POLITECNICO
MILANO 1863

EKF-SLAM for drone localization

Thesis project - Leonardo Drone Contest

Supervisor:

Prof. Matteo Matteucci

2020

Diego Emanuel Avila

Problem

In the context of Leonardo Drone Competition, a fully autonomous drone should, in the first round, take off from the base position and inspect the environment in order to generate a map; during the second round, the drone should follow and land on a (previously set) sequence of points.

The environment is composed by a set of obstacles of different sizes, 6 landmarks and 10 path-markers, plus the base marker.

The drone should be able to fly and simultaneously map and localize itself in the environment, and be able to identify, store and localize the path-markers, so to follow the predefined path.

Proposed solution

Since the drone has not been built yet, I am assuming the information provided by the flight controller will be:

- Drone angular and linear velocities in the global reference frame
- Poles range and bearing with respect to the local reference frame.
- Marker pose with respect to the local reference frame (x, y, z, ϕ, θ, ψ)

Given this information, the proposed solution involves the development of an EKF-SLAM algorithm, *able to dynamically add markers in the state vector*.

Extended Kalman Filter

The state vector $\boldsymbol{\mu}$ in this particular case will be $\begin{bmatrix} \boldsymbol{r} & \psi & ... & \boldsymbol{m}_1 & \boldsymbol{m}_2 & ... \end{bmatrix}$, where:

- $\boldsymbol{r}$ is the position of the robot in the world reference frame $\begin{bmatrix} x & y & z \end{bmatrix}$
- ψ is the drone yaw in the world reference frame
- $\boldsymbol{m_i}$ is the pose of the marker i $\begin{bmatrix} x & y & z & \phi & \theta & \psi \end{bmatrix}$

As known, the Kalman Filter can be divided in 2 steps: **prediction** and **correction**:

1. prediction
   - 1.1 $\hat{\mu} = g(u_t, \mu_t)$
   - 1.2 $\hat{\Sigma} = G_t \Sigma_{t-1} G_t^T + R_t$
2. correction
   - 2.1 $K_t = \hat{\Sigma} H_t^T \left(H_t \hat{\Sigma} H_t^T + Q_t \right)^{-1}$
   - 2.2 $\mu_{t+1} = \hat{\mu} + K_t (z_t - h(\hat{\mu}))$
   - 2.3 $\Sigma_{t+1} = (I - K_t H_t) \hat{\Sigma}$

- g(·) is the vehicle kinematics or motion model, which involves the motion noise, the control variables $\boldsymbol{u}$ at time t and the state vector $\boldsymbol{\mu}$ in the previous time step.
- G_t is the Jacobian matrix of the motion model g, Σ the covariance matrix in the previous time step and R, which provides a mapping between the motion noise in control space to the motion noise in state space

$$\hat{\mu} = g\left(u_t, \mu_t\right)$$

$$\hat{\Sigma} = G_t \Sigma_{t-1} G_t^T + R_t$$

$$K_t = \hat{\Sigma} H_t^T \left(H_t \hat{\Sigma} H_t^T + Q_t\right)^{-1}$$

$$\mu_{t+1} = \hat{\mu} + K_t \left(z_t - h\left(\hat{\mu}\right)\right)$$

$$\Sigma_{t+1} = \left(I - K_t H_t\right) \hat{\Sigma}$$

- K represents the Kalman Gain, and its computation involves the estimation of the covariance ($\hat{\Sigma}$), the Jacobian of the obse rvation model (H) and the observation noise given by the sensors.
- Finally, we use the predicted mean/state vector, the Kalman gain, the observation vector z and the observation model (h) to compute the current state vector; and we use the Kalman gain, the Jacobian of the observation model and the predicted covariance matrix, to compute the current covariance matrix.

$$\hat{\mu} = g\left(u_t, \mu_t\right)$$

$$\hat{\Sigma} = G_t \Sigma_{t-1} G_t^T + R_t$$

$$K_t = \hat{\Sigma} H_t^T \left(H_t \hat{\Sigma} H_t^T + Q_t\right)^{-1}$$

$$\mu_{t+1} = \hat{\mu} + K_t \left(z_t - h\left(\hat{\mu}\right)\right)$$

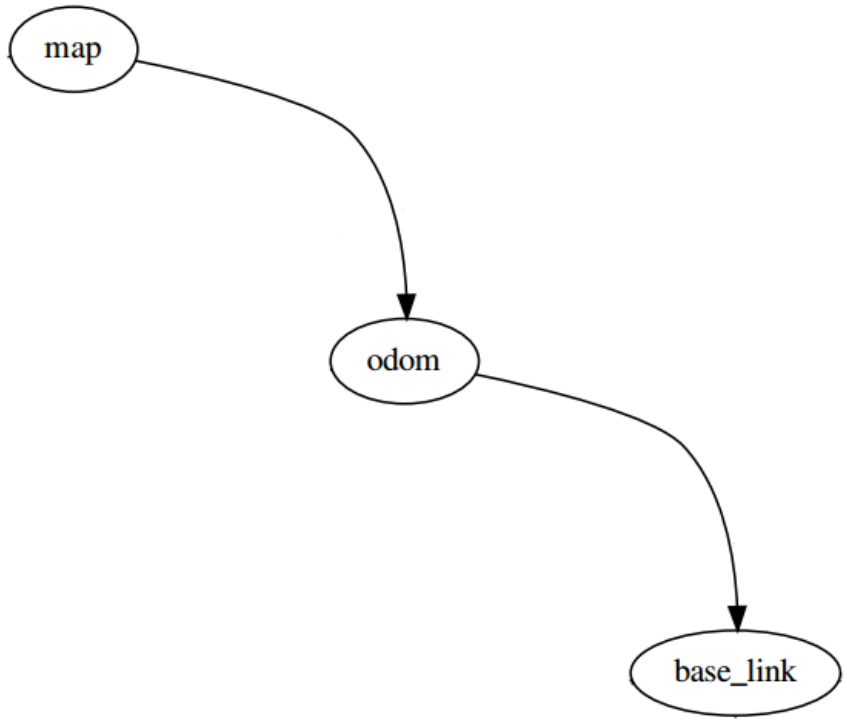
$$\Sigma_{t+1} = \left(I - K_t H_t\right) \hat{\Sigma}$$

The algorithm can be summarized in the following steps:

1. Predict the new state vector $\hat{\mu}$
2. Predict the new covariance matrix $\hat{\Sigma}$
3. For each of the seen landmarks i
   - 3.1 Compute the observation model h for the landmark i, and its Jacobian
   - 3.2 For each sensor j of the observation i:
     - 3.2.1 Compute the innovation and the gain K_i^j
     - 3.2.2 Update the state vector μ_{t+1}
     - 3.2.3 Update the covariance matrix Σ_{t+1}
4. If necessary, introduce new landmarks to the state vector.

Reference frames

All the reference frames are expressed with the East-North-Up (ENU) convention, in which the *map* reference frame, is the fixed frame; and *base_link* reference frame, refers to the drone frame.



Motion Model $g(u_t, \mu_t)$

The state vector μ (without markers [1]) is:

$$\mu = \begin{bmatrix} x^w & y^w & z^w & \psi \end{bmatrix}^T$$

- x^w y^w z^w represent the position of the drone in the map reference frame
- ψ represents the drone's orientation in the map reference frame

[1] Complete μ should be: $\begin{bmatrix} x & y & z & \psi & x_1 & y_1 & z_1 & \psi_1 & \dots & x_j & y_j & z_j & \psi_j \end{bmatrix}$

The control vector u is:

$$u = \begin{bmatrix} v_x^b & v_y^b & v_z^b & \omega_x^b & \omega_y^b & \omega_z^b & \phi^b & \theta^b & \psi^b \end{bmatrix}^T$$

- v_x^b v_y^b v_z^b represent the linear velocity in the odometry (*odom*) reference frame
- ω_x^b ω_y^b ω_z^b represents the angular velocity in the odometry reference frame
- ϕ^b θ^b ψ^b represents pose of the drone as roll, pitch and yaw in the odometry reference frame

Since the control vector elements are in the *odom* reference frame, we need to transform this into the *map* reference so to store the drone's pose with respect to the world.

The motion model is then:

Position and orientation update

$$\begin{bmatrix} \hat{\mu}_{r,x} \\ \hat{\mu}_{r,y} \\ \hat{\mu}_{r,z} \end{bmatrix} = \begin{bmatrix} \mu_{r,x} \\ \mu_{r,y} \\ \mu_{r,z} \end{bmatrix}_t + \Delta t * \mathbf{T} * \begin{bmatrix} u_{v_x} \\ u_{v_y} \\ u_{v_z} \end{bmatrix}$$

$$\hat{\mu}_{r,\psi} = \mu_{r,\psi} + \Delta t * u_{\omega_z}$$

Homogeneous transformation matrix ($\mathbf{T}$)

$$\begin{bmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi & 0 \\ \sin\psi\cos\theta & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\cos\phi & 0 \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

being ϕ, θ, ψ, roll, pitch, yaw respectively.

Observation Model $h(\hat{\mu})$

Pole j

$$\begin{bmatrix} p^b_{j,x} \\ p^b_{j,y} \\ p^b_{j,z} \end{bmatrix} = \mathbf{T}^{-1} \begin{bmatrix} p^w_{j,x} \\ p^w_{j,y} \\ p^w_{j,z} \end{bmatrix}$$

$$p^b_{j,\rho} = \sqrt{p^2_{j,x^b} + p^2_{j,y^b}}$$

$$p^b_{j,\alpha} = \text{atan2}\left(p^b_{j,y}, p^b_{j,x}\right)$$

$$p^b_{j,\beta} = \text{atan2}\left(p^b_{j,z}, p^b_{j,\rho}\right)$$

being T the transformation matrix, where $T_{0;3} = \mu_x$, $T_{1;3} = \mu_y$ and $T_{2;3} = \mu_z$ and $p^b_{j,z}$ represents the z value of the j^{th}-pole's tip

Marker j

The drone has a camera looking downwards, and every time it sees a marker, a message is broadcasted with the marker's pose in the *camera* frame. In order to estimate the marker's position and orientation, we move the pose in the *camera* reference frame to the *world* reference frame.

$$T_{m_j}^{c} = (T_r^w * T_c^r)^{-1} * T_{m_j}^{w}$$

Where $T_{m_j}^{w}$ is the marker's homogeneous transformation w.r.t the world reference frame, T_r^w is the homogeneous transformation of the drone w.r.t the world reference frame, T_c^r is the homogeneous transformation of the camera w.r.t the drone reference frame, and finally, $T_{m_j}^{c}$ is the marker's homogeneous transformation of the marker in the camera reference frame.

Marker j

From this homogeneous transformation matrix, we can extract the marker's position in the camera reference frame:

$$\begin{bmatrix} x^c_{m_j} \\ y^c_{m_j} \\ z^c_{m_j} \end{bmatrix} = \begin{bmatrix} T^c_{m_j}(0;3) \\ T^c_{m_j}(1;3) \\ T^c_{m_j}(2;3) \end{bmatrix}$$

We can also extract the marker's rotation in the camera reference frame:

$$\phi^c_{m_j} = \text{atan2}\left(T^c_{m_j}(2;1), T^c_{m_j}(2;2)\right)$$

$$\theta^c_{m_j} = \text{atan2}\left(-T^c_{m_j}(2;0), \sqrt{T^c_{m_j}(2;1)^2 + T^c_{m_j}(2;2)^2}\right)$$

$$\psi^c_{m_j} = \text{atan2}\left(T^c_{m_j}(1;0), T^c_{m_j}(0;0)\right)$$

Inverse Observation Model $h^{-1}(\hat{\mu})$

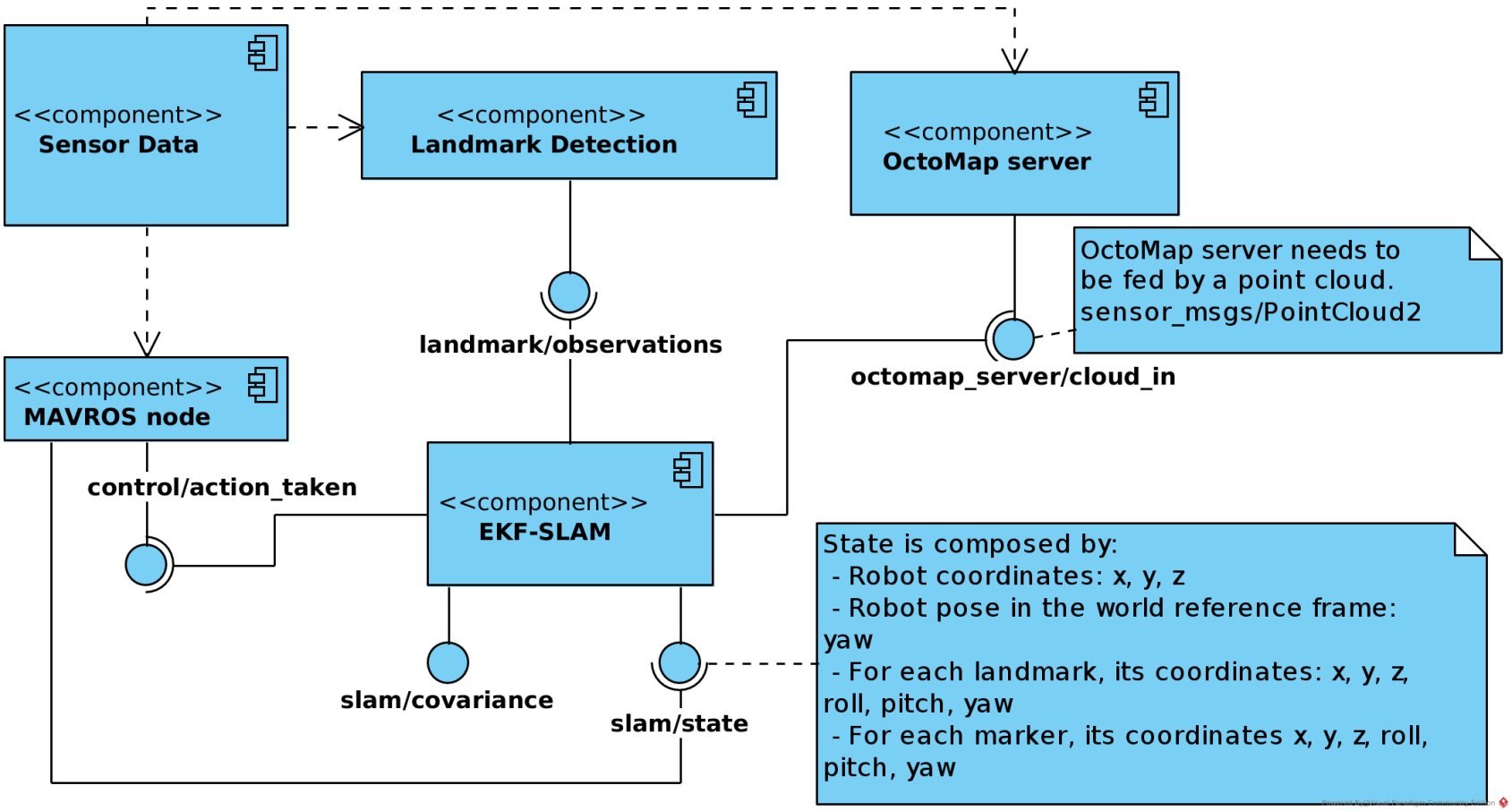
Marker j

The inverse observation model is used to add newly observed markers to the state vector. We obtain the marker pose w.r.t the camera reference frame, and we transform this to the map reference frame in the following way:

$$T_{m_j}^{w} = T_{r}^{w} * T_{c}^{r} * T_{m_j}^{c}$$

From $T_{m_j}^{w}$ we can extract the position w.r.t the world reference frame and the orientation w.r.t the world reference frame in the same way as in the observation model for markers.

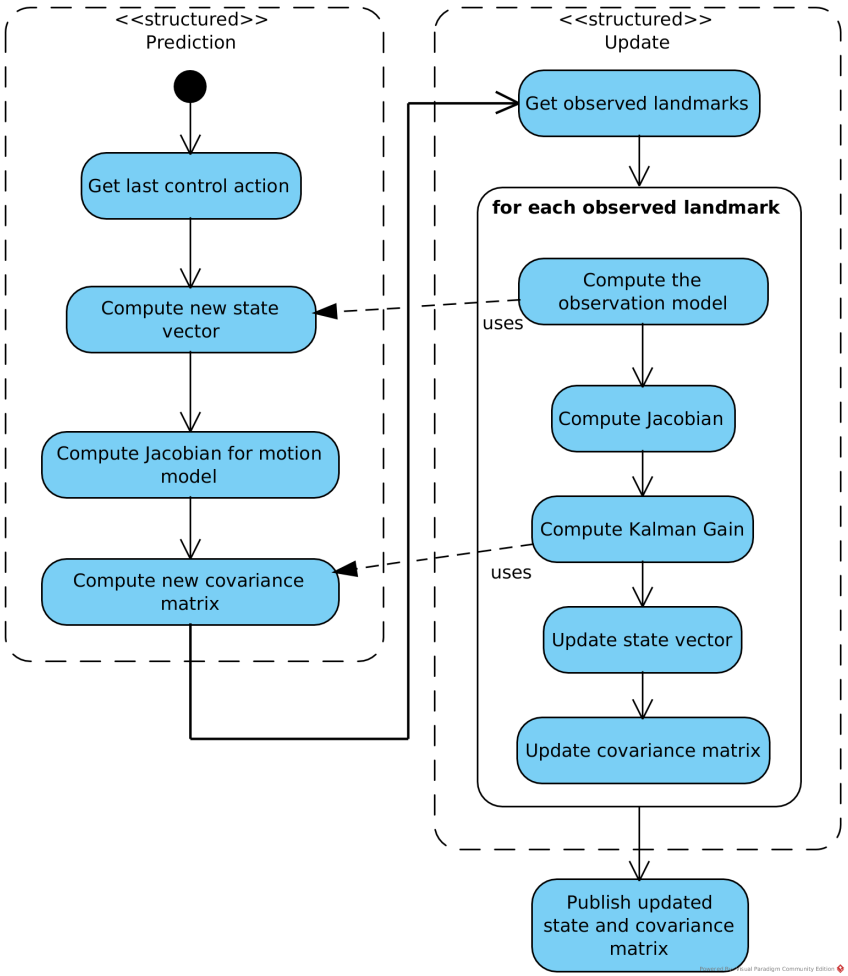
Architecture overview



The system's architecture has been done using ROS framework

- There is a ROS node which is responsible of detecting the landmarks and publishing its coordinates relative to the drone.
- Also, there is a control node, which is responsible of taking the control actions needed to drive the drone to the desired point.
  - This node publishes the actions taken as pose increments
- The EKF-SLAM node is responsible of update the state of the robot and keep track of the observed landmarks.
  - It publishes the covariance matrix and the state vector at each time step
- Finally, there is the OctoMap server, which should be fed with a point cloud

Activity flow



In the following activity diagram it can be seen the computation of the state vector and the covariance matrix.

Also, at the end of the update stage, it publishes the final state vector.

Conclusion

Considering that several things need to be defined, it seems that an EKF-SLAM should not be complicated to implement.

As stated by Bailey and Durrant-Whyte [2006]:

> *Implementing SLAM in 3-D is, in principle, a straightforward extension of the 2-D case. However, it involves significant added complexity due to the more general vehicle motion model and, most importantly, greatly increased sensing and feature modeling complexity*

Finally, I have found several EKF-SLAM implementations, being the MRPT (Blanco [2008]) implementation the most general one. Moreover, it has a wrapper for ROS called mrpt_navigation (Blanco and Bader), but it is not clear whether it accepts 3D maps

References I

Pose ekf-slam ros package. URL http://wiki.ros.org/pose_ekf_slam.

Modeling vehicle dynamics - euler angles. URL https://charlestytler.com/modeling-vehicle-dynamics-euler-angles/.

A. Alaimo, V. Artale, C. Milazzo, A. Ricciardello, and L. Trefiletti. Mathematical modeling and control of a hexacopter. In *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1043–1050, 2013.

V. Artale, Angela Ricciardello, and C. Milazzo. A quaternion-based simulation of multirotor dynamics. *International Journal of Modeling, Simulation, and Scientific Computing*, 06, 01 2015. doi: 10.1142/S1793962315500099.

References II

T. Bailey and H. Durrant-Whyte. Simultaneous localization and mapping (slam): part ii. *IEEE Robotics Automation Magazine*, 13(3):108–117, 2006.

J.L. Blanco. Derivation and implementation of a full 6d ekf-based solution to bearing-range slam. Technical report, University of Malaga, 2008.

J.L. Blanco and M. Bader. Ros mrpt navigation package. URL `http://wiki.ros.org/mrpt_navigation?distro=melodic`.

Luca Carlone, Vito Macchia, Federico Tibaldi, and Basilio Bona. Quaternion-based ekf-slam from relative pose measurements: observability analysis and applications. *Robotica*, 33(6):1250–1280, 2015. doi: 10.1017/S0263574714000678.

References III

dno89. Ekf-slam implementation. URL https://github.com/dno89/SLAM.

H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13(2): 99–110, 2006.

E. Fresk and G. Nikolakopoulos. Full quaternion based attitude control for a quadrotor. In *2013 European Control Conference (ECC)*, pages 3864–3869, 2013.

Andrew Gibiansky. Quadcopter dynamics and simulation, 2012. URL https://andrew.gibiansky.com/blog/physics/quadcopter-dynamics/.

Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN 0262201623.

References IV

A. Vitali. Exploiting the gyroscope to update tilt measure and e-compass. URL https://www.st.com/resource/en/design_tip/dm00286303-exploiting-the-gyroscope-to-update-tilt-measu pdf.