pose_tracker.cc

Go to the documentation of this file.

```
/*
 * Copyright 2016 The Cartographer Authors
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#include "cartographer/kalman_filter/pose_tracker.h"

#include <cmath>
#include <limits>
#include <utility>

#include "Eigen/Geometry"
#include "cartographer/common/lua_parameter_dictionary.h"
#include "cartographer/common/math.h"
#include "cartographer/common/time.h"
#include "cartographer/kalman_filter/gaussian_distribution.h"
#include "cartographer/kalman_filter/unscented_kalman_filter.h"
#include "cartographer/transform/transform.h"
#include "glog/logging.h"

namespace cartographer {
namespace kalman_filter {

namespace {

PoseTracker::State AddDelta(const PoseTracker::State& state,
                            const PoseTracker::State& delta) {
  PoseTracker::State new_state = state + delta;
  const Eigen::Quaterniond orientation =
      transform::AngleAxisVectorToRotationQuaternion(
          Eigen::Vector3d(state[PoseTracker::kMapOrientationX],
                          state[PoseTracker::kMapOrientationY],
                          state[PoseTracker::kMapOrientationZ]));
  const Eigen::Vector3d rotation_vector(delta[PoseTracker::kMapOrientationX],
                                        delta[PoseTracker::kMapOrientationY],
                                        delta[PoseTracker::kMapOrientationZ]);
  CHECK_LT(rotation_vector.norm(), M_PI / 2.)
      << "Sigma point is far from the mean, recovered delta may be incorrect.";
  const Eigen::Quaterniond rotation =
      transform::AngleAxisVectorToRotationQuaternion(rotation_vector);
  const Eigen::Vector3d new_orientation =
      transform::RotationQuaternionToAngleAxisVector(orientation * rotation);
  new_state[PoseTracker::kMapOrientationX] = new_orientation.x();
  new_state[PoseTracker::kMapOrientationY] = new_orientation.y();
  new_state[PoseTracker::kMapOrientationZ] = new_orientation.z();
  return new_state;
}

PoseTracker::State ComputeDelta(const PoseTracker::State& origin,
                                const PoseTracker::State& target) {
  PoseTracker::State delta = target - origin;
```

```
  63   const Eigen::Quaterniond origin_orientation =
  64       transform::AngleAxisVectorToRotationQuaternion(
  65           Eigen::Vector3d(origin[PoseTracker::kMapOrientationX],
  66                           origin[PoseTracker::kMapOrientationY],
  67                           origin[PoseTracker::kMapOrientationZ]));
  68   const Eigen::Quaterniond target_orientation =
  69       transform::AngleAxisVectorToRotationQuaternion(
  70           Eigen::Vector3d(target[PoseTracker::kMapOrientationX],
  71                           target[PoseTracker::kMapOrientationY],
  72                           target[PoseTracker::kMapOrientationZ]));
  73   const Eigen::Vector3d rotation =
  74       transform::RotationQuaternionToAngleAxisVector(
  75           origin_orientation.inverse() * target_orientation);
  76   delta[PoseTracker::kMapOrientationX] = rotation.x();
  77   delta[PoseTracker::kMapOrientationY] = rotation.y();
  78   delta[PoseTracker::kMapOrientationZ] = rotation.z();
  79   return delta;
  80 }
  81
  82 // Build a model matrix for the given time delta.
  83 PoseTracker::State ModelFunction(const PoseTracker::State& state,
  84                                  const double delta_t) {
  85   CHECK_GT(delta_t, 0.);
  86
  87   PoseTracker::State new_state;
  88   new_state[PoseTracker::kMapPositionX] =
  89       state[PoseTracker::kMapPositionX] +
  90       delta_t * state[PoseTracker::kMapVelocityX];
  91   new_state[PoseTracker::kMapPositionY] =
  92       state[PoseTracker::kMapPositionY] +
  93       delta_t * state[PoseTracker::kMapVelocityY];
  94   new_state[PoseTracker::kMapPositionZ] =
  95       state[PoseTracker::kMapPositionZ] +
  96       delta_t * state[PoseTracker::kMapVelocityZ];
  97
  98   new_state[PoseTracker::kMapOrientationX] =
  99       state[PoseTracker::kMapOrientationX];
 100   new_state[PoseTracker::kMapOrientationY] =
 101       state[PoseTracker::kMapOrientationY];
 102   new_state[PoseTracker::kMapOrientationZ] =
 103       state[PoseTracker::kMapOrientationZ];
 104
 105   new_state[PoseTracker::kMapVelocityX] = state[PoseTracker::kMapVelocityX];
 106   new_state[PoseTracker::kMapVelocityY] = state[PoseTracker::kMapVelocityY];
 107   new_state[PoseTracker::kMapVelocityZ] = state[PoseTracker::kMapVelocityZ];
 108
 109   return new_state;
 110 }
 111
 112 }  // namespace
 113
 114 PoseAndCovariance operator*(const transform::Rigid3d& transform,
 115                             const PoseAndCovariance& pose_and_covariance) {
 116   GaussianDistribution<double, 6> distribution(
 117       Eigen::Matrix<double, 6, 1>::Zero(), pose_and_covariance.covariance);
 118   Eigen::Matrix<double, 6, 6> linear_transform;
 119   linear_transform << transform.rotation().matrix(), Eigen::Matrix3d::Zero(),
 120       Eigen::Matrix3d::Zero(), transform.rotation().matrix();
 121   return {transform * pose_and_covariance.pose,
 122           (linear_transform * distribution).GetCovariance()};
 123 }
 124
 125 proto::PoseTrackerOptions CreatePoseTrackerOptions(
 126     common::LuaParameterDictionary* const parameter_dictionary) {
 127   proto::PoseTrackerOptions options;
 128   options.set_position_model_variance(
 129       parameter_dictionary->GetDouble("position_model_variance"));
 130   options.set_orientation_model_variance(
 131       parameter_dictionary->GetDouble("orientation_model_variance"));
 132   options.set_velocity_model_variance(
 133       parameter_dictionary->GetDouble("velocity_model_variance"));
 134   options.set_imu_gravity_time_constant(
 135       parameter_dictionary->GetDouble("imu_gravity_time_constant"));
 136   options.set_imu_gravity_variance(
 137       parameter_dictionary->GetDouble("imu_gravity_variance"));
```

```
138   options.set_num_odometry_states(
139       parameter_dictionary->GetNonNegativeInt("num_odometry_states"));
140   CHECK_GT(options.num_odometry_states(), 0);
141   return options;
142 }
143
144 PoseTracker::Distribution PoseTracker::KalmanFilterInit() {
145   State initial_state = State::Zero();
146   // We are certain about the complete state at the beginning. We define the
147   // initial pose to be at the origin and axis aligned. Additionally, we claim
148   // that we are not moving.
149   StateCovariance initial_covariance = 1e-9 * StateCovariance::Identity();
150   return Distribution(initial_state, initial_covariance);
151 }
152
153 PoseTracker::PoseTracker(const proto::PoseTrackerOptions& options,
154                          const common::Time time)
155     : options_(options),
156       time_(time),
157       kalman_filter_(KalmanFilterInit(), AddDelta, ComputeDelta),
158       imu_tracker_(options.imu_gravity_time_constant(), time),
159       odometry_state_tracker_(options.num_odometry_states()) {}
160
161 PoseTracker::~PoseTracker() {}
162
163 PoseTracker::Distribution PoseTracker::GetBelief(const common::Time time) {
164   Predict(time);
165   return kalman_filter_.GetBelief();
166 }
167
168 void PoseTracker::GetPoseEstimateMeanAndCovariance(const common::Time time,
169                                                    transform::Rigid3d* pose,
170                                                    PoseCovariance* covariance) {
171   const Distribution belief = GetBelief(time);
172   *pose = RigidFromState(belief.GetMean());
173   static_assert(kMapPositionX == 0, "Cannot extract PoseCovariance.");
174   static_assert(kMapPositionY == 1, "Cannot extract PoseCovariance.");
175   static_assert(kMapPositionZ == 2, "Cannot extract PoseCovariance.");
176   static_assert(kMapOrientationX == 3, "Cannot extract PoseCovariance.");
177   static_assert(kMapOrientationY == 4, "Cannot extract PoseCovariance.");
178   static_assert(kMapOrientationZ == 5, "Cannot extract PoseCovariance.");
179   *covariance = belief.GetCovariance().block<6, 6>(0, 0);
180   covariance->block<2, 2>(3, 3) +=
181       options_.imu_gravity_variance() * Eigen::Matrix2d::Identity();
182 }
183
184 const PoseTracker::Distribution PoseTracker::BuildModelNoise(
185     const double delta_t) const {
186   // Position is constant, but orientation changes.
187   StateCovariance model_noise = StateCovariance::Zero();
188
189   model_noise.diagonal() <<
190       // Position in map.
191       options_.position_model_variance() * delta_t,
192       options_.position_model_variance() * delta_t,
193       options_.position_model_variance() * delta_t,
194
195       // Orientation in map.
196       options_.orientation_model_variance() * delta_t,
197       options_.orientation_model_variance() * delta_t,
198       options_.orientation_model_variance() * delta_t,
199
200       // Linear velocities in map.
201       options_.velocity_model_variance() * delta_t,
202       options_.velocity_model_variance() * delta_t,
203       options_.velocity_model_variance() * delta_t;
204
205   return Distribution(State::Zero(), model_noise);
206 }
207
208 void PoseTracker::Predict(const common::Time time) {
209   imu_tracker_.Advance(time);
210   CHECK_LE(time_, time);
211   const double delta_t = common::ToSeconds(time - time_);
212   if (delta_t == 0.) {
```

```
213       return;
214     }
215     kalman_filter_.Predict(
216         [this, delta_t](const State& state) -> State {
217           return ModelFunction(state, delta_t);
218         },
219         BuildModelNoise(delta_t));
220     time_ = time;
221   }
222
223   void PoseTracker::AddImuLinearAccelerationObservation(
224       const common::Time time, const Eigen::Vector3d& imu_linear_acceleration) {
225     imu_tracker_.Advance(time);
226     imu_tracker_.AddImuLinearAccelerationObservation(imu_linear_acceleration);
227     Predict(time);
228   }
229
230   void PoseTracker::AddImuAngularVelocityObservation(
231       const common::Time time, const Eigen::Vector3d& imu_angular_velocity) {
232     imu_tracker_.Advance(time);
233     imu_tracker_.AddImuAngularVelocityObservation(imu_angular_velocity);
234     Predict(time);
235   }
236
237   void PoseTracker::AddPoseObservation(const common::Time time,
238                                        const transform::Rigid3d& pose,
239                                        const PoseCovariance& covariance) {
240     Predict(time);
241
242     // Noise covariance is taken directly from the input values.
243     const GaussianDistribution<double, 6> delta(
244         Eigen::Matrix<double, 6, 1>::Zero(), covariance);
245
246     kalman_filter_.Observe<6>(
247         [this, &pose](const State& state) -> Eigen::Matrix<double, 6, 1> {
248           const transform::Rigid3d state_pose = RigidFromState(state);
249           const Eigen::Vector3d delta_orientation =
250               transform::RotationQuaternionToAngleAxisVector(
251                   pose.rotation().inverse() * state_pose.rotation());
252           const Eigen::Vector3d delta_translation =
253               state_pose.translation() - pose.translation();
254           Eigen::Matrix<double, 6, 1> return_value;
255           return_value << delta_translation, delta_orientation;
256           return return_value;
257         },
258         delta);
259   }
260
261   // Updates from the odometer are in the odometer's map-like frame, called the
262   // 'odometry' frame. The odometer_pose converts data from the map frame
263   // into the odometry frame.
264   void PoseTracker::AddOdometerPoseObservation(
265       const common::Time time, const transform::Rigid3d& odometer_pose,
266       const PoseCovariance& covariance) {
267     if (!odometry_state_tracker_.empty()) {
268       const auto& previous_odometry_state = odometry_state_tracker_.newest();
269       const transform::Rigid3d delta =
270           previous_odometry_state.odometer_pose.inverse() * odometer_pose;
271       const transform::Rigid3d new_pose =
272           previous_odometry_state.state_pose * delta;
273       AddPoseObservation(time, new_pose, covariance);
274     }
275
276     const Distribution belief = GetBelief(time);
277
278     odometry_state_tracker_.AddOdometryState(
279         {time, odometer_pose, RigidFromState(belief.GetMean())});
280   }
281
282   const mapping::OdometryStateTracker::OdometryStates&
283   PoseTracker::odometry_states() const {
284     return odometry_state_tracker_.odometry_states();
285   }
286
287   transform::Rigid3d PoseTracker::RigidFromState(
```

```
288       const PoseTracker::State& state) {
289     return transform::Rigid3d(
290         Eigen::Vector3d(state[PoseTracker::kMapPositionX],
291                         state[PoseTracker::kMapPositionY],
292                         state[PoseTracker::kMapPositionZ]),
293         transform::AngleAxisVectorToRotationQuaternion(
294             Eigen::Vector3d(state[PoseTracker::kMapOrientationX],
295                             state[PoseTracker::kMapOrientationY],
296                             state[PoseTracker::kMapOrientationZ])) *
297             imu_tracker_.orientation());
298 }
299
300 PoseCovariance BuildPoseCovariance(const double translational_variance,
301                                    const double rotational_variance) {
302   const Eigen::Matrix3d translational =
303       Eigen::Matrix3d::Identity() * translational_variance;
304   const Eigen::Matrix3d rotational =
305       Eigen::Matrix3d::Identity() * rotational_variance;
306   // clang-format off
307   PoseCovariance covariance;
308   covariance <<
309       translational, Eigen::Matrix3d::Zero(),
310       Eigen::Matrix3d::Zero(), rotational;
311   // clang-format on
312   return covariance;
313 }
314
315 }  // namespace kalman_filter
316 }  // namespace cartographer
```

---

cartographer

Author(s):

autogenerated on Wed Jun 5 2019 21:57:58