of PnP, the number of solutions may be infinite, finite, one, or zero. Some 2D examples are shown in Figure 9.23. Generally, if the last link is constrained, the freedom of motion for the intermediate links increases as the number of links increases. The *Chebychev-Grübler-Kutzbach criterion* gives the number of DOFs, assuming the links are not in some special, singular configurations [2]. A common problem in animating video game characters is to maintain a kinematic constraint, such as the hand grasping a doorknob, even though the torso or door is moving. In this case, *iterative optimization* is often applied to perturb each joint parameter until the error is sufficiently reduced. The error would measure the distance between the hand and the doorknob in our example.



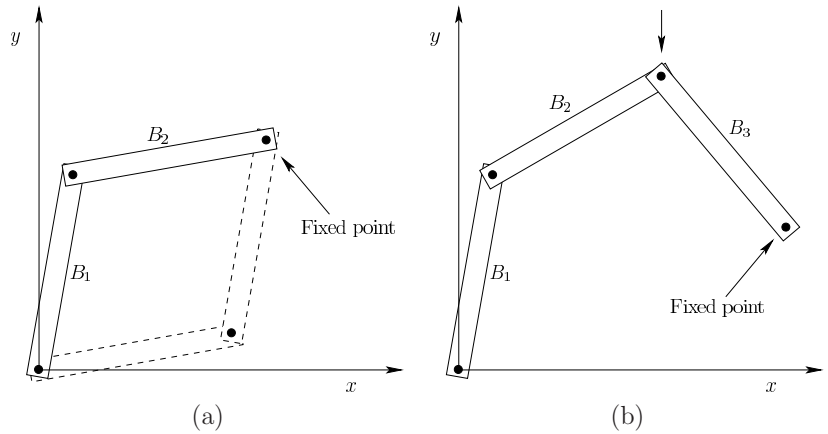



Figure 9.23: (a) The orientations of both links can be inferred from the position of the fixed point; however, there is a second solution if the angles are not restricted. (b) In the case of three links, a one-dimensional family of solutions exists when the end is fixed. This can be visualized by pushing down on the top joint, which would cause $B_1$ to rotate counter-clockwise. This is equivalent to the classical *four-bar mechanism*, which was used to drive the wheels of a steam engine (the fourth "link" is simply the fixed background).

**Motion capture systems** Tracking systems for attached bodies use kinematic constraints to improve their accuracy. The most common application is tracking the human body, for which the skeleton is well-understood in terms of links and joints [32]. Such motion capture systems have been an important technology for the movie industry as the motions of real actors are brought into a virtual world for animation. Figure 9.24 illustrates the operation. Features, of the same kind as introduced in Section 9.3, are placed over the body and are visible to cameras mounted around the capture studio. The same options exist for visibility, with the most common approach over the past decade being to use cameras with surrounding IR LEDs and placing retroreflective markers on the actor.

To obtain a unique pose for each body part, it might seem that six features are needed (recall P6P from Section 9.3); however, many fewer are sufficient because of kinematic constraints. Additional features may nevertheless be used if the goal is to also capture skin motion as it moves along the skeleton. This is especially important for facial movement. Many new MOCAP technologies are currently under development. For example, a system developed by Noitom captures human body movement solely by placing IMUs on the body. Some systems capture motion by cameras alone, as in the case of Leap Motion (see Figure 9.25) for hand tracking, and systems by Microsoft and 8i for full-body tracking by extracting contours against a green screen. Solutions based on modern depth sensors may

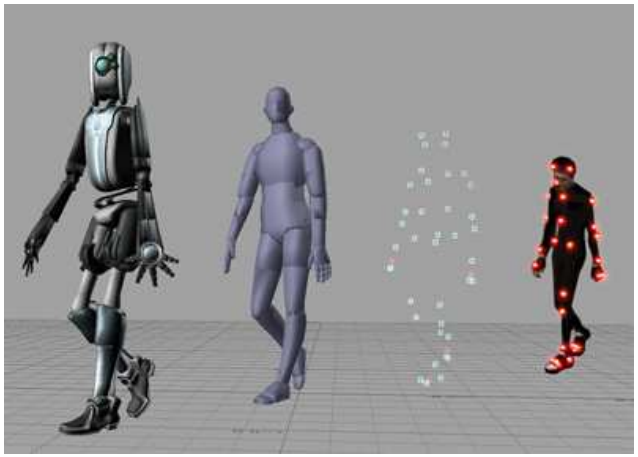


Figure 9.24: With a *motion capture* (*MOCAP*) system, artificial features are placed around the body of a human actor. The motions are extracted and matched to a kinematic model. Each rigid body in the model has an associated geometric model that is rendered to produce the final animated character. (Picture from Wikipedia user Hipocrite.)

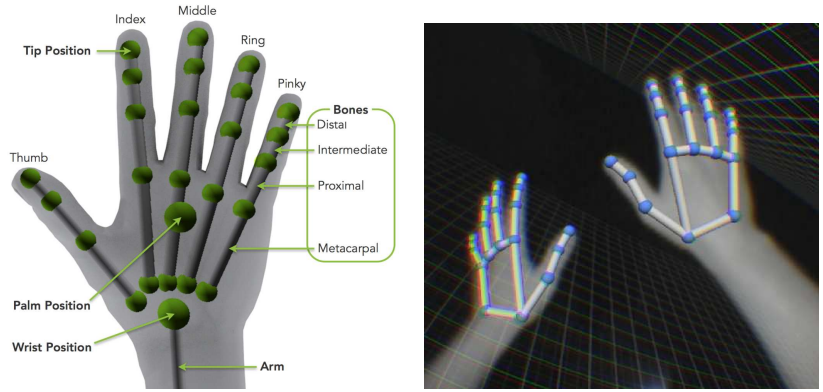
also become prevalent in the near future. One challenge is to make highly accurate and reliable systems for low cost and installation effort.

## 9.5 3D Scanning of Environments

Up until now, this chapter has described how to use sensors to track the motions of one or more rigid bodies. By contrast, this section describes how sensors are used to build geometric models of rigid bodies. These could be movable or stationary models, as introduced in Section 3.1. A movable model typically corresponds to an object that is be manipulated by the user, such as a sword, hammer, or coffee cup. These models are often built from a *3D scanner*, which images the object from many viewpoints in a controlled way. The object may be placed on a turntable that is surrounded by cameras and other sensors, or the sensors may move around while the object remains stationary; see Figure 9.26(a).

**SLAM** A 3D scanner is useful for smaller objects, with surrounding sensors facing inward. For larger objects and stationary models, the sensors are usually inside facing out; see Figure 9.26(b). A common example of a stationary model is the inside of a building. Scanning such models is becoming increasingly important for surveying and forensics. This is also the classical robotics problem of *mapping*, in which a robot carrying sensors builds a 2D or 3D representation of its world






(a) (b)

Figure 9.25: (a) The hand model used by Leap Motion tracking. (b) The tracked model superimposed in an image of the actual hands.

for the purposes of navigation and collision avoidance. Robots usually need to estimate their locations based on sensors, which is called the *localization* problem. Robot localization and tracking bodies for VR are fundamentally the same problems, with the main distinction being that known motion commands are given to robots, but the corresponding human intent is not directly given. Robots often need to solve mapping and location problems at the same time, which results in the *simultaneous localization and mapping* problem; the acronym *SLAM* is widely used. Due to the similarity of localization, mapping, and VR tracking problems, deep connections exist between robotics and VR. Therefore, many mathematical models, algorithms, and sensing technologies overlap.

Consider the possible uses of a large, stationary model for VR. It could be captured to provide a virtual world in which the user is placed at the current time or a later time. Image data could be combined with the 3D coordinates of the model, to produce a photorealistic model (recall Figure 2.14 from Section 2.2). This is achieved by texture mapping image patches onto the triangles of the model.

**Live capture of the current location** Rather than capturing a world in which to transport the user, sensors could alternatively be used to capture the physical world where the user is currently experiencing VR. This allows obstacles in the matched zone to be rendered in the virtual world, which might be useful for safety or to improve interactivity. For safety, the boundaries of the matched zone could be rendered to indicate that the user is about to reach the limit. Hazards such as a hot cup of coffee or a toddler walking across the matched zone could be indicated. Interactivity can be improved by bringing fixed objects from the physical world


(a) (b)

Figure 9.26: (a) The Afinia ES360 scanner, which produces a 3D model of an object while it spins on a turntable. (b) The FARO Focus3D X 330 is an outward-facing scanner for building accurate 3D models of large environments; it includes a GPS receiver to help fuse individual scans into a coherent map.

into the virtual world. For example, if the user is sitting in front of a desk, then the desk can be drawn in the virtual world. If she touches the virtual desk, she will feel the desk pushing back. This is an easy way to provide touch feedback in VR.

**Are panoramas sufficient?** Before embarking on the process of creating a large, detailed map of a surrounding 3D world, it is important to consider whether it is necessary. As mentioned in Section 7.5, panoramic images and videos are becoming increasingly simple to capture. In some applications, it might be sufficient to build an experience in which the user is transported between panoramas that were captured from many locations that are close to each other.

**The main ingredients** Building a 3D model from sensor data involves three important steps:

1. Extracting a 3D *point cloud* from a fixed location.
2. Combining point clouds from multiple locations.
3. Converting a point cloud into a mesh of triangles.

For the first step, a sensor is placed at a fixed position and orientation while 3D points are extracted. This could be accomplished in a number of ways. In theory, any of the depth cues from Section 6.1 can be applied to camera images to



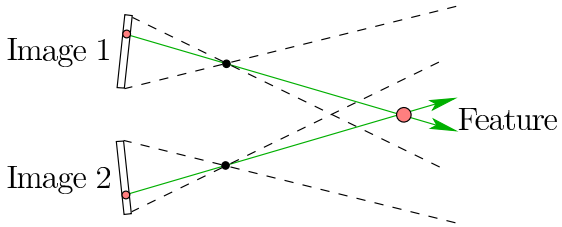



Figure 9.27: By using two cameras, *stereo vision* enables the location of a feature in the 3D world to be determined by intersecting the corresponding project ray from each camera. To accomplish this, the camera calibration parameters and relative poses must be known. Similarly, one camera could be replaced by a laser that illuminates the feature so that it is visible to the remaining camera. In either case, the principle is to intersect two visibility rays to obtain the result.

extract 3D points. Variations in focus, texture, and shading are commonly used in computer vision as monocular cues. If two cameras are facing the same scene and their relative positions and orientations are known, then binocular cues are used to determine depth. By identifying the same natural feature in both images, the corresponding visibility rays from each image are intersected to identify a point in space; see Figure 9.27. As in Section 9.3, the choice between natural and artificial features exists. A single camera and an IR projector or laser scanner may be used in combination so that depth is extracted by identifying where the lit point appears in the image. This is the basis of the Microsoft Kinect sensor (recall Figure 2.10 from Section 2.1). The resulting collection of 3D points is often called a *point cloud*.

In the second step, the problem is to merge scans from multiple locations. If the relative position and orientation of the scanner between scans is known, then the problem is solved. In the case of the object scanner shown in Figure 9.26(a), this was achieved by rotating the object on a turntable so that the position remains fixed and the orientation is precisely known for each scan. Suppose the sensor is instead carried by a robot, such as a drone. The robot usually maintains its own estimate of its pose for purposes of collision avoidance and determining whether its task is achieved. This is also useful for determining the pose that corresponds to the time at which the scan was performed. Typically, the pose estimates are not accurate enough, which leads to an optimization problem in which the estimated pose is varied until the data between overlapping scans nicely aligns. The *estimation-maximization (EM) algorithm* is typically used in this case, which incrementally adjusts the pose in a way that yields the maximum likelihood explanation of the data in a statistical sense. If the sensor is carried by a human, then extra sensors may be included with the scanning device, as in the case of GPS for the scanner in Figure 9.26(b); otherwise, the problem of fusing data from multiple scans could become too difficult.

In the third stage, a large point cloud has been obtained and the problem is to generate a clean geometric model. Many difficulties exist. The point density may vary greatly, especially where two or more overlapping scans were made. In this case, some points may be discarded. Another problem is that outliers may exist, which correspond to isolated points that are far from their correct location. Methods are needed to detect and reject outliers. Yet another problem is that large holes or gaps in the data may exist. Once the data has been sufficiently cleaned, surfaces are typically fit to the data, from which triangular meshes are formed. Each of these problems is a research area in itself. To gain some familiarity, consider experimenting with the open-source *Point Cloud Library*, which was developed to handle the operations that arise in the second and third stages. Once a triangular mesh is obtained, texture mapping may also be performed if image data is also available. One of the greatest challenges for VR is that the resulting models often contain numerous flaws which are much more noticeable in VR than on a computer screen.

## Further Reading

Need IMU calibration papers.

Mag calibration: [9, 15, 27].

Complementary filters over rotation group, with convergence proofs: [22].

Why you might as well use a complementary filter if your system model is just a double integrator: [12].

Fusion of IMU and Vision for Absolute Scale Estimation: Nutzi, Scaramuzza, Weiss, Siegwart.

Oculus VR blogs: [20, 18, 19]

Oculus Rift tracking: [21]

PnP with ambiguities analyzed: [23, 31]

Fast PnP, linear in number of points: [33]

Bundle adjustment: [26]

RANSAC for robust outlier rejection: [8]

Old but good VR tracking survey: [30].

Survey of human body tracking [34]

Eye tracking references: [6, 28]

Kinematics: [1, 3]; [5]

SLAM ([25]; Section 12.3.5 of LaValle 06)

Filtering or sensor fusion in the larger context can be characterized in terms of *information spaces* (Chapter 11 of [17]).

# Bibliography


[1] J. Angeles. *Spatial Kinematic Chains. Analysis, Synthesis, and Optimisation*. Springer-Verlag, Berlin, 1982.

[2] J. Angeles. *Rotational Kinematics*. Springer-Verlag, Berlin, 1989.

[3] J. Angeles. *Fundamentals of Robotic Mechanical Systems: Theory, Methods, and Algorithms*. Springer-Verlag, Berlin, 2003.

[4] C. K. Chui and G. Chen. *Kalman Filtering*. Springer-Verlag, Berlin, 1991.

[5] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms*. Springer-Verlag, Berlin, 1992.

[6] A. T. Duchowski. *Eye Tracking Methodology: Theory and Practice, 2nd Ed.* Springer-Verlag, Berlin, 2007.

[7] J. Favre, B. M. Jolles, O. Siegrist, and K. Aminian. Quaternion-based fusion of gyroscopes and accelerometers to improve 3D angle measurement. *Electronics Letters*, 32(11):612–614, 2006.

[8] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[9] D. Gebre-Egziabher, G. Elkaim, J. David Powell, and B. Parkinson. Calibration of strapdown magnetometers in magnetic field domain. *Journal of Aerospace Engineering*, 19(2):87–102, 2006.

[10] R. S. Hartenberg and J. Denavit. A kinematic notation for lower pair mechanisms based on matrices. *Journal of Applied Mechanics*, 77:215–221, 1955.

[11] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision, 2nd Ed.* Cambridge University Press, Cambridge, U.K., 2004.

[12] W. T. Higgins. A comparison of complementary and Kalman filtering. *IEEE Transactions on Aerospace and Electronic Systems*, 11(3):321–325, 1975.

[13] R. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME, Journal of Basic Engineering*, 82:35–45, 1960.







[14] W. Khalil and J. F. Kleinfinger. A new geometric notation for open and closed-loop robots. In *Proceedings IEEE International Conference on Robotics & Automation*, volume 3, pages 1174–1179, 1986.

[15] C. Konvalin. Compensating for tilt, hard-iron, and soft-iron effects. Available at http://www.sensorsmag.com/sensors/motion-velocity-displacement/compensating-tilt-hard-iron-and-soft-iron-effects-6475, December 2009. Last retrieved on May 30, 2016.

[16] P. R. Kumar and P. Varaiya. *Stochastic Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1986.

[17] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[18] S. M. LaValle. Help! My cockpit is drifting away. Oculus blog post. Retrieved from https://developer.oculus.com/blog/magnetometer/, December 2013. Last retrieved on Jan 10, 2016.

[19] S. M. LaValle. The latent power of prediction. Oculus blog post. Retrieved from https://developer.oculus.com/blog/the-latent-power-of-prediction/, July 2013. Last retrieved on Jan 10, 2016.

[20] S. M. LaValle. Sensor fusion: Keeping it simple. Oculus blog post. Retrieved from https://developer.oculus.com/blog/sensor-fusion-keeping-it-simple/, May 2013. Last retrieved on Jan 10, 2016.

[21] S. M. LaValle, A. Yershova, M. Katsev, and M. Antonov. Head tracking for the Oculus Rift. In *Proc. IEEE International Conference on Robotics and Automation*, pages 187–194, 2014.

[22] R. Mahoney, T. Hamel, and J.-M. Pfimlin. Nonlinear complementary filters on the special orthogonal group. *IEEE Transactions on Automatic Control*, 53(5):1203–1218, 2008.

[23] G. Schweighofer and A. Pinz. Robust pose estimation from a planar target. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(12):2024–2030, 2006.

[24] B. R. Sorensen, M. Donath, G.-B. Yanf, and R. C. Starr. The minnesota scanner: A prototype sensor for three-dimensional tracking of moving body segments. *IEEE Transactions on Robotics*, 5(4):499–509, 1989.

[25] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, Cambridge, MA, 2005.

[26] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzbiggon. Bundle adjustment - a modern synthesis. In *Proceedings IEEE International Workshop on Vision Algorithms*, pages 298–372, 1999.

[27] J. F. Vasconcelos, G. Elkaim, C. Silvestre, P. Oliveira, and B. Cardeira. Geometric approach to strapdown magnetometer calibration in sensor frame. *Transactions on Aerospace and Electronic Systems*, 47(2):1293–1306, 2011.

[28] X. Wang and B. Winslow. Eye tracking in virtual environments. In K. S. Hale and K. M. Stanney, editors, *Handbook of Virtual Environments, 2nd Edition*. CRC Press, Boca Raton, FL, 2015.

[29] D. S. Watkins. *Fundamentals of Matrix Computations*. Wiley, Hoboken, NJ, 2002.

[30] G. Welch and E. Foxlin. Motion tracking: no silver bullet, but a respectable arsenal. *IEEE Computer Graphics and Applications*, 22(6):24–28, 2002.

[31] Y. Wu and Z. Hu. PnP problem revisited. *Journal of Mathematical Imaging and Vision*, 24(1):131–141, 2006.

[32] V. M. Zatsiorsky and B. I. Prilutsky. *Biomechanics of Skeletal Muscles*. Human Kinetics, Champaign, IL, 2012.

[33] Y. Zheng, Y. Kuang, S. Sugimoto, and K. Aström. Revisiting the pnp problem: A fast, general and optimal solution. In *Proceedings IEEE International Conference on Computer Vision*, pages 2344–2351, 2013.

[34] H. Zhou and H. Hu. Human motion tracking for rehabilitation - A survey. *Biomedical Signal Processing and Control*, 3(1):1–18, 2007.