

HYPEREAL

Open Source Document

|  |  |  |
| --- | --- | --- |
| Date | Modified by | Comments |
| 2017-02-10 | Hypereal | Init Version |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

This document is provided as part of due diligence materials for Soft-Bank China. Note that all the contents are Hypereal Confidential. If you are not the target reader, please do not read and delete the copy immediately. Hypereal reserves all the rights for situations including but not limit to mis-use or intentionally disclosure of this document.

# Tracking Algorithm

In this chapter, Hypereal’s tracking algorithms are introduced. Given lighthouse data and devices’ IMU (Inertial Measurement Unit) data, the tracking algorithm transforms raw data into device’s pose in real world.

The tracking algorithm uses some techniques in computer vision and graphics, and some mathematical methods to solve tracking problem, finally turns the input into device’s pose – translation and rotation, in six degrees of freedom. The device’s pose could be directly applied into 3D engine as game controllers and cameras’ transform.

## System Design

The tracking algorithm is a component of Hypereal tracking system. It is compiled as a shared library (DLL in Windows), and loaded by other component.

The tracking algorithm provides a simple interface, *TrackingObjectManager*, as an input/output port receiving hardware’s sensor data and reporting tracking results.

This component is made of three abstract modules – driver, pre-processing and tracking module.

The figure below describes the pipeline of these three modules.



### Driver Module

Driver module is not a part of tracking algorithm, but an important row providing light sensor and IMU data.

Driver module is in charge of communication with hardware via USB, maintaining device status and forwarding sensor data to tracking algorithm, fetching tracking result and reporting to other components.

When communicating with tracking algorithm, driver module maintains two [ringbuffers](https://en.wikipedia.org/wiki/Circular_buffer), as channels of transferring IMU and light sensor packets.

The format and structure of the two types of packets is described in hardware chapter.

### Pre-processing Module

Obviously, the raw sensor data cannot be directly put into pose solving. They are processed first and converted into some physical data. The conversion contains:

1. IMU data:
   1. Convert accelerometer’s data into acceleration, in unit of mm/s2.
   2. Convert gyroscope’s data into angular velocity, in unit of radian/s.
   3. Convert magnetometer’s data into magnetic field intensity, in unit of µT.
2. Light sensor data:
   1. Convert each light sensor’s time ticks, into angles relative to lighthouse.
   2. Aggregate two axes of a lighthouse into *HitMap*, representing which light sensor is hit by which lighthouse, with X and Y angle.

The *HitMap* data structure contains information about light sensor’s hit status of lighthouses. It is used to solve device’s pose, which is described in next sub-section.

As of IMU data, acceleration and angular velocity could be used to estimate pose, especially rotation.

Because of IMU’s high sample rate (1000Hz) and robustness, IMU data is primarily taken as rotation data source. Tracking algorithm maintains a *Rotation Model*, and integrates gyroscope’s angular velocity, resulting device’s rotation with high update rate and very low latency.

But IMUs have side effects, they have zero drift that errors increase over time. Also, integration of angular velocity don’t know the device’s initial orientation, so that the Rotation Model has a constant difference from device’s actual rotation.

Therefore, the rotation model should be corrected by light sensor’s data, which is solved in tracking module, and sent as a feedback to Rotation Model.

### Tracking Module

Tracking module is the last station of processing sensor data, and they are finally turned into tacking result, as known as device’s translation and rotation.

Tracking module mainly receives *HitMap*. Each element of a HitMap represents one light sensor is hit by two axes of a lighthouse. For a lighthouse, HitMap is the projection result of a device’s light sensors, and it could be easily converted to an image, where hit sensors are image points, and lighthouse is a pinhole camera.

Given image points and each point’s position relative to an object, we can solve this problem and get translation and rotation of the object, relative to the camera, which is called [Perspective-n-Point problem](https://en.wikipedia.org/wiki/Perspective-n-Point) (PnP problem).

The tracking module has multiple strategies to solve PnP, depending on the number of light sensors hit, balancing on accuracy and robustness.

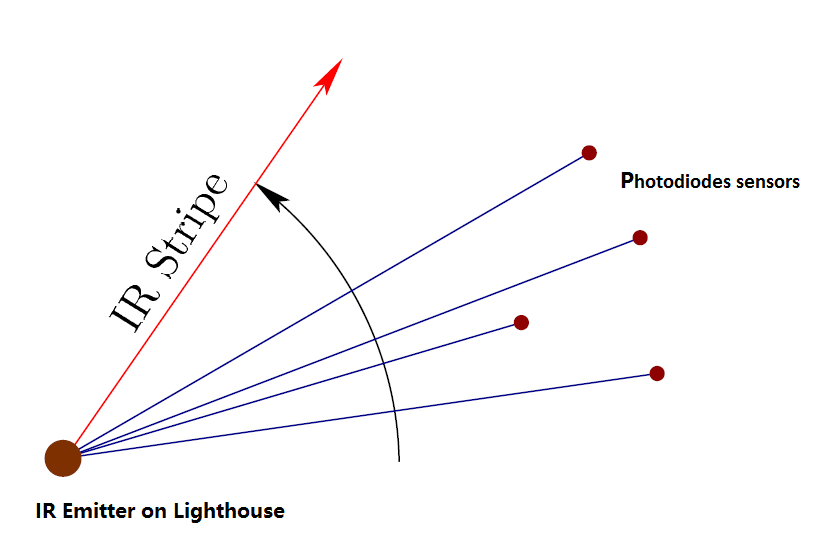
Besides PnP solving, tracking module applies [Kalman Filter](https://en.wikipedia.org/wiki/Kalman_filter) to its result, reducing device’s jitters and improve tracking smoothness, and provides prediction of translation and rotation in the future, for [timewarp](http://xinreality.com/wiki/Timewarp). Tracking module is highly extensible, and could be applied with various tracking strategy and optimization.

## Tracking Techniques

### Laser Scanning and HitMap



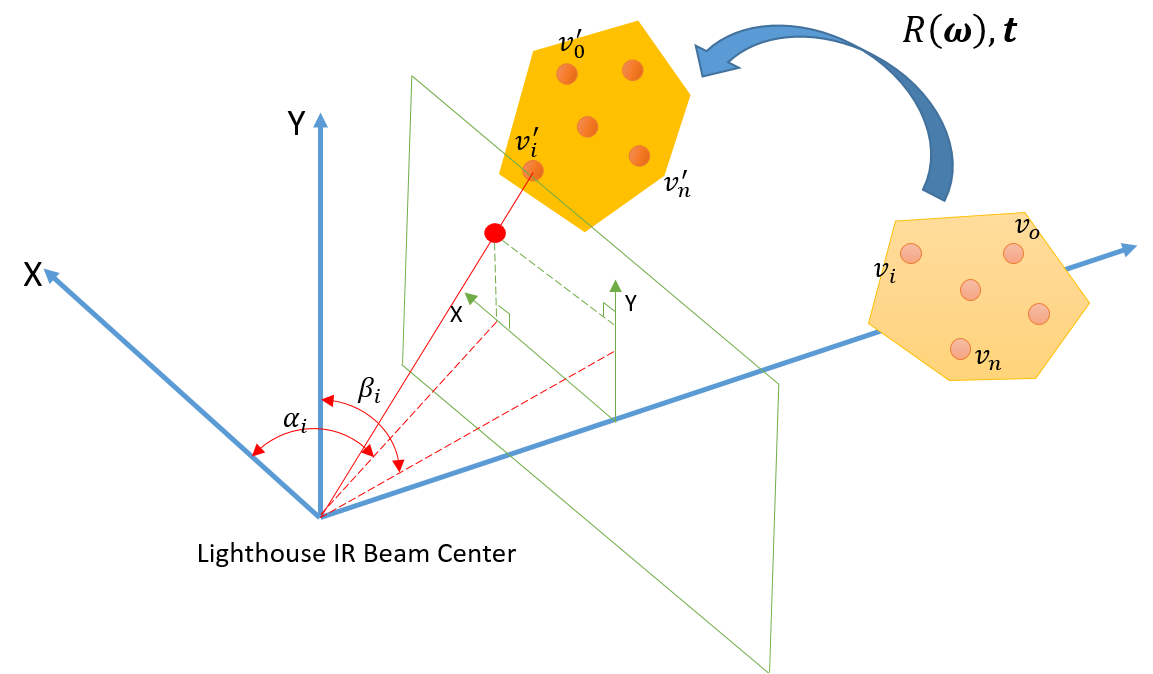
Here are figures of the hardware devices in Hypereal’s lighthouse tracking approach. The left is a base station containing motors that emit horizontal and vertical sheets of IR light. The right figure shows the photodiodes sensors on the front of the HMD to detect the incident IR light.



Above figure is a 2D view of the angular sweep of the IR stripe in our laser-based tracking approach. 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 photodiodes, the direction is known because of the spinning rate and time since the scan beginning which is wirelessly synchronized. Below we will show it could become a well formulated problem.



As shown in the above figure, assuming a tracking object undergoes a rigid transformation, where is the axis–angle representation of the rotation and is the translation component. Let denote the positions of the photodiodes sensors attached on the object. For each sensor, we can get the lighthouse signals, and it could be projected to a plane, and get an image point on this plane:

Given each sensor’s hit on image plane, namely *HitMap*, and each sensor’s position in device coordinate, we could regard it as PnP problem, and discuss this on next sub-section.

### PnP Problem

PnP problem is defined as:

“Perspective-n-Point is the problem of estimating the pose of a calibrated camera given a set of n 3D points in the world and their corresponding 2D projections in the image. The camera pose consists of 6 degrees-of-freedom (DOF) which are made up of the rotation (roll, pitch, and yaw) and 3D translation of the camera with respect to the world.” (From [wikipedia](https://en.wikipedia.org/wiki/Perspective-n-Point))

We use [EPnP](http://cvlab.epfl.ch/EPnP/index.php) to solve device’s pose on initial status, which requires 3 or more perspective points. And for consequence HitMaps, we could choose either EPnP or iterative method to solve pose. The latter is based on Levenberg-Marquardt optimization, using previous result to iterate and solve faster and more accurate.

The iterative method requires less points than EPnP, depending on points’ distribution. This helps the tracking algorithm solve pose on some very bad pose, in which few points are hit and usually occurs in VR experiencing.

Besides, the tracking algorithm uses [another PnP strategy](http://ieeexplore.ieee.org/document/6906963/), which requires device’s rotation and few points, and extracts device’s translation. Device’s rotation could be integrated by IMU’s gyroscope, which is real-time and robust.

The tracking algorithm leverages three approach of PnP solving, for various situation to provide stable pose estimation.

### Dual Lighthouse Fusion

In dual lighthouse system, each lighthouse emits one solved pose of the device, while the two lighthouse’s results are more or less different, due to measurement error and number of points hit by each lighthouse. So we should fuse multiple results into one.

The fusion strategy uses weighted average on results, according to the confidence of each lighthouse’ result. As we all know, more points hit means more accurate and stable result, leading to more confidence. So we use number of sensor points hit to represent the weight.

The output contains translation and rotation , the latter is in the form of quaternions.

So we have

Where are lighthouse 0 and lighthouse 1’s solved translation and rotation. is [spherical linear interpolation](https://en.wikipedia.org/wiki/Slerp) of quaternions. And

Where are numbers of points hit by lighthouse 0 and lighthouse 1 respectively.