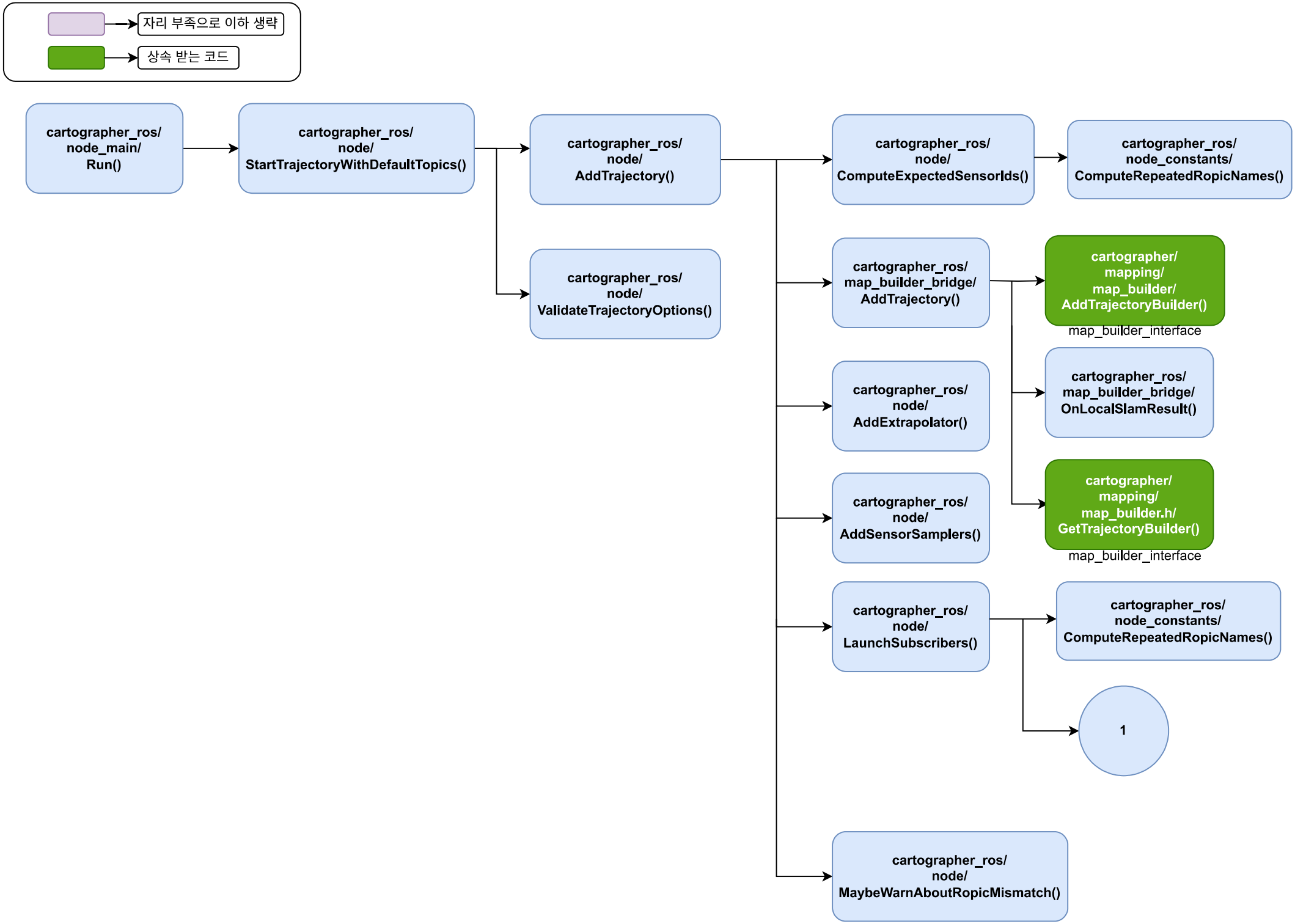
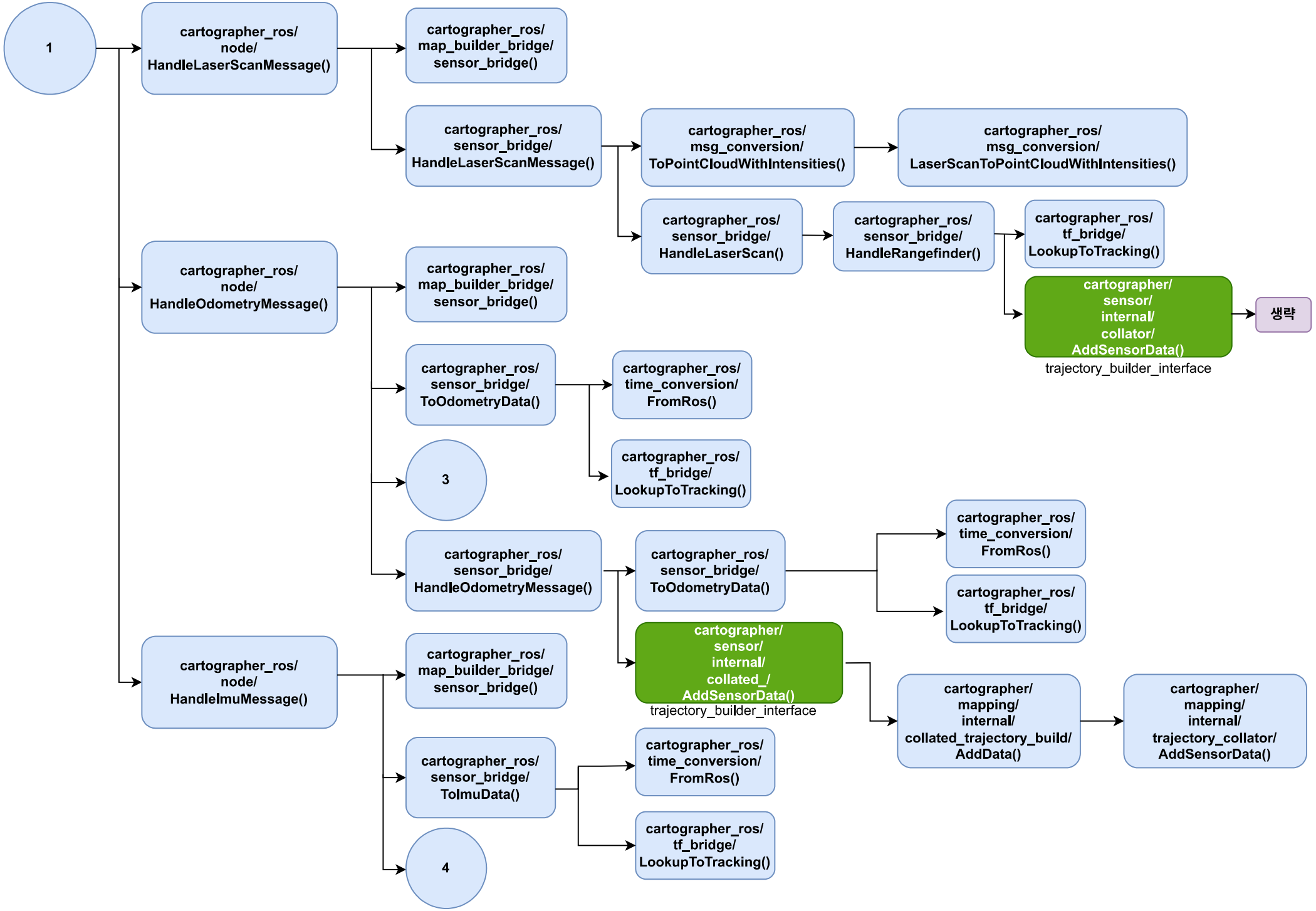
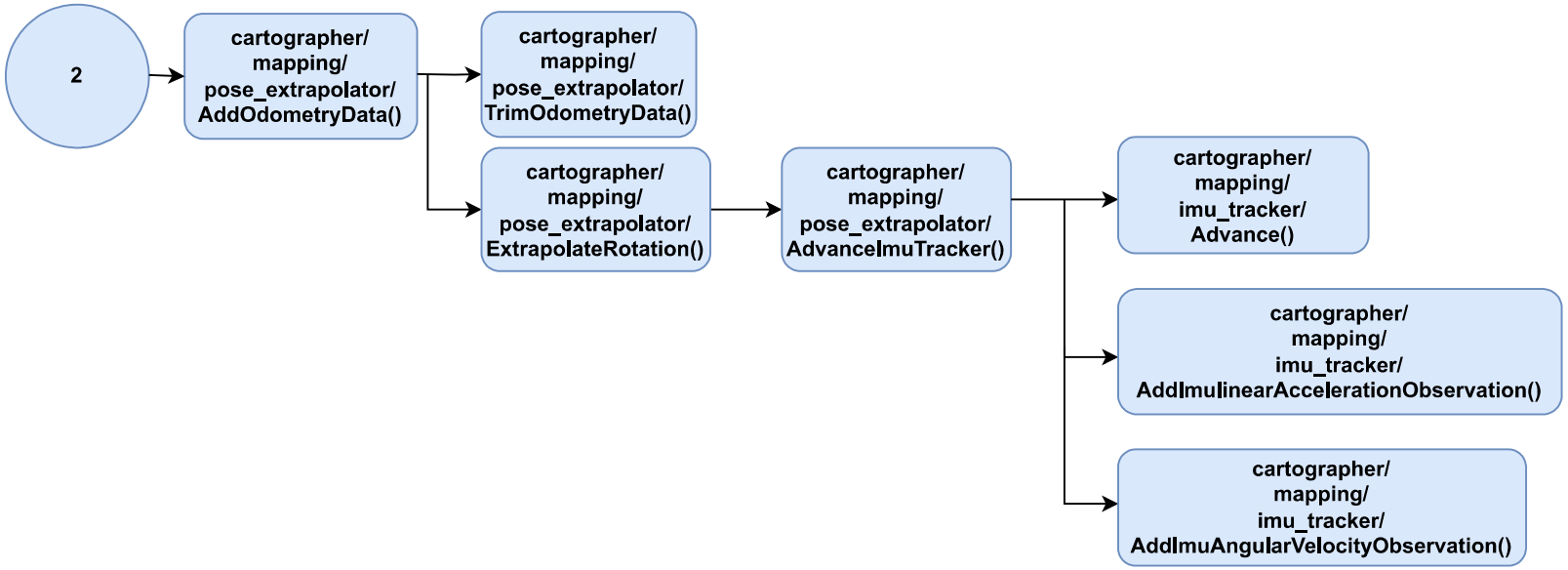
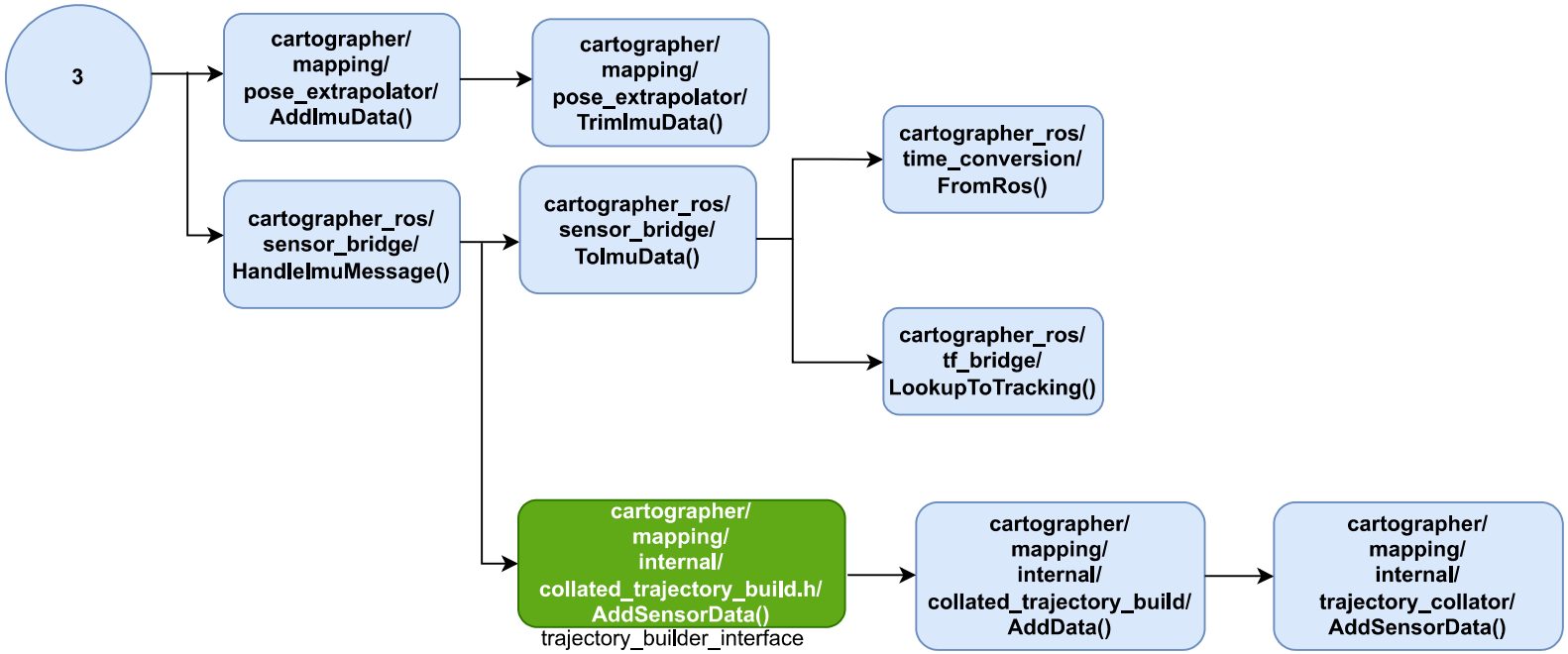
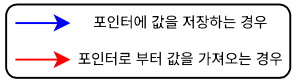

자리 부족으로 이하 생략
상속 받는 코드
cartographer_ros/
node_main/
Run()
cartographer_ros/
node/
StartTrajectoryWithDefaultTopics()
cartographer_ros/
node/
AddTrajectory()
cartographer_ros/
node/
ValidateTrajectoryOptions()
cartographer_ros/
node/
ComputeExpectedSensorIds()
cartographer_ros/
node_constants/
ComputeRepeatedRopicNames()
cartographer_ros/
map_builder_bridge/
AddTrajectory()
cartographer/
mapping/
map_builder/
AddTrajectoryBuilder()
map_builder_interface
cartographer_ros/
map_builder_bridge/
OnLocalSlamResult()
cartographer_ros/
node/
AddExtrapolator()
cartographer_ros/
node/
AddSensorSamplers()
cartographer/
mapping/
map_builder.h/
GetTrajectoryBuilder()
map_builder_interface
cartographer_ros/
node/
LaunchSubscribers()
cartographer_ros/
node_constants/
ComputeRepeatedRopicNames()
1
cartographer_ros/
node/
MaybeWarnAboutRopicMismatch()


1
cartographer_ros/ node/ HandleLaserScanMessage()
cartographer_ros/ map_builder_bridge/ sensor_bridge()
cartographer_ros/ sensor_bridge/ HandleLaserScanMessage()
cartographer_ros/ msg_conversion/ ToPointCloudWithIntensities()
cartographer_ros/ msg_conversion/ LaserScanToPointCloudWithIntensities()
cartographer_ros/ sensor_bridge/ HandleLaserScan()
cartographer_ros/ sensor_bridge/ HandleRangefinder()
cartographer_ros/ tf_bridge/ LookupToTracking()
cartographer/ sensor/ internal/ collator/ AddSensorData()
trajectory_builder_interface
생략
cartographer_ros/ node/ HandleOdometryMessage()
cartographer_ros/ map_builder_bridge/ sensor_bridge()
cartographer_ros/ sensor_bridge/ ToOdometryData()
cartographer_ros/ time_conversion/ FromRos()
cartographer_ros/ tf_bridge/ LookupToTracking()
3
cartographer_ros/ sensor_bridge/ HandleOdometryMessage()
cartographer_ros/ sensor_bridge/ ToOdometryData()
cartographer_ros/ time_conversion/ FromRos()
cartographer_ros/ tf_bridge/ LookupToTracking()
cartographer/ sensor/ internal/ collated_/ AddSensorData()
trajectory_builder_interface
cartographer/ mapping/ internal/ collated_trajectory_build/ AddData()
cartographer/ mapping/ internal/ trajectory_collator/ AddSensorData()
cartographer_ros/ node/ HandleImuMessage()
cartographer_ros/ map_builder_bridge/ sensor_bridge()
cartographer_ros/ sensor_bridge/ ToImuData()
cartographer_ros/ time_conversion/ FromRos()
cartographer_ros/ tf_bridge/ LookupToTracking()
4


2
cartographer/
mapping/
pose_extrapolator/
AddOdometryData()
cartographer/
mapping/
pose_extrapolator/
TrimOdometryData()
cartographer/
mapping/
pose_extrapolator/
ExtrapolateRotation()
cartographer/
mapping/
pose_extrapolator/
AdvanceImuTracker()
cartographer/
mapping/
imu_tracker/
Advance()
cartographer/
mapping/
imu_tracker/
AddImulinearAccelerationObservation()
cartographer/
mapping/
imu_tracker/
AddImuAngularVelocityObservation()


3
cartographer/
mapping/
pose_extrapolator/
AddImuData()
cartographer/
mapping/
pose_extrapolator/
TrimImuData()
cartographer_ros/
sensor_bridge/
HandleImuMessage()
cartographer_ros/
sensor_bridge/
ToImuData()
cartographer_ros/
time_conversion/
FromRos()
cartographer_ros/
tf_bridge/
LookupToTracking()
cartographer/
mapping/
internal/
collated_trajectory_build.h/
AddSensorData()
trajectory_builder_interface
cartographer/
mapping/
internal/
collated_trajectory_build/
AddData()
cartographer/
mapping/
internal/
trajectory_collator/
AddSensorData()



<node.cpp>

cartographer_ros/ node/ AddSensorSamplers() → sensor_samplers_

```
sensor_samplers_.emplace(
    std::piecewise_construct, std::forward_as_tuple(trajectory_id),
    std::forward_as_tuple(
        options.rangefinder_sampling_ratio, options.odometry_sampling_ratio,
        options.fixed_frame_pose_sampling_ratio, options.imu_sampling_ratio,
        options.landmarks_sampling_ratio));
```

cartographer_ros/ node/ LaunchSubscriber() → subscriber_

```
subscribers_[trajectory_id].push_back(
    {SubscribeWithHandler<sensor_msgs::msg::LaserScan>(
        &Node::HandleLaserScanMessage, trajectory_id, topic, node_, this),
    topic}); //LM scan sub 하는 코드
```

cartographer_ros/ node/ AddExtrapolator() → extrapolators_

```
extrapolators_.emplace(
    std::piecewise_construct, std::forward_as_tuple(trajectory_id),
    std::forward_as_tuple(
        ::cartographer::common::FromSeconds(kExtrapolationEstimationTimeSec),
        gravity_time_constant));
```

cartographer_ros/ node/ Node() → map_builder_bridge_

```
map_builder_bridge_.reset(new cartographer_ros::MapBuilderBridge(node_options_, std::move(map_builder), tf_buffer.get()));
```

<sensor_bridge.cpp>

cartographer_ros/ sensor_bridge/ HandleOdometryMessage() → imu_tracker_

```
trajectory_builder_->AddSensorData(
    sensor_id,
    carto::sensor::OdometryData{odometry_data->time, odometry_data->pose});
```

cartographer_ros/ sensor_bridge/ HandleImuMessage() → trajectory_builder_

```
trajectory_builder_->AddSensorData(
    sensor_id,
    carto::sensor::ImuData{imu_data->time, imu_data->linear_acceleration,
                           imu_data->angular_velocity});
```

cartographer_ros/ sensor_bridge/ ToOdometryData() → extrapolators_

```
return absl::make_unique<carto::sensor::OdometryData>(
    carto::sensor::OdometryData{
        time, ToRigid3d(msg->pose.pose) * sensor_to_tracking->inverse()});
```

cartographer_ros/ sensor_bridge/ ToImuData() → extrapolators_

```
return absl::make_unique<carto::sensor::ImuData>(carto::sensor::ImuData{
    time, sensor_to_tracking->rotation() * ToEigen(msg->linear_acceleration),
    sensor_to_tracking->rotation() * ToEigen(msg->angular_velocity)});
```

cartographer_ros/ sensor_bridge/ HandleRangefinder() → trajectory_builder_

```
trajectory_builder_->AddSensorData(
    sensor_id, carto::sensor::TimedPointCloudData{
                   time, sensor_to_tracking->translation().cast<float>(),
                   carto::sensor::TransformTimedPointCloud(
                       ranges, sensor_to_tracking->cast<float>()), intensities_});
```

<map_builder_bridge.cpp>

cartographer_ros/
map_builder_bridge/
Addtrajectory()

→ trajectory_options_

```
trajectory_options_.emplace(trajectory_id, trajectory_options);
```

cartographer_ros/
map_builder_bridge/
OnLocalSlamResult()

→ local_slam_data_

```
std::shared_ptr<const LocalTrajectoryData::LocalSlamData> local_slam_data =
    std::make_shared<LocalTrajectoryData::LocalSlamData>(
        LocalTrajectoryData::LocalSlamData{time, local_pose,
                                           std::move(range_data_in_local)});
absl::MutexLock lock(&mutex_);
local_slam_data_[trajectory_id] = std::move(local_slam_data);
```

cartographer_ros/
map_builder_bridge/
AddTrajectory()

→ sensor_bridges_

```
sensor_bridges_[trajectory_id] = absl::make_unique<SensorBridge>(
    trajectory_options.num_subdivisions_per_laser_scan,
    trajectory_options.tracking_frame,
    node_options_.lookup_transform_timeout_sec, tf_buffer_,
    map_builder_->GetTrajectoryBuilder(trajectory_id));
```

cartographer_ros/
map_builder_bridge/
sensor_bridge()

← sensor_bridges_

```
return sensor_bridges_.at(trajectory_id).get();
```

<map_builder.cc, .h>

cartographer/
mapping/
map_builder/
AddTrajectorybuilder()

→ trajectory_builders_

```
trajectory_builders_.push_back(absl::make_unique<CollatedTrajectoryBuilder>(
    trajectory_options, sensor_collator_.get(), trajectory_id,
    expected_sensor_ids,
    CreateGlobalTrajectoryBuilder2D(
        std::move(local_trajectory_builder), trajectory_id,
        static_cast<PoseGraph2D*>(pose_graph_.get()),
        local_slam_result_callback, pose_graph_odometry_motion_filter)));
```

→ local_trajectory_builder

```
local_trajectory_builder = absl::make_unique<LocalTrajectoryBuilder2D>(
    trajectory_options.trajectory_builder_2d_options(),
    SelectRangeSensorIds(expected_sensor_ids));
```

cartographer/
mapping/
map_builder.h/
GetTrajectoryBuilder()

← trajectory_builders_

```
return trajectory_builders_.at(trajectory_id).get();
```

cartographer/
mapping/
pose_extrapolator/
AddOdometryData()

→ odometry_data_

```
odometry_data_.push_back(odometry_data);
```

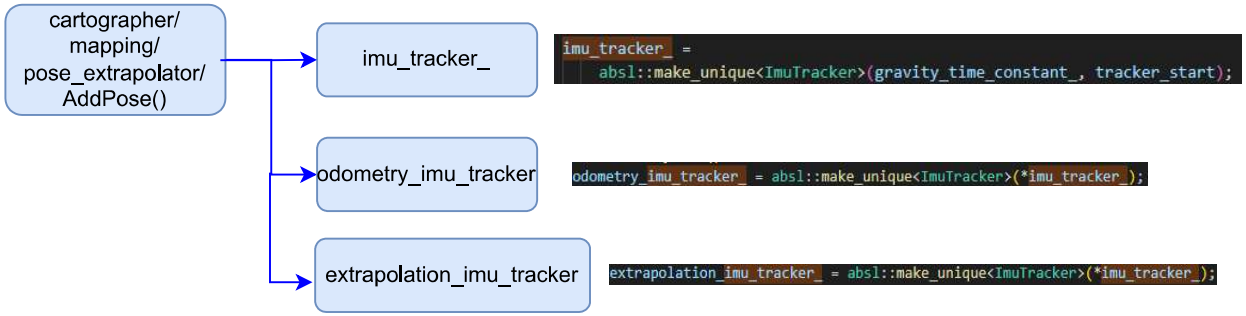
→ odometry_data_

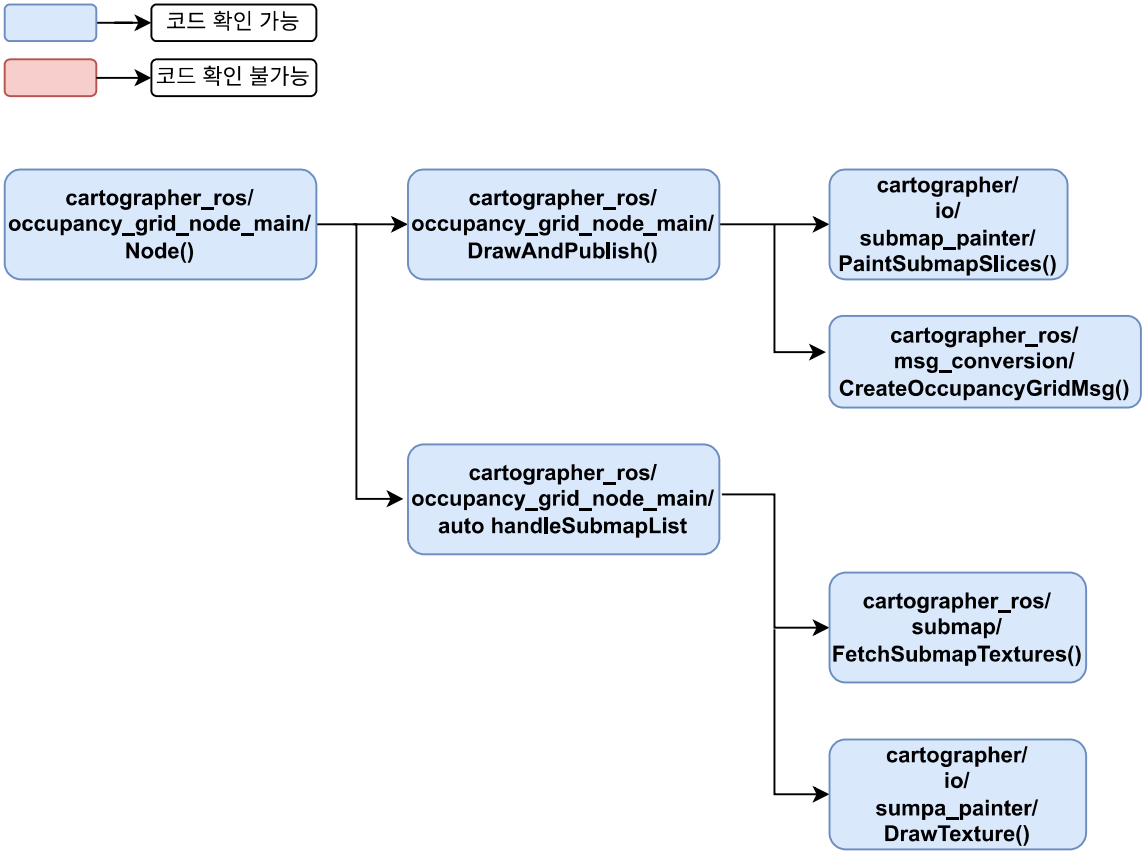
```
linear_velocity_from_odometry_ =
    orientation_at_newest_odometry_time *
    linear_velocity_in_tracking_frame_at_newest_odometry_time;
```

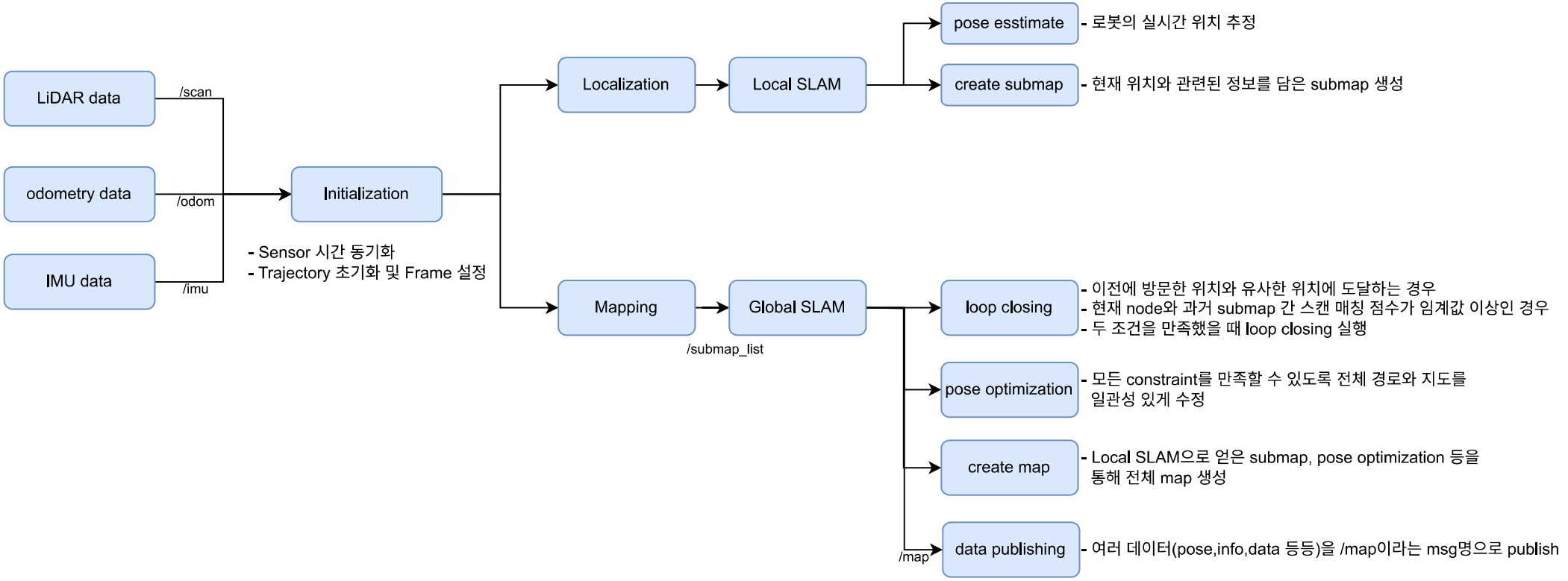
→ odometry_data_

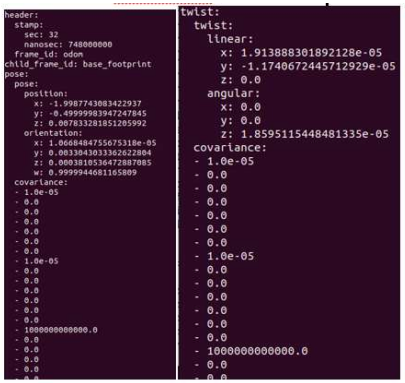
```
angular_velocity_from_odometry_ =
    transform::RotationQuaternionToAngleAxisVector(
        odometry_pose_delta.rotation()) /
    odometry_time_delta;
```

<pose_extrapolator.cc>




코드 확인 가능
코드 확인 불가능
cartographer_ros/
occupancy_grid_node_main/
Node()
cartographer_ros/
occupancy_grid_node_main/
DrawAndPublish()
cartographer/
io/
submap_painter/
PaintSubmapSlices()
cartographer_ros/
msg_conversion/
CreateOccupancyGridMsg()
cartographer_ros/
occupancy_grid_node_main/
auto handleSubmapList
cartographer_ros/
submap/
FetchSubmapTextures()
cartographer/
io/
sumpa_painter/
DrawTexture()



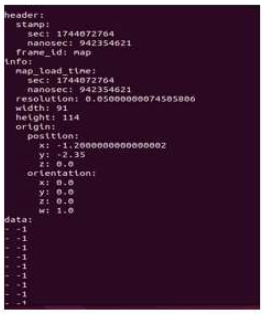


</odom>

</scan>

</submap_list>

- 특정 주기 별로 submap version의 값이 증가함.
  - 로봇의 속도가 올라가면 주기가 빨라지는 것 같음.
  - 회전에서 특히 빨리 올라감.
  - submap version은 180이 최대로 보임.
- Submap version = 90 이 되면 submap_index가 증가함.



</map>

- 로봇의 현재 위치
- Map 파일의 넓이와 높이
- 맵의 인식 값을 확인할 수 있는 것으로 보임
  - -1: 알려지지 않은 공간**(회색)**
  - 0: 자유공간 **(흰색)**
  - 100: 완전하게 점유된 공간 **(검정색)**