TP3: Localización basada en EKF: Predicción y Corrección

Introducción a la Robótica Movil

October 7, 2016

1 Introdución

En este TP se **continuará** con el trabajo realizado en talleres anteriores para lograr una **implementación completa de un sistema de localización** basado en el Filtro de Kalman Extendido.
Se trabajará mayormente sobre la fase de corrección o actualización de la medición (*measure update*) de manera de utilizar la información provista por el **modulo de detección de postes** (sensor láser) para corregir las predicciones generadas por el filtro.

1.1 Activar y desactivar la Corrección

Se trabajará sobre el mismo paquete provisto en el taller de Predicción, **robmovil_ekf**. Para activar la integración de la información provista por el sensor láser existe un parámetro de configuración en el archivo:
/robmovil_ekf/launch/ekf.launch

```
<param name="only_prediction" type="bool" value="true"/>
```

Cambiando el valor del parámetro **only_prediction** a **false**, se toman en cuenta los mensajes publicados por el nodo de detección de postes. El filtro es entonces "alimentado" con la información de los **centroides** calculados en **coordenadas polares respecto del robot**.

1.2 Mapa construido por el sistema

El sistema de localización construye un mapa con los postes detectados **al momento de ser iniciado**. La posición de los postes se almacena en **coordenadas cartesianas** y en referencia a la posición donde se encontraba el robot al momento de ser iniciado (en referencia al mundo).

Podrán acceder a esta información utilizando el vector global **map_landmarks** definido en el archivo:
/robmovil_ekf/localizer_ekf.h

Contiene las coordenadas cartesianas $(x_l, y_l, 0)$ representando las coordenadas del poste (o landmark) l.

NOTA: El tipo de los datos es **tf::Point**.

Ejercicio 1: Estimar la posición de un poste en el mapa

Toda medición recibida nos da información de algún poste que se encuentra cercano al robot.

Las nuevas mediciones ingresan al modelo por el método (**localizer_ekf.cpp**):

```
bool set_measure(const Vector& new_measure_z)
```

Lo primero que se tiene que resolver cuando se obtiene una nueva medición es calcular **la posición más probable** en que se encuentra **el poste dentro del mapa**.

- **Completar** el método:

  ```
  tf::Point measure2landmark(const Vector& measure)
  ```

  El cual debe recibir una medición en **coordenadas polares en referencia al robot** y devolver la posición estimada en **coordenadas cartesianas del poste en referencia al mapa**.

Ejercicio 2: Corresponder el poste detectado

Habiendo estimado la posición del poste en el mapa, lo siguiente que se requiere es definir **cual es su verdadera posición**. Para esto se requiere corresponder la posición estimada con alguno de los postes **pertenecientes al mapa**.

NOTA: La correspondencia será exitosa siempre y cuando el sistema mantenga una estimación de localización "suficientemente" buena.

- **Completar** el método:

  ```
  bool find_corresponding_landmark(
                    const tf::Point& measured_landmark,
                    tf::Point& corresponding_landmark,
                    float delta_radio)
  ```

  El cual debe recibir las coordenadas cartesianas estimadas de un poste (en referencia al mapa) y debe devolver la posición **real** de dicho poste en el mapa asignando la referencia **corresponding_landmark**. Para esto deberán "buscar" el poste del mapa que se encuentre **dentro de un determinado radio** tomando como centro la posición del poste estimada.

  NOTA: Pueden suponer que los postes se encuentran suficientemente separados.

Ejercicio 3: Modelo de sensado

Por último, habiendo establecido la posición donde se encuentra el poste en el mapa. Se debe utilizar el modelo de sensado para **predecir** "la manera en que

se debería haber observado" dicho poste. De esta manera el filtro de kalman extendido será capaz de contrastar la medición recibida originalmente con la medición predicha y refinar los diferentes estimadores.

El modelo de sensado $h(\vec{x})$ se notifica al filtro por medio del método:

```
void makeMeasure(void)
```

Asignando a la **variable global z** (ya definida) la medición predicha.

- **Completar** el método:

  ```
  Vector landmark2measure(const tf::Point& landmark)
  ```

  El cual debe recibir la posición de un poste en el mapa y debe devolver la medición predicha en referencia al robot y en coordenadas polares.

  NOTA: Leer los comentarios en el código. Existen variables locales auxiliares para facilitar la conversión de un marco de coordenadas a otro.

- Notificar las derivadas del modelo completando los métodos:

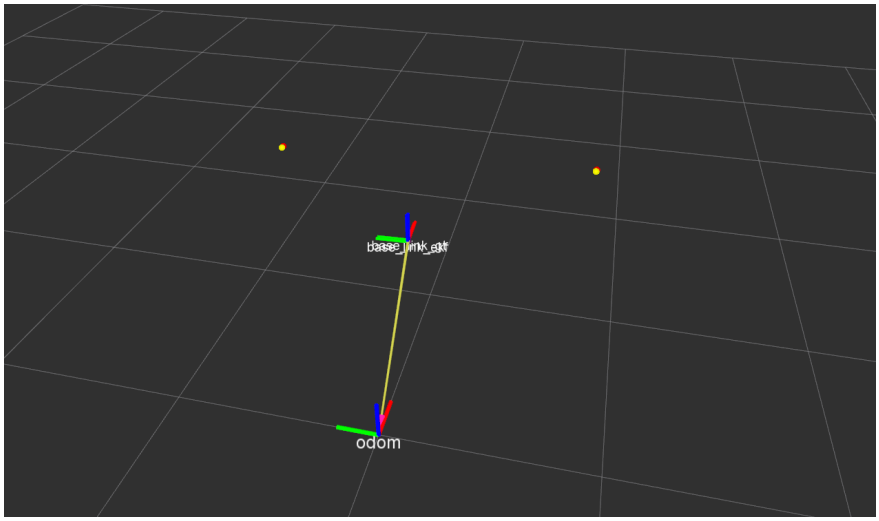
  ```
  void makeH(void)
  void makeBaseV(void)
  ```

  Donde `makeH` refiere al Jacobiano de h respecto de $\vec{x}$) y `makeV` al Jacobiano de f respecto de $\vec{v}$.

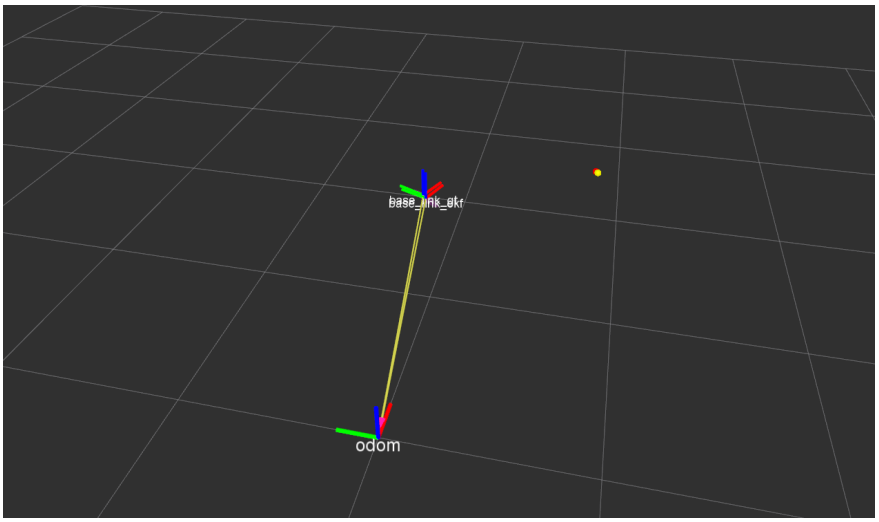
  NOTA: Para construir la matriz H deberán utilizar **la variable global correspondence_landmark** (ya definida) para realizar los cambios de marcos de coordenadas y establecer el valor de las derivadas.

Experiencias y preguntas

- 1) Mover el robot "en linea recta" hasta que solo sea capaz de detectar los dos postes del frente:



Luego girar al robot de manera de "perder de vista" uno de los postes:



¿Que empieza a suceder con la covarianza? ¿por que se debe esto?. Dar una captura de pantalla y explicar lo observado.

- 2) Plantear un Filtro de Kalman Extendido para el caso de un **robot omnidireccional**. Considerar que solo se tiene información de velocidad instantánea proveniente de los encoders y un sensor de postes basado en láser idéntico al trabajado en clase.
  Modelar el estado $\vec{x}$, las entradas de control $\vec{u}$, las mediciones $\vec{z}$, el ruido del actuador $\vec{w}$, el ruido del sensor $\vec{z}$, el modelo de movimiento $f(\vec{x}, \vec{u}, \vec{w})$, el modelo de sensado $h(\vec{x}, \vec{v})$ y los respectivos Jacobianos.

  NOTA: Considerar que aspecto del problema visto en clase es posible **simplificar**.