UNIVERSITÀ DEGLI STUDI DI PERUGIA

DIPARTIMENTO DI INGEGNERIA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA E ROBOTICA

Relazione di Progetto:

Localizzazione di un robot differenziale Duckie-Bot attraverso Extended Kalman Filter e sensori range finder con metodo ICP

Coordinatore del Lavoro

Prof. Paolo Valigi

Studente

Mehdi Belal

Anno Accademico 2017/2018

INDICE

1. INTRODUZIONE ... 3
2. OVERVIEW DEL SISTEMA ... 4
2.1. Processo di Localizzazione a Mappa Nota con EKF ... 4
2.2. Struttura del Robot Differenziale ... 5
2.2.1. Sensoristica del Robot ... 5
2.3. Struttura del Progetto ... 6
2.3.1. Struttura del package mybot_ekf_localization ... 7
3. PROCESSO DI GENERAZIONE DELLA MAPPA ... 10
4. PROCESSO DI LOCALIZZAZIONE ... 11
4.1. Struttura dei Dati ... 11
4.2. Motion Model e Predict Step ... 11
4.3. Filtro di Kalman ... 12
4.4. Processo di Misurazione ... 13
4.4.1. Iterative Closest Point ... 13
4.5. Measurement Model e Correction Step ... 16
4.6. Gestione dei Sistemi di Rifermento ... 18
4.7. Risultato del Processo di Localizzazione ... 19
5. PROBLEMI RISCONTRATI, SOLUZIONI ADOTTATE E CONSIDERAZIONI FINALI ... 20
5.1. Utilizzo del filtro di Kalman classico ... 20
5.2. Caso particolare: ambiente diverso dalla mappa nota del sistema ... 20
5.3. Problema: caso di disallineamento nei sistemi di riferimento ... 21
5.4. Eventuale Implementazione in ambiente reale ... 22
5.5. Possibili sviluppi ... 22

1. INTRODUZIONE

Questo lavoro ha come obbiettivo l'implementazione di un sistema di localizzazione per un robot differenziale, riconducibile alla piattaforma duckiebot, in ambiente ROS. Il lavoro è composto da un package ROS, che se eseguito in ambiente simulato, consente di visualizzare il risultato della localizzazione.

Il problema di localizzazione che viene preso in considerazione è un problema a mappa nota, in questo lavoro viene anche mostrato uno strumento utilizzato per generare una mappa dell'ambiente nel quale dovrà localizzarsi il robot, in una rappresentazione coerente con il processo di localizzazione. Lo strumento di misurazione preso in considerazione in questo lavoro è un *range finder*, nello specifico un laser scanner; uno strumento in grado di rilevare ostacoli nell'ambiente, restituendo il risultato come una nuvola di punti.

Per la determinazione della posizione si fa uso del Kalman Filter, uno strumento matetico ricorsivo, in grado di determinare lo stato di un sistema dinamico a partire da misure soggette a rumore. Lo stato è modellato come una variabile aleatoria gaussiana, così come tutte le grandezze prese in considerazione nel sistema; nel processo di localizzazione in realtà si fa uso dell'Extended Kalman Filter, uno strumento in grado di usare lo stesso paradigma precedente anche in presenza di modelli non lineari.

Nel sistema simulato si fa uso di due ambienti:

- Gazebo: ambiente di simulazione e prototipazione robotica
- RViz: tool di visualizzazione 3D per ROS

Il risultato dell'operazione di localizzazione è una rappresentazione visuale su RViz della posizione predetta e dell'incertezza di tale predizione, il formato del risultato sarà dettagliato meglio del seguito.

2. OVERVIEW DEL SISTEMA

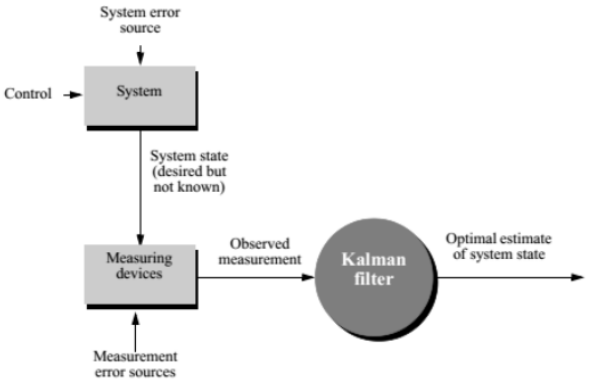
2.1. Processo di Localizzazione a Mappa Nota con EKF

Il problema della localizzazione è il problema di ***determinare la posa di un robot (posizione e orientamento)*** nell'ambiente, facendo uso e rispettando una rappresentazione dell'ambiente (definita da una mappa, in una specifica rappresentazione). Il problema della localizzazione è un problema di valutazione di una stima, tenendo conto dell'incertezza dovuta a attuatori e sensori a cui sono legate le misurazioni.

Extended Kalman Filter è uno dei metodi che possono essere utilizzati per determinare la stima della posa di un robot, questo metodo è un metodo stocastico e non deterministico, il che significa che la grandezza che viene determinata è riferita a una probabilità, ogni stima deve essere accompagnata da una stima dell'incertezza, come la covarianza associata a quel dato. EKF è un metodo basato sull' algoritmo di Bayes, una metodologia in cui iterativamente si cerca di determinare:

$$bel(x_t) = p(x_t|u_t, z_t)$$

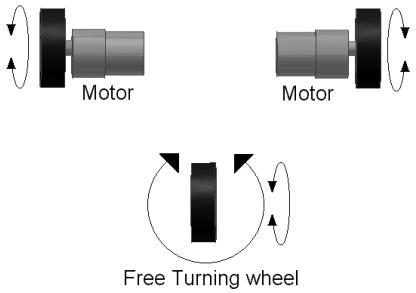
La grandezza $bel(x_t)$ è detta belief, e rappresenta la stima probabilistica dello stato, in funzione dell'ingresso u_t e della misurazione z_t. Questa classe di algoritmi è iterativo e si basa sull'esecuzione di due passaggi:



- ***Prediction step***: calcolo della belief in funzione degli ingressi e dello stato precedente, senza tenere in considerazione le misurazioni. Bisogna conoscere il motion model, ovvero il modello che evolve lo stato in funzione della cinematica del robot,
- ***Correction step***: integrazione delle misurazioni esterne, in modo da ottenere una stima più accurata, questo passaggio richiede una rappresentazione della mappa dell'ambiente in modo da dare un significato consistente alle misurazioni. In questo lavoro lo strumento di misurazione è rappresentato da un sensore laser.

Il Kalman Filter è un filtro di Bayes in cui si fa uso di un modello parametrico: tutte le grandezze vengono ricondotte a delle gaussiane in modo da semplificare il modello, inoltre elaborando una gaussiana attraverso un modello lineare, il risultato è sempre una gaussiana. Si parla di Extended Kalman Filter quando in presenza di un modello non lineare viene effettuata una operazione di linearizzazione per mantenere questa proprietà. In figura una sintesi del processo di localizzazione con EKF.

2.2. Struttura del Robot Differenziale



Un robot **differential drive** è un robot mobile il cui movimento è basato su due ruote azionate separatamente, poste su entrambi i lati del corpo del robot. Il robot può quindi cambiare la sua direzione variando la velocità di rotazione relativa delle sue ruote, e non richiede un movimento di sterzo aggiuntivo. Per bilanciare il robot, è possibile aggiungere ruote aggiuntive che fungono da punto di appoggio passivo.

Una approssimazione del motion model può essere stimata a partire da l'integrazione del movimento con un modello puntuale; sia $p = [x \quad y \quad \theta]^T$ la posizione del robot descritta in maniera puntuale e sia $u = [v \quad \theta]^T$ l'ingresso del robot, l'evoluzione della posizione:

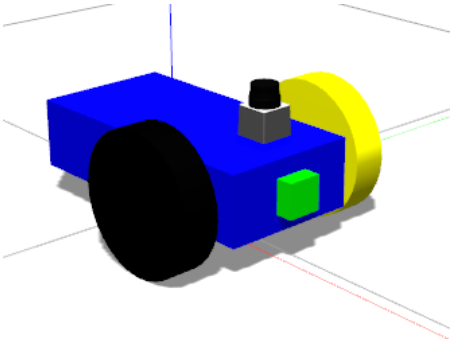
$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = \omega \end{cases}$$

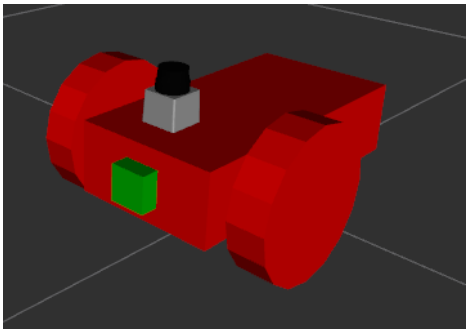
Che può essere discretizzata con:

$$\begin{cases} x_{k+1} = x_k + vT\cos\left(\theta + \dfrac{\omega T}{2}\right) \\ y_{k+1} = y_k + vT\sin\left(\theta + \dfrac{\omega T}{2}\right) \\ \theta_{k+1} = \theta_k + \omega T \end{cases}$$

A partire da questa legge di evoluzione della posizione si può determinare il modello di posizione per EKF, il motion model viene descritto nel capitolo intitolato '*Processo di localizzazione*'.

In ROS l'architettura, la geometria e l'inerzia del robot sono descritte in un file chiamato mybot.xacro, descritto nel formato Universal Robotic Description Format (URDF) si tratta di una macro XML che contiene la descrizione delle varie componenti del robot e come sono legate tra loro. Questo file consente agli ambienti di simulazione di ROS di emulare un robot con queste caratteristiche. Nelle due figure vengono mostrati i modelli del robot differenziale utilizzati nel lavoro in Gazebo e RViz.





2.2.1. Sensoristica del Robot

Il robot utilizzato per questo lavoro fa uso di un sensore laser come strumento eterocettivo per la fase di measurement correction. Questo genere di sensori è in grado di misurare la distanza da un ostacolo e di restituire come risultato una nuvola di punti che rappresenta gli ostacoli che sono stati individuati, per questo lavoro si è fatto uso di un sensore laser in grado di lavorare solamente su un asse.

La rappresentazione della mappa deve essere coerente con lo strumento di misura, per questo motivo si è fatto uso di una Occupancy Grid, ovvero una griglia di occupazione in cui ogni cella della griglia

rappresenta uno stato *occupato/non occupato*. Entrambe le rappresentazioni possono essere ricondotte a delle nuvole di punti, garantendo la confrontabilità tra le due grandezze. La struttura del sensore laser, e come questa si collega al resto del robot, è sempre definita in mybot.xacro.

```
<!-- hokuyo -->
  <gazebo reference="hokuyo">
    <sensor type="ray" name="head_hokuyo_sensor">
      <pose>0 0 0 0 0 0</pose>
      <visualize>false</visualize>
      <update_rate>40</update_rate>
      <ray>
        <scan>
          <horizontal>
            <samples>720</samples>
            <resolution>1</resolution>
            <min_angle>-1.570796</min_angle>
            <max angle>1.570796</max angle>
          </horizontal>
        </scan>
        <range>
          <min>0.10</min>
          <max>30.0</max>
          <resolution>0.01</resolution>
        </range>
        <noise>
          <type>gaussian</type>
          <mean>0.0</mean>
          <stddev>0.01</stddev>
        </noise>
      </ray>

      <plugin name="gazebo_ros_head_hokuyo_controller" filename="libgazebo_ros_laser.so">
       <topicName>/mybot/laser/scan</topicName>
       <frameName>hokuyo</frameName>
     </plugin>
    </sensor>
  </gazebo>
```

L'elemento `<gazebo>` è un'estensione dell'URDF utilizzato per specificare le proprietà aggiuntive necessarie per scopi di simulazione in Gazebo. Dunque le caratteristiche del sensore utilizzato sono:

Update rate	40 Hz	**Numero di campioni**	720
Angolo coperto	$\pi\ rad$ - 180°	**Range resolution**	0.01
Range min	0.1 m	**Range max**	30 m
Media noise	0	**Cov. noise**	0.01

Al sensore laser viene aggiunto del rumore gaussiano per simulare un errore di misurazione. Si suppone che il sensore sia affetto da rumore gaussiano di media nulla e varianza 0.01 Questo file consente di modificare i parametri del laser in funzione delle esigenze, ad esempio nel caso di simulazioni di modelli reali di hardware; è sufficiente modificare i diversi parametri con quelli delle specifiche a disposizione.

2.3. Struttura del Progetto

ROS (*acronimo di Robot Operating System*) è un insieme di framework per lo sviluppo e la programmazione di robot, fornisce i servizi standard di un sistema operativo, come: astrazione dell'hardware, controllo dei dispositivi tramite driver, comunicazione tra processi, gestione delle applicazioni e altre funzioni di uso comune. Il livello applicativo può essere considerato composto dai package, pacchetti contenenti applicazioni e codice utilizzati da processi client in ROS. I package in questo lavoro:

- **mybot_ekf_localization**: package sviluppato per questo progetto, contiene il codice del programma per la localizzazione e alcuni elementi che fanno uso di altri package, come per la creazione della mappa.
- **mybot_description**: package che contiene la descrizione del robot differenziale preso in considerazione; nella cartella urdf è contenuta la descrizione del robot, nel file mybot.xacro. Viene riportata la descrizione strutturale, geometrica, inerziale dell'oggetto da rappresentare, in modo che gli ambienti di simulazione siano in grado di replicarlo come se fosse un oggetto reale.
- **mybot_gazebo**: package che contiene gli elementi per la simulazione dell'ambiente virtuale in ROS, in modo da replicare un ambiente reale in ROS, questo package contiene la descrizione degli elementi dell'ambiente (nella cartella models) e la descrizione dell'ambiente in generale (nella cartella worlds).
- **slam_gmapping**: package per la creazione della mappa per l'ambiente, la sua implementazione dipende completamente dal package successivo
- **gmapping**: package che contiene un wrapper ROS per Gmapping di OpenSlam. Il package gmapping fornisce degli strumenti per lo SLAM basato su laser (*Simultaneous Localization and Mapping*), attraverso il nodo ROS slam_gmapping. Usando slam_gmapping, si può creare una mappa nel formato Occupancy Grid 2D (come una planimetria di un edificio) dai dati laser e di posa raccolti dal robot mobile. Questa funzionalità verrà utilizzata nel processo di generazione della mappa
- **laser_geometry**: package che contiene una classe per la conversione da una scansione laser 2D definita dal tipo sensor_msgs/LaserScan in una nuvola di punti definita dal tipo sensor_msgs/PointCloud. Questa funzionalità verrà utilizzata per rendere confrontabili le scansioni laser e i dati della mappa durante il passaggio di measurement correction, attraverso l'algoritmo ICP (iterative closest point).

2.3.1. Struttura del package mybot_ekf_localization

Il package mybot_ekf_localization ha come obbiettivo di fornire all'utente uno strumento per la localizzazione nell'ambiente ROS di un robot differenziale. L'esecuzione di questo package avviene attraverso due launch di ROS:

- **localization.launch**: per eseguire il processo di localizzazione, una volta che è stato definito l'ambiente di simulazione e la sua mappa.
- **mapping.launch**: per eseguire il processo di cattura della mappa dell'ambiente, questa operazione richiede alcuni passaggi illustrati nel capitolo '*Processo di generazione della mappa*'.

Per l'esecuzione non vengono lanciati direttamente i file .launch ma vengono utilizzati degli script Linux che consentono l'esecuzione di altre operazioni di diagnostica, come grafici dell'errore di predizione della posizione durante il processo di localizzazione.

Topic sottoscritti
I topic sottoscritti da parte del nodo di localizzazione sono:

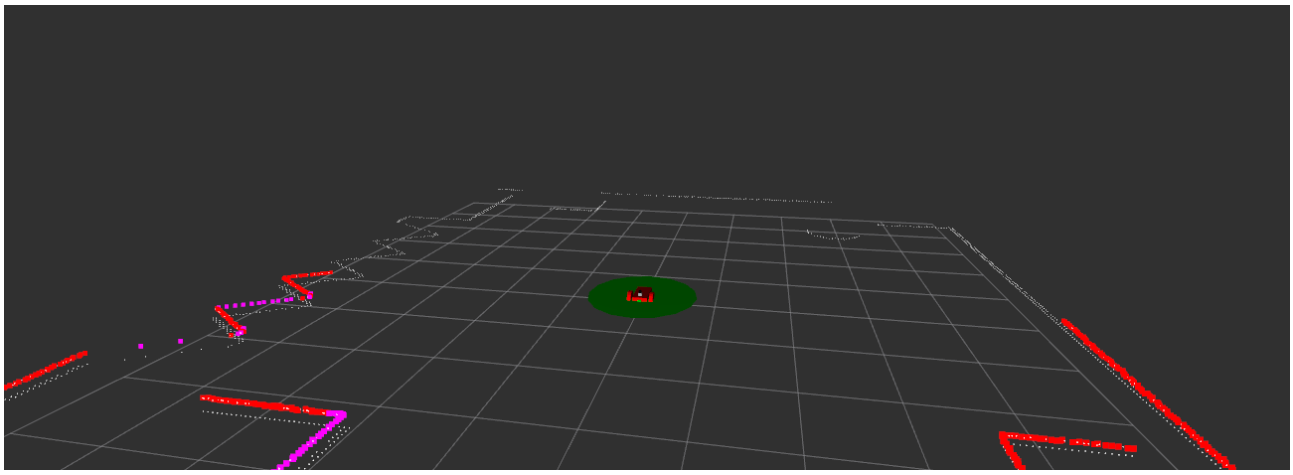
- **map**: ovvero il topic in cui vengono pubblicate le informazioni della mappa, nel formato nav_msgs/OcupnacyGrid,
- **laserscan**: topic in cui vengono registrate le scansioni del sensore laser, nel formato sensor_msgs/LaserScan,
- **mybot/odom**: topic in cui viene pubblicata la posizione esatta del robot determinata dall'ambiente di simulazione, consente di determinare la differenza dalla posizione predetta.

- **mybot/cmd_vel**: topic in cui vengono registrate le informazioni sull'input fornito al sistema, nel formato geometry_msgs/Twist.

Topic pubblicati

I topic pubblicati da parte del nodo di localizzazione sono:

- **map_pub**: topic in cui viene pubblicata la mappa nel formato sensor_msgs/PointCloud2, per la visualizzazione e il confronto in RViz,
- **laser_pub**: in cui viene pubblicato il risultato del laser nel formato sensor_msgs/PointCloud2, per la visualizzazione e il confronto in RViz,
- **draw_position**: in cui viene pubblicato un marker nel formato visualization_msgs/Marker che rappresenta il risultato della stima della localizzazione. EKF restituisce come risultato una gaussiana, per cui il marker in questione è un disco centrato nella posizione stimata, e che cambia dimensione in funzione dell'incertezza, ovvero della matrice di covarianza.



Contenuto del package

| CMakeLists.txt | package.xml | README.txt | +---include | \\---conditional_pdf | nonlinearanalyticconditionalgaussianmobile.h | +---launch | localization.launch | mapping.launch | +---script | localization.sh | mapping.sh | | +---maps | mymap.pgm | mymap.yaml | +---rviz | amcl1.rviz | amcl2.rviz | \\---src | ekf.cpp | ekf.hpp | main.cpp | \\---conditional_pdf nonlinearanalyticconditionalgaussianmobile.cpp nonlinearanalyticconditionalgaussianmobile.h	Launch per la localizzazione e per la creazione della mappa Script linux da lanciare per avviare processi di localizzazione e creazione della mappa Cartella contente la mappa dell'ambiente in cui localizzarsi Configurazioni per l'ambiente di simulazione RViz, nel caso di mappatura o localizzazione Source del package, il file ekf,cpp e il suo header contengono il codice del nodo da lanciare per la localizzazione, la cartella conditional_pdf contiene il motion model non lineare

3. PROCESSO DI GENERAZIONE DELLA MAPPA

Questo passaggio ha come obbiettivo la generazione di una mappa dell'ambiente di simulazione, in un formato adeguato. Per generare la mappa si fa uso del package ROS gmapping che mette a disposizione gli strumenti per questa operazione.

Il launch di ROS:

```
roslaunch mybot_ekf_localization mapping.launch
```

consente di avviare tutti gli elementi per l'esecuzione dell'ambiente virtuale, e di eseguire il nodo di gmapping. A partire dalle scansioni laser viene creata una Occupancy Grid 2D composta da un numero fisso di celle (*viene impostato per default a un numero di elementi molto alto, per avere molto margine nell'ambiente*) e laddove il sensore laser rileva un ostacolo, anche nella mappa viene registrata la stessa informazione.

Il sensore laser a disposizione è un sensore con un angolo di misurazione di 180°, per cui per avere una rappresentazione di tutto l'ambiente è necessario far muovere il robot. gmapping consente di effettuare SLAM (*Simultaneous Localization and Mapping*), per cui il sistema è in grado di identificare la mappa anche durante il movimento del robot.

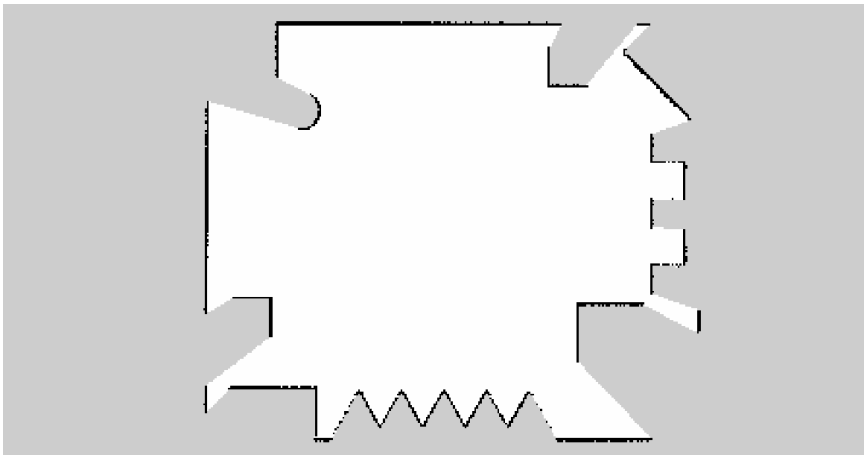
Una volta che il sistema ha catturato elementi sufficienti da considerare la rappresentazione adeguata viene eseguito:

```
rosrun map_sever map_saver mymap
```

Questo nodo consente di salvare la mappa con il come mymap. Il salvataggio prevede che vengano memorizzati due file:

- mymap.yaml: formato per la serializzazione, contiene una serie di metadati della mappa
- mymap.pgm: formato per memorizzare dati grayscale, contiene la Occupancy Grid della mappa

Questi due elementi devono essere copiati nella cartella maps del package mybot_ekf_localization.



4. PROCESSO DI LOCALIZZAZIONE

Nel seguito viene descritto il processo di localizzazione, facendo riferimento ai passaggi di un algoritmo basato sulla struttura di Bayes.

4.1. Struttura dei Dati

L'obbiettivo della localizzazione è stimare la posa 2D del robot nel formato $p = [x \quad y \quad \theta]^T$.

Il filtro di Kalman fa uso di grandezze gaussiane per stimare lo stato di una variabile, per cui la stima della localizzazione è data da una funzione di densità di probabilità che ogni volta viene processata dal filtro in funzione del suo stato. In questo modo lo stato viene stimato a partire dalla media della gaussiana (valore atteso), mentre l'incertezza è data dalla matrice di covarianza della stessa.

Anche motion model e measurement model (*descritti nel dettaglio nel seguito*) sono rappresentati da delle funzioni di densità di probabilità, rispettivamente $p(x_t|x_{t-1}, u_t)$ e $p(z_t|x_t)$, per rappresentare le grandezze stocastiche si fa uso della libreria BFL: Bayesian Filtering Library.

La mappa è rappresentata da una OccupancyGrid, mente le misurazioni sono nel formato LaserScan, entrambe vengono poi trasformate nel formato PointCloud per poterle confrontare, e per poter applicare l'algoritmo ICP (*Iterative Closest Point*) in fase di measurement correction.

4.2. Motion Model e Predict Step

Nel caso del filtro di Kalman esteso il modello di sistema è dato da:

$$x_t = f(x_{t-1}, u_t) + \omega_t, \qquad con\ \omega_t = \mathcal{N}(0, Q_t)$$

Viene creato un `system_model`:

```
BFL::ColumnVector system_noise_mean(3);
system noise mean = 0.0;

BFL::SymmetricMatrix Q(3);
Q = 0.0;
Q(1,1) = 1.0;
Q(2,2) = 1.0;
Q(3,3) = 1.0;

//incertezza gaussiana
BFL::Gaussian system uncertainty(system noise mean, Q);
//modello
system_pdf = boost::shared_ptr<BFL::NonLinearAnalyticConditionalGaussianMobile>
(new  BFL::NonLinearAnalyticConditionalGaussianMobile(system uncertainty));

system model = boost::shared ptr<BFL::AnalyticSystemModelGaussianUncertainty>
(new BFL::AnalyticSystemModelGaussianUncertainty (system_pdf.get()));
```

Che rappresenta il motion model del sistema, questo elemento viene inizializzato a partire da una PDF del tipo `NonLinearAnalyticConditionalGaussianMobile`; ovvero una $p(x_t|x_{t-1}, u_t)$ non lineare a partire da un valore di incertezza additivo che assume valori secondo una gaussiana a media nulla e varianza unitaria, la quale rappresenta il modello odometrico/cinematico del robot, in seguito viene

creato il system model del tipo `AnalyticSystemModelGaussianUncertainty` che rappresenta il modello del sistema.

All'interno di `NonLinearAnalyticConditionalGaussianMobile.cpp`, è presente la descrizione dell'evoluzione della media nel sistema non lineare (*che evolve come lo stato*), e l'evoluzione della covarianza (*grazie allo Jacobiano del modello di moto*) [2]:

```
 //evoluzione media
ColumnVector NonLinearAnalyticConditionalGaussianMobile::ExpectedValueGet() const {
    ColumnVector state = ConditionalArgumentGet(0);
    ColumnVector vel  = ConditionalArgumentGet(1);
    state(1) += cos(state(3)) * vel(1);
    state(2) += sin(state(3)) * vel(1);
    state(3) += vel(2);
}

//evoluzione covarianza, restituisce lo Jacobiano del modello di misura
Matrix NonLinearAnalyticConditionalGaussianMobile::dfGet(unsigned int i) const {
    if (i==0)//derivative to the first conditional argument (x){
        ColumnVector state = ConditionalArgumentGet(0);
        ColumnVector vel = ConditionalArgumentGet(1);
        Matrix df(3,3);
        df(1,1)=1;
        df(1,2)=0;
        df(1,3)=-vel(1)*sin(state(3));
        df(2,1)=0;
        df(2,2)=1;
        df(2,3)=vel(1)*cos(state(3));
        df(3,1)=0;
        df(3,2)=0;
        df(3,3)=1;
        return df;
      }
}
```

Nelle EKF la media della predizione evolve come l'evoluzione dello stato (*come anche nel caso del filtro di kalman classico*), mentre nel caso della covarianza, dato che il legame non è lineare, non sarebbe sufficiente un semplice prodotto tra matrici; utilizzando lo jacobiano, EKF è in grado di effettuare una sorta di linearizzazione e estendere la teoria del filtro di Kalman anche per sistemi non lineari. Nella struttura che si sta descrivendo `dfGet` consente di determinare lo jacobiano del legame non lineare $f(\cdot)$, la struttura del filtro di Kalman attraverso questa grandezza sarà in grado di determinare come si sviluppa l'incertezza[2].

Il passaggio di predict step, una volta ottenute le grandezze di controllo input, prevede:

```
filter->Update(system_model.get(),input);
```

Di fatto il filtro di kalman, descritto nel seguito, contiene le informazioni per l'aggiornamento del sistema a partire da motion model e input.

4.3. Filtro di Kalman

Il filtro di Kalman utilizzato in questo lavoro è uno strumento fornito dalla libreria Bayesian Filtering Library, le motivazioni dell'utilizzo di questa struttura sono descritte nel capitolo successivo. In fase di inizializzazione viene creato un oggetto del tipo `ExtendedKalmanFilter` della libreria BFL [5]:

```
BFL::Gaussian prior(prior_mu,prior_cov); //gaussiana
filter=boost::shared_ptr<BFL::ExtendedKalmanFilter> (new BFL::ExtendedKalmanFilter(&prior));
```

L'oggetto `filter` viene creato a partire da una prior, rappresenta da una semplice gaussiana a media nulla e varianza unitaria per inizializzare il sistema, questo rappresenta la condizione iniziale del sistema, ovvero il robot che si trova all'origine del sistema di riferimento fisso del sistema. L'oggetto `filter` presenta i metodi per l'aggiornamento del filtro e per ottenere il risultato del processo di localizzazione:

- `filter->update(SystemModel *const sysmodel, const StateVar &u)`: aggiornamento del filtro durante operazione di prediction, ovvero a partire dalle informazioni odometriche
- `filter->update(MeasurementModel*const measmodel, const MeasVar &z)`: aggiornamento del filtro durante operazione di correction, ovvero con le informazioni di misurazione
- `Posterior = filter->PostGet()`: ottiene la posterior, il risultato del filtro sotto forma di grandezza gaussiana
    - `Posterior->ExpectedValueGet()`: ottiene il valore atteso della posterior
    - `Posterior->CovarianceGet()`: ottiene il valore atteso della posterior

Per `posterior` non si intende il risultato del passaggio di correction, ma in generle il risultato del filtro, anche dopo il passaggio di prediction il risultato verrà chiamato posterior e non prior.

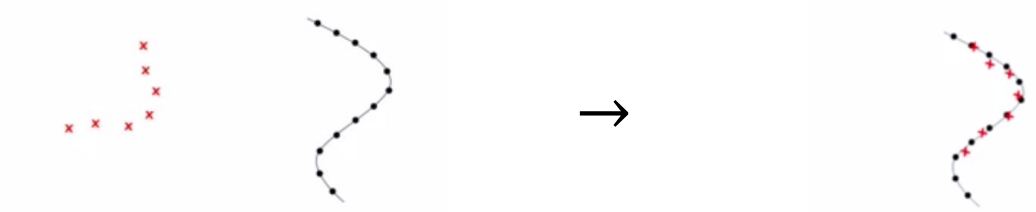
4.4. Processo di Misurazione

Una volta ottenute le misurazioni da parte del sensore, queste devono essere, in primo luogo confrontate con gli elementi della mappa per determinare le corrispondenze, e in seguito elaborate per ottenere una grandezza utile al passo di measurement correction.

4.4.1. Iterative Closest Point

Il processo di determinazione delle corrispondenze può essere molto oneroso, soprattutto quando si ha a che fare con grandezze costituite da molti elementi (come nuvole di punti), per questo lavoro si è scelto di utilizzare il metodo **ICP** (***Iterative Closest Point***), in modo da risolvere in una volta sola sia il problema delle corrispondenze, sia l'elaborazione delle misurazioni.

ICP è un algoritmo iterativo utilizzato per determinare la differenza, in termini di rotazione e di traslazione, tra due nuvole di punti che hanno subito una trasformazione rigida, la PointCloud delle scansioni laser sarà sempre un sottoinsieme della PointCloud della mappa (*infatti la mappa del sistema viene costruita con questo stesso strumento di misurazione*), per questo motivo questo metodo può essere utilizzato in questo contesto. Questo algoritmo consente di identificare la trasformazione rigida tra le due PointCloud.



ICP è un algoritmo che ha come obbiettivo di determinare corrispondenze nelle nuvole di punti, e di stimare la trasformazione che minimizza la distanza tra le corrispondenze, si tratta di un algoritmo iterativo basato su Expectation-Maximization (*metodo iterativo per determinare la massima stima a*

posteriori dei parametri in modelli stocastici); l'obbiettivo è determinare l'errore di mappatura, in funzione della rotazione R e del vettore di traslazione Δt [4]:

$$(R, \Delta t) = \arg\min \sum \left|\left|d(x_i, y_i)\right|\right|^2$$

La libreria Point Cloud Library (PCL) fornisce degli strumenti per implementare l'algoritmo ICP [3], siano date due PointCloud `laser_cloud` e `map_cloud`:

```
icp.setInputSource(laser_cloud);
icp.setInputTarget(map_cloud);

//criteri
icp.setTransformationEpsilon (icp_optimization_epsilon);
icp.setMaxCorrespondenceDistance(max_correspondence_distance);

//numero iterazioni
icp.setMaximumIterations(max_iterations); //200 iterazioni
//RANSAC per outliers
icp.setRANSACIterations (ransac_iterations);
icp.setRANSACOutlierRejectionThreshold(ransac_outlier_threshold);

//risutalto
pcl::PointCloud<point_type> Final;
icp.align(Final);

Final.header.frame_id = map_frame;
pcl_conversions::toPCL(time_, Final.header.stamp);

if(icp.hasConverged()){
    ROS_INFO("ICP converge correttamente -> measurement update");
}else{
    ROS_WARN("ICP non converge");
}
```

Vengono impostati alcuni valori come:

- Massima distanza per ottenere una corrispondenza
- Numero massimo di iterazioni
- Numero massimo di iterazioni RANSAC
- Valore massimo di tasso di outliers

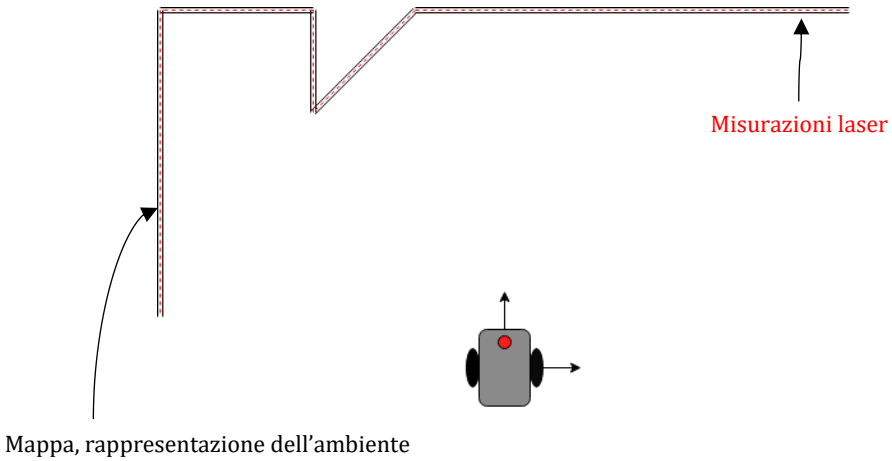
RANSAC gira nell'algoritmo come strumento di consensus, ovvero consente di capire l'effetto degli outlier nell'elaborazione del sistema. Si è visto che per rendere l'effetto del sensore laser più simile a un modello reale, viene aggiunto rumore di misurazione, può capitare che false corrispondenze causino problemi nel processo di allineamento, l'utilizzo di RANSAC consente di non cadere in un minimo locale. L'algoritmo può terminare in due casi:

1. Viene raggiunto il numero massimo di iterazioni impostate in `icp.setMaximumIterations(max_iterations);`
2. La trasformazione a una iterazione successiva non differisce dalla precendente per più di un fattore descritto dal parametro `icp.setTransformationEpsilon (icp_optimization_epsilon);` il che significa che la trasformazione ottenuta può essere considerata quella definitiva e si può fermare l'algoritmo prima di completare tutte le iterazioni

Il metodo `icp.hasConverged()` informa se il processo è andato a buon fine, determinando le corrispondenze e la trasformazione rigida tra le due PointCloud. Questa procedura è utile nella fase di measurement model in quanto la point cloud ottenuta dal sensore laser viene trasformata nel sistema di rifermento fisso (della mappa), in cui si trova anche la point cloud della mappa; questo significa che

la trasformazione che intercorre tra le due point cloud, è la stessa che intercorre tra il sistema di rifermento della mappa e il sistema di riferimento odom, quello relativo allo spostamento del robot. A partire da questa informazione quindi, si può procedere con il processo di aggiornamento del filtro a seguito della misurazione.

In Figura 1 è mostrato il caso di partenza, i sistemi di rifermento fisso e del robot sono allineati, per cui la matrice di rototraslazione è nulla.



$$(R,t) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figura 1

In questa condizione le point cloud di mappa e di laser sono allineate, questo significa che l'innovazione non apporta nessuna modifica al passaggio di measurement correction nella stima della media.

Nel momento in cui le due point cloud non si trovano più allineate, e il robot inizia a muoversi (come in Figura 2), il sistema di misurazione restituirà una matrice di roto-traslazione non nulla; in realtà di questa matrice interessano solamente la traslazione lungo x, la traslazione lungo y e l'angolo intorno all'asse z.

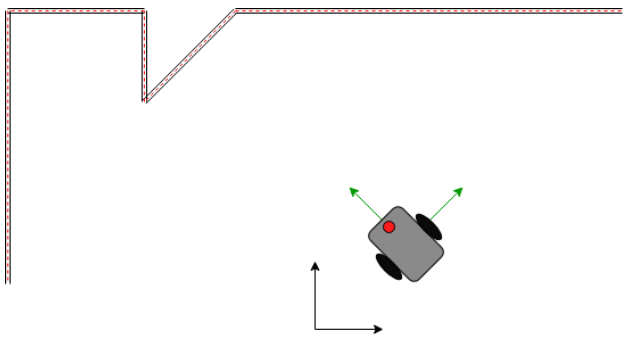


Figura 2

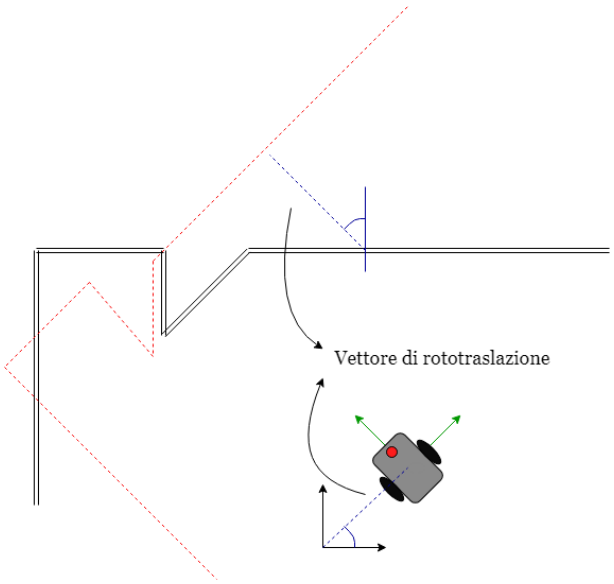


Figura 3

$$(R,t) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 & \mathrm{d_x} \\ \sin\gamma & \cos\gamma & 0 & \mathrm{d_y} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In Figura 3 viene mostrato che quando la point cloud dei laser viene riportata nel sistema di riferimento della mappa questa differisce dalla point cloud della mappa per una trasformazione rigida, che può essere appunto descritta da i tre parametri $[d_x \quad d_y \quad \gamma]$.

4.5. Measurement Model e Correction Step

Il processo di elaborazione delle misurazioni precedente precede il passo di measurement correction. Il metodo `icp.getFinalTransformation()` consente di ottenere la trasformazione rigida tre le due PointCloud, il che rappresenta una misurazione della posizione del robot nella mappa. Durante la fase di inizializzazione viene inizializzato anche il measurement model:

```
// measurement noise, scalare
BFL::ColumnVector measurement_noise_mean(3);
measurement_noise_mean = 0.0;

BFL::SymmetricMatrix R(3);
R(1,1) = 1.0;
R(2,2) = 1.0;
R(3,3) = 1.0;

//incertezza gaussiana
BFL::Gaussian measurement_uncertainty(measurement_noise_mean, R);


// modello di misura lineare, matrice H
BFL::Matrix H(3,3);
H = 0.0;
H(1,1) = 1.0;
H(2,2) = 1.0;
H(3,3) = 1.0;

//crea modello in funzione di:
// -- matrice H
// -- incertezza gaussiana
measurement_pdf=boost::shared_ptr<BFL::LinearAnalyticConditionalGaussian>
    (new BFL::LinearAnalyticConditionalGaussian(H, measurement_uncertainty));

measurement_model=boost::shared_ptr<BFL::LinearAnalyticMeasurementModelGaussianUncertainty>
    (new BFL::LinearAnalyticMeasurementModelGaussianUncertainty(measurement_pdf.get()));
```

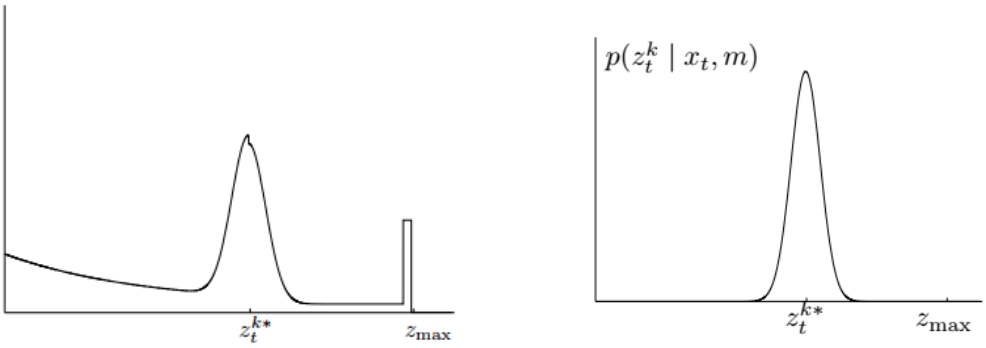
A partire dalla matrice H e da una gaussiana che rappresenta incertezza nel processo di misurazione, viene determinata `measurement_pdf`, ovvero la distribuzione $p(z|x)$; a partire da questa viene creato un oggetto `LinearAnalyticMeasurementModelGaussianUncertainty`, ovvero il measurement model del sistema. La matrice H è la matrice che lega $z = Hx$, inizialmente viene presa la matrice unitaria, poi viene rideterminata ad ogni operazione di misurazione come illustrato nel seguito.

In genere durante la modellazione di un measurement model per un sensore laser si considerano diversi fattori, come [1]:

- Range massimo del dispositivo non sufficiente a determinare gli oggetti,
- La possibilità di oggetti inaspettati nella scena,
- Errore di misurazione, dovuto all'utilizzo di un dispositivo reale,
- Fallimento nel processo di misurazione, lo strumento restituisce la portata massima.

In questo lavoro si prende in considerazione un ambiente simulato e controllato, il range massimo del dispositivo virtuale, come descritto precedentemente, è di $30\,m$, valore molto maggiore rispetto all'ordine di grandezza della mappa utilizzata. Inoltre, come verrà descritto successivamente, nel caso in cui si presentino oggetti inaspettati nella scena, o in generale il sistema laser rileva una point cloud che non coincide esattamente con quella della mappa, il sistema è robusto, purché queste variazioni non superino eccessivamente il fattore di outliers tollerato nel sistema.

Inoltre c'è da considerare il fatto che le misurazioni prese in considerazione non sono la distanza dagli ostacoli elaborata dal laser, ma il vettore che lega la trasformazione tra i due sistemi di riferimento.



Per queste ragioni per il measurement model si considera un modello lineare, in questo modo non vengono descritte tutte le non idealità descritte precedentemente, ma se si considerano gli accorgimenti descritti questo è in grado di descrivere sufficientemente bene il sistema di misurazione.

In generale il passaggio di measurement correction, con measurement model lineare, richiede diversi passaggi tra cui:

• Calcolo innovazione	$\tilde{y} = z_t - Cx_{t\|t-1}$ $S_t = R_t + C\,P_{t\|t-1}C^T$
• Calcolo guadagno del filtro	$K_t = P_{t\|t-1}C^T S_t^{-1}$
• Calcolo *posterior*	$x_{t\|t} = x_{t\|t-1} + K_t\,\tilde{y}_t$ $P_{t\|t} = P_{t\|t-1} - K_t C P_{t\|t-1}$

Il guadagno del filtro di kalman è una grandezza che rappresenta il peso che l'effetto della misurazione avrà sul calcolo della posterior. In questo caso la struttura dell'oggetto `ExtendedKalmanFilter` mantiene nel suo stato queste informazioni (calcolo dell'innovazione a partire da stato predetto e misurazione, calcolo covarianza innovazione, calcolo guadagno del filtro)

Sperimentalmente si hanno avuto risultati migliori con questa metodologia piuttosto che con l'utilizzo del filtro di Kalman in cui si calcolano innovazione, guadagno e posterior; l'utilizzo di questa struttura ha garantito maggiore robustezza. Durante il passaggio di measurement correction deve essere anche aggiornato il measurement model.

```
//risultato ICP
Eigen::Matrix4f correction_transform = icp.getFinalTransformation();

//media osservazioni
BFL::ColumnVector observation_mean(3);
observation_mean(1)=correction_transform(0,3);
observation_mean(2)=correction_transform(1,3);
observation_mean(3)=correction_transform.block<3,3>(0,0).eulerAngles(0,1,2)(2);
```

```
//covarianza osservazioni
BFL::SymmetricMatrix observation_noise(3);
double icp_fitness_score=icp.getFitnessScore(max_correspondence_distance);
observation_noise = 0.0;
observation_noise(1,1) = icp_score_scale*icp_fitness_score;
observation_noise(2,2) = icp_score_scale*icp_fitness_score;
observation_noise(3,3) = icp_score_scale*icp_fitness_score;

//aggiornamento del measurement model
measurement_pdf->AdditiveNoiseSigmaSet(observation_noise);

//update filtro
filter->Update(measurement_model.get(),observation_mean);
```

Vengono determinati, a partire dalla matrice di rototraslazione della trasformazione rigida, la matrice di rotazione per determinare l'angolo (*sappiamo che il robot può variare il suo angolo solamente in una direzione: angolo di yaw*), e il vettore di traslazione. Queste due grandezze consentono di determinare il vettore delle osservazioni, la covarianza della osservazione viene costruita come una matrice diagonale (*misurazioni indipendenti*) in cui i valori sono dati da `icp_fitness_score`, che rappresenta la *somma delle distanze al quadrato dai vari source al target*, infatti la varianza di variabili aleatorie può essere calcolata come [6]:

$$\sigma^2 = \sum \frac{(x_i - \hat{x})^2}{N-1}$$

Infatti una volta completato il processo di matching, per capire come queste misurazioni siano affette da rumore, si può usare questo indicatore. Infine viene aggiornato il filtro con la media e la covarianza della misurazione.

4.6. Gestione dei Sistemi di Rifermento

Il processo di localizzazione consente di determinare la posizione del robot nell'ambiente, ma alcune grandezze durante il procedimento dipendono proprio da questa grandezza, come la matrice di trasformazione tra il sistema di riferimento del laser e della mappa, necessario per importare la PointCloud in maniera corretta per la misurazione, per capire come trasformare la pointCloud del laser del sistema di riferimento della mappa, è necessario conoscere la posizione del robot, che dipende dal processo di localizzazione che si sta effettuando.

L'ambiente ROS mette a disposizione due elementi per gestire correttamente i cambiamenti tra i sistemi di riferimento:

- `Tf listener`
- `Tf broadcaster`

Il primo elemento consente di *mettersi in ascolto* e determinare la trasformazione tra due sistemi di riferimento, in funzione della condizione attuale, questo strumento viene utilizzato per determinare la trasformazione tra i sistemi di rifermento del laser e dell'odometria, necessaria per trasformare la pointcloud del laser nel sistema di rifermento mappa.

Il secondo consente di *trasmettere* una trasformazione tra due sistemi di rifermento, alla fine del processo di localizzazione, ad ogni iterazione, viene aggiornato il legame tra sistema di rifermento della mappa e sistema di riferimento della posizione del robot, in modo di essere in grado di reperire la trasformazione aggiornata alla prossima iterazione. In questo modo si è in grado di gestire le trasformazioni che cambiano nel tempo, come quelle legate alla posizione del robot nell'ambiente.

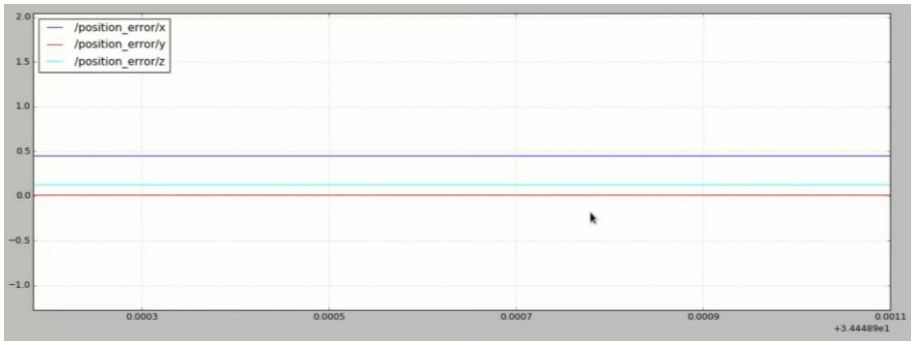
4.7. Risultato del Processo di Localizzazione

Il risultato del processo di localizzazione è la stima del vettore di stato $p = [x \quad y \quad \theta]^T$, nel caso di EKF attraverso una grandezza gaussiana. Per rappresentare questa grandezza nell'ambiente di visualizzazione RViz si è fatto uso di un marker di visualizzazione a forma di ellisse, centrato nel punto del valore medio, e con un'ampiezza proporzionale all'incertezza dello stato, ovvero proporzionale ai valori della matrice do covarianza.

```
visualization_msgs::Marker tempMarker;
tempMarker.pose.position.x = pose.pose.position.x;
tempMarker.pose.position.y = pose.pose.position.y;

Eigen::SelfAdjointEigenSolver<Eigen::Matrix2f> eig(covMatrix);

const Eigen::Vector2f& eigValues (eig.eigenvalues());
const Eigen::Matrix2f& eigVectors (eig.eigenvectors());

float angle = (atan2(eigVectors(1, 0), eigVectors(0, 0)));

tempMarker.type = visualization_msgs::Marker::SPHERE;//sfera schiacciata come un disco

double lengthMajor = sqrt(eigValues[0]);
double lengthMinor = sqrt(eigValues[1]);

tempMarker.scale.x = 3.0*lengthMajor;
tempMarker.scale.y = 3.0*lengthMinor;
tempMarker.scale.z = 0.001; //schiacciato

tempMarker.color.a = 1.0;
tempMarker.color.r = 0.0;
tempMarker.color.g = 1.0;
tempMarker.color.b = 0.0;

tempMarker.pose.orientation.w = cos(angle*0.5);
tempMarker.pose.orientation.z = sin(angle*0.5);

tempMarker.header.frame_id = map_frame;
tempMarker.id = 0;

draw_position.publish(tempMarker); //pubblica nel topic
```

A partire dalla matrice di covarianza vengono determinati autovalori e autovettori, in modo da determinare le diagonali maggiori dell'ellisse da rappresentare, oltre a posizione e angolo per centrare e orientare l'ellissoide; infine viene pubblicato il marker nel topic.

Oltre alla visualizzazione nell'ambiente RViz viene restituito un grafico, facendo uso del plugin `rqt_plot`, che restituisce il vettore di errore tra posizione reale (*determinata grazie al framework Gazebo, che simulando l'ambiente virtuale conosce la posizone esatta del robot*) e posizione predetta.



5. PROBLEMI RISCONTRATI, SOLUZIONI ADOTTATE E CONSIDERAZIONI FINALI

5.1. Utilizzo del filtro di Kalman classico

In questo lavoro, prima di utilizzare l'approccio descritto, si è provato ad utilizzare il filtro di Kalman classico con l'utilizzo dell'approssimazione dell'Extended Kalman Filter per il passaggio di prediction, e l'uso di un measurement model sempre lineare durante la fase di correction, in cui i parametri vengono calcolati secondo le formule:

• Calcolo *prior*	$x_{t\|t-1} = f(x_{t-1\|t-1}, u_t)$ $P_{t\|t-1} = F_x P_{t-1\|t-1} F_x^T$
• Calcolo innovazione	$\tilde{y} = z_t - C x_{t\|t-1}$ $S_t = R_t + C\, P_{t\|t-1} C^T$
• Calcolo guadagno del filtro	$K_t = P_{t\|t-1} C^T S_t^{-1}$
• Calcolo *posterior*	$x_{t\|t} = x_{t\|t-1} + K_t\, \tilde{y}_t$ $P_{t\|t} = P_{t\|t-1} - K_t C P_{t\|t-1}$

I risultati del processo di localizzazione non erano soddisfacenti, in quanto il valore della media era sensibilmente distante dal valore di posizionamento ottenuto mediante l'odometria di Gazebo. Il valore del guadagno del filtro di Kalman assumeva valori molto alti, che portavano il valore dell'innovazione a modificare sensibilmente la predizione della posizione del robot, causando una forte imprecisione nell'operazione di localizzazione. In alcune condizioni il valore della matrice del guadagno del filtro assumeva valori nulli, portando il sistema a non aggiornare la media del valore predetto durante la fase di correction.

L'utilizzo di una struttura come quella della libreria Bayesian Filtering Library ha consentito di semplificare le operazioni di aggiornamento e gestione del filtro, designando tutte le operazioni a un modello più robusto.

5.2. Caso particolare: ambiente diverso dalla mappa nota del sistema

Per verificare la robustezza del sistema, e la capacità di ICP a determinare le corrispondenze e gestire il processo di misurazione anche con una mappa inesatta rispetto all'ambiente, è stato effettuato un test in cui venivano eliminati alcuni elementi della mappa mantenendo la stessa rappresentazione della stessa, ovvero senza ripetere il processo di generazione della mappa.

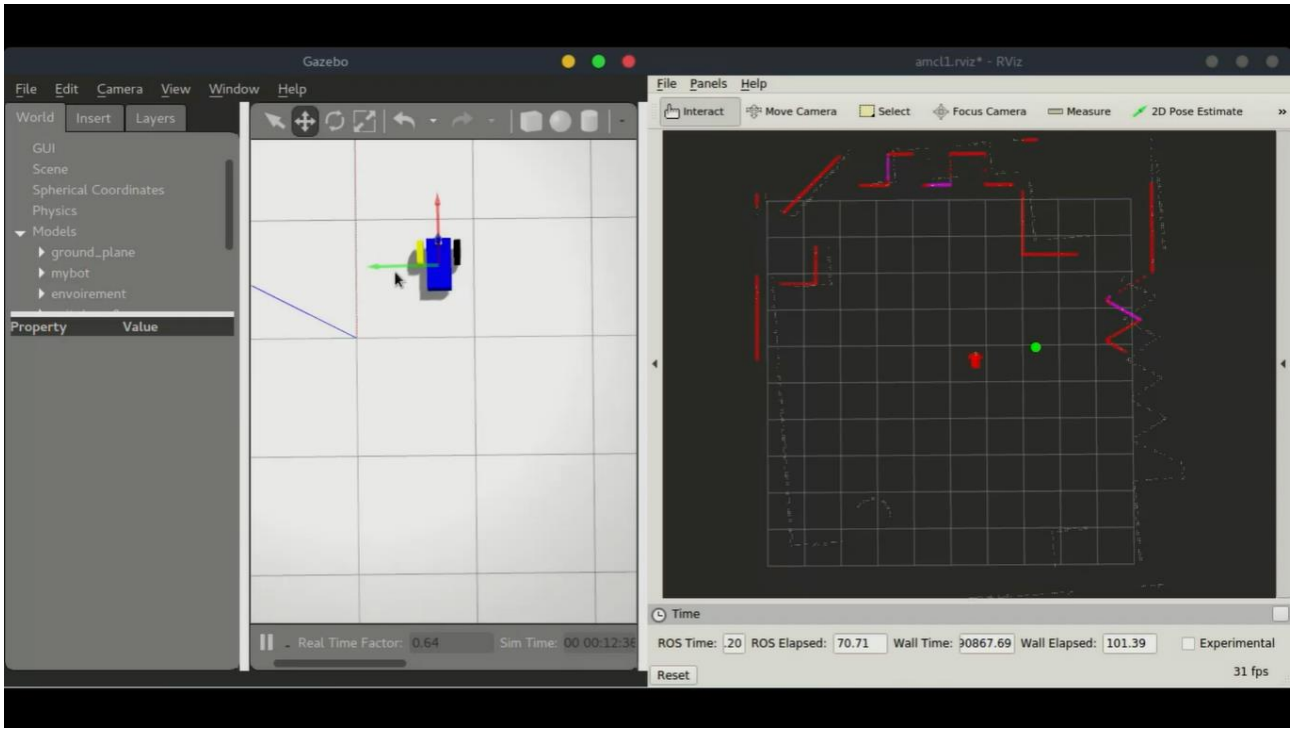
In questo modo il sistema ha una mappa non esatta rispetto all'ambiente. Il metodo ICP ha un meccanismo di consensus, per determinare gli outlier nel determinare la trasformazione tra due nuvole di punti, questa operazione viene effettuata attraverso un algoritmo basato su RANSAC, come descritto precedentemente, il quale richiede tra i diversi parametri la soglia di outliers tollerabile nel sistema, dopo alcune prove sperimentali questo valore è stato impostato a 0.1 (10%).

Questo significa che se l'ambiente non viene completamente stravolto rispetto alla rappresentazione della mappa, il sistema è comunque in grado di determinare una trasformazione tra le due nuvole di

punti, nel caso in cui la rappresentazione della mappa è completamente inesatta rispetto all'ambiente l'algoritmo non converge a una soluzione.

5.3. Problema: caso di disallineamento nei sistemi di riferimento

Uno dei limiti importanti del sistema implementato in questa maniera è il fatto che funziona correttamente se le condizioni iniziali nel sistema di localizzazione sono corrette, deve verificarsi che il robot parta dall'origine del sistema di riferimento della mappa. L'origine di questo sistema di riferimento è il punto da cui si inizia a raccogliere dati durante il processo di creazione della mappa, il filtro di kalman durante la creazione viene inizializzato con i valori di $[x \quad y \quad \theta] = [0 \quad 0 \quad 0]$. Il filtro inoltre non deve perdere l'informazione sul robot. Ad esempio se si "*disallinea manualmente*" il robot rispetto alla posizione predetta, non con un comando sulla velocità ma spostandolo nell'ambiente simulato di Gazebo (corrisponderebbe all'operazione di prendere il robot differenziale e spostarlo fisicamente da un punto a un altro senza "*informare*" il filtro) si verifica il fatto che l'operazione di localizzazione non funziona correttamente, questo anche perché la rappresentazione della point cloud del laser dipende dalla posizione del robot, se questa non è esatta non è corretta nemmeno la rappresentazione dell'ambiente per il passaggio di correction, come mostrato nell'immagine in esempio.



In Figura viene mostrato il caso in cui il robot viene "" rispetto alla posizione nota nel filtro, il processo di localizzazione fallisce completamente. Se il robot inizia a muoversi dopo qualche iterazione il sistema si riallinea, grazie ai passaggi di prediction e correction, facendo uso dell'informazione dell'odometria del movimento e grazie al passaggio di correction che con il sensore laser, in questo modo esso è in grado dopo alcune iterazioni dell'algoritmo di determinare nuovamente la sua posizione rispetto all'ambiente che ha intorno.

5.4. Eventuale Implementazione in ambiente reale

ROS è un meta-sistema che consente lo sviluppo di tecnologie attraverso diverse piattaforme, questo è un vantaggio in fase di sviluppo in quanto consente di poter sviluppare un sistema su una piattaforma virtuale e simulata, e utilizzare i risultati ottenuti anche per una implementazione reale. In quanto caso il package che è stato sviluppato per il processo di localizzazione è stato implementato partendo da un ambiente simulato, esso potrebbe essere comunque utilizzato anche in un ambiente reale con alcuni accorgimenti.

Si supponga di avere a disposizione un robot differenziale controllabile attraverso la piattaforma ROS (esempio single board con ambiente Linux-like embedded), dato che il lavoro svolto è stato sviluppato in un ambiente simulato, per portarlo in un ambiente reale sarebbero necessari:

- Una descrizione accurata del robot, e della sua odometria
- Descrizione accurata del laser e del suo modello di misurazione

Il limite più importante per una implementazione reale sarebbe la struttura del sensore e il measurement model, in questo lavoro si è fatto uso di un ambiente estremamente controllato, in cui il sensore laser, a parte rumore di misurazione additiva con struttura gaussiana a media nulla e varianza, non presenta altre forme di non idealità. Nel caso di un sensore laser reale sarebbe necessaria un'analisi più precisa del measurement model, e del suo funzionamento in un contesto non simulato, tenendo in considerazione gli effetti dell'ambiente sulle misurazioni, e le caratteristiche stesse del sensore.

5.5. Possibili sviluppi

Esempi di possibili sviluppi di questo lavoro potrebbero essere:

1. Adozione di un measurement model meno ideale (*non lineare*), che rispetta le caratteristiche del sensore.
2. Se non si hanno a disposizione dei sensori laser l'utilizzo di descrittori ORB (Oriented FAST and Rotated BRIEF) potrebbe essere una alternativa, si tratta di descrittori puntuali ottenuti delle immagini. Il risultato di questi descrittori potrebbe essere rappresentato sempre come una nuvola di punti, sarebbe sempre necessario determinare la mappa del sistema a partire dallo stesso strumento utilizzato per la misurazione, il che garantirebbe una struttura sviluppabile mediante lo stesso algoritmo con ICP.
   Un sistema basato su visione sarebbe più influenzabile rispetto alle condizioni ambientali, anche in questo caso sarebbe necessario un modello di misurazione preciso, inoltre per determinare la posizione dei punti nella point cloud, se non si ha a disposizione una camera stereo bisognerebbe adottare soluzioni di *structure from motion*.

BIBLIOGRAFIA

[1] Thrun, Sebastian. Fox "Probabilistic robotics" Massachusetts Institute of Technology – 2006

[2] https://people.mech.kuleuven.be/~tdelaet/bfl_doc/getting_started_guide/node18.html

[3] http://pointclouds.org/documentation/tutorials/iterative_closest_point.php

[4] https://www.coursera.org/lecture/robotics-learning/iterative-closest-point-1jDix

[5] http://docs.ros.org/jade/api/bfl/html/classBFL_1_1ExtendedKalmanFilter.html

[6] Barczyk, Martin, et al. "Invariant EKF Design for Scan Matching-Aided Localization." IEEE Trans. Contr. Sys. Techn. 23.6 (2015): 2440-2448.