Università degli Studi di Bergamo

SCUOLA DI INGEGNERIA

Corso di Laurea Magistrale in Ingegneria Informatica

Progetto di Robotica

Orocos KDL

Prof.
Davide Brugali

Candidati
Davide Salvetti
Matricola 1057596

Gianpaolo Dalini
Matricola 1045938

Matteo Verzeroli
Matricola 1057926

Anno Accademico 2022–2023

Introduzione

L'obiettivo del progetto realizzato è quello di integrare la libreria Orocos KDL (`https://www.orocos.org/kdl.html`) nel component framework *STAR* per lo sviluppo di un'applicazione che consente di inviare dei comandi di movimento ai motori di un robot manipolatore affinché l'end-effector raggiunga una particolare posizione finale, espressa nel sistema di riferimento della base del robot, seguendo un percorso lineare. Il robot manipolatore utilizzato è il modello P-Rob 3, prodotto dalla F&P Personal Robotics (Fig.1.1). In particolare, sono state sviluppate le seguenti funzionalità:

- calcolo della posizione della pinza espressa nel sistema di riferimento della base del robot a partire dal valore degli angoli dei giunti (cinematica diretta);
- calcolo del valore dei giunti a partire dalla posizione della pinza espressa nel sistema di riferimento della base del robot (cinematica inversa);
- calcolo di un percorso lineare a partire da una posizione iniziale e una finale della pinza espresse nel sistema di riferimento della base del robot.

Figura 1.1: Robot manipolatore P-Rob 3.

Libreria Orocos KDL

2.1 Installazione

Per effettuare i calcoli cinematici si è scelto di utilizzare la libreria Orocos KDL che implementa alcuni algoritmi, come il calcolo della cinematica diretta e inversa, per un robot opportunamente descritto da un file di configurazione. Seguendo la guida di installazione disponibile nella documentazione (`https://www.orocos.org/wiki/Installation_Manual.html`), la libreria è stata installata nel framework *STAR* al seguente percorso:

`STAR/AutonomousRobot/Libraries/others/orocos_kinematics_dynamics.`

Inoltre, per semplificare alcune operazioni di inizializzazione degli oggetti utilizzati dalla libreria Orocos KDL, è stata installata anche la libreria *KDL Parser* (`https://github.com/ros/kdl_parser`) nella cartella

`STAR/AutonomousRobot/Libraries/others/kdl_parser`.

Infine, è stato modificato il file *env-star.sh* inserendo le informazioni per includere i file e recuperare i binari delle librerie installate necessari alla compilazione dei componenti e plugin sviluppati.

2.2 Definizione del modello del robot

La libreria Orocos KDL ha bisogno di conoscere la struttura cinematica del robot. La catena cinematica del robot è stata definita in un file .urdf (`http://wiki.ros.org/urdf`), nel quale sono descritti in modo standard tutti i parametri, quali ad esempio il numero, la tipologia e i limiti dei giunti e le caratteristiche dei link. Per definire questo file, si è partiti dal modello fornito sulla pagina github del produttore (`https://github.com/fp-robotics/fp_descriptions`), generando il file urdf seguendo la procedura indicata in tale pagina. Successivamente, il file è stato modificato per estrapolare le informazioni rilevanti per il progetto e, eseguendo delle prove con l'interfaccia di controllo web del robot, sono stati verificati alcuni dati presenti nell'urdf. La catena cinematica descritta dall'urdf è riportata in figura 2.1. Nel file urdf, che è strutturato come un file xml, è possibile specificare i giunti e i link del robot tramite i seguenti elementi:

- <link name="xxx">...</link>, elemento che descrive le caratteristiche di un link;
- <joint name="yyy" type="revolute">...</joint>, elemento che descrive le caratteristiche di un giunto; il robot utilizzato è costituito da sei giunti di rotazione, per cui il tipo dei giunti è definito come "revolute".

All'interno degli elementi è possibile poi specificare diverse caratteristiche tra le quali (elenco completo reperibile al link `http://wiki.ros.org/urdf/XML/joint`):

- <parent link="xxx"/> e <child link="yyy"/> indicano rispettivamente il link padre e il link figlio di un giunto;
- <axis xyz="0 1 0"/> indica l'asse di rotazione del giunto;

- <origin xyz="0 0 0.29"/> indica la rototraslazione tra il link padre e il link figlio (in radianti).
- <limit effort="1.09" lower="-2.94960643587" upper="2.9670597283903604" velocity="0.8290313946973065"/> descrive i limiti di velocità, angolo massimo e minimo ed effort di un giunto;

Per descrivere il robot utilizzato sono quindi stati definiti sei giunti di rotazione (tre con asse di rotazione lungo l'asse z e tre con asse di rotazione lungo l'asse y) e sette link dalla base del robot fino al centro di presa. Sono stati poi aggiunti un giunto di tipo *fisso* e un link terminale per rappresentare correttamente il centro di presa.

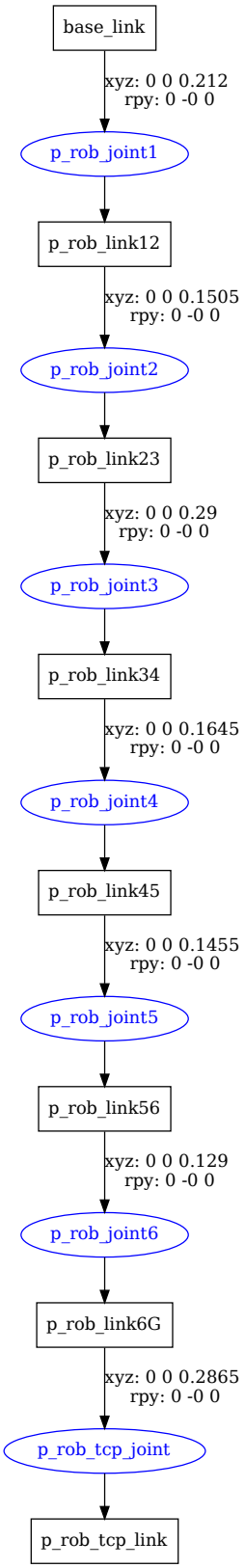

Figura 2.1: Catena cinematica descritta dal file urdf.

2.3 KDL Paser

A partire dalla descrizione del robot contenuta nel file urdf, è necessario costruire una struttura dati di tipo `KDL::Chain`, utilizzata dalla libreria Orocos KDL per i calcoli cinematici. Per fare ciò, si è utilizzato la libreria KDL Parser. Inizialmente, viene eseguito il parsing del file urdf tramite la funzione `urdf::parseURDFFile(path_urdf)` che restituisce un oggetto di tipo `urdf::ModelInterfaceSharedPtr` che rappresenta il modello del file urdf. Se questa operazione viene effettuata correttamente, l'oggetto appena creato viene utilizzato per generare un oggetto di tipo `KDL::Tree` tramite la funzione `kdl_parser::treeFromUrdfModel` che rappresenta la struttura del robot. Se l'oggetto viene creato correttamente, si utilizza il metodo `my_tree.getChain("base_link", "last_link", m_chain)` per definire un oggetto di tipo `KDL::Chain` che rappresenta la catena cinematica del robot dal link "base_link", il cui nome viene specificato come primo parametro della funzione, fino al link finale "last_link", specificato come secondo parametro e che corrisponde al centro di presa. Un oggetto di tipo `KDL::Chain` è costituito da `KDL::Segment`, dove un segmento rappresenta un link con un giunto e con due collegamenti alle estremità con altri segmenti. Successivamente, dall'oggetto `KDL::Chain` vengono estratti i limiti dei giunti (in termini di angoli minimi e massimi raggiungibili dai giunti). Essi vengono salvati in un array di tipo `KDL::JntArray` e saranno utilizzati nel calcolo della cinematica inversa.

2.4 Sistemi di riferimento del robot

Nelle figure 2.2 e 2.3 sono riportati i sistemi di riferimento rispetto alla base del robot e del tool che sono stati considerati nel progetto. Inoltre le convenzioni scelte per indicare le rotazioni prevedono che gli angoli di roll (α), pitch (β) e yaw (γ) siano definiti rispetto alla terna fissa (x,y,z) della base del robot, eseguendo le rotazioni nel seguente ordine: rotazione attorno all'asse x di angolo α, rotazione attorno all'asse y di angolo β e rotazione attorno all'asse z di angolo γ.

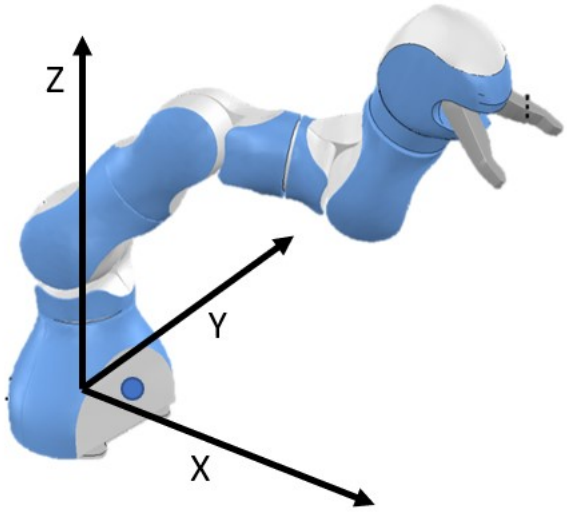

Figura 2.2: Sistema di riferimento alla base del robot.

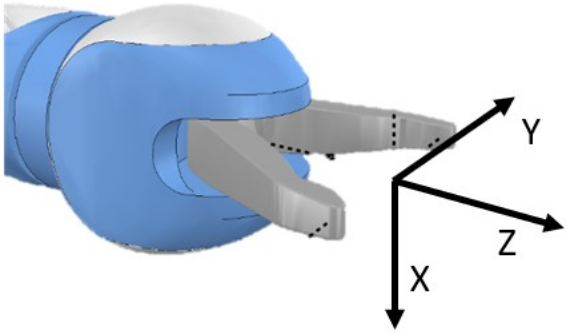

Figura 2.3: Sistema di riferimento del tool.

2.5 Cinematica diretta

La cinematica diretta permette di calcolare la posizione e l'orientamento dell'end-effector a partire dal valore dei giunti. La libreria Orocos KDL mette a disposizione il metodo

`int KDL::ChainFkSolverPos_recursive::JntToCart(const JntArray& q_in, Frame& p_out)`

che riceve in ingresso un vettore *q_in* di tipo `KDL::JntArray` che contiene la configurazione dei giunti e restituisce la posizione dell'end-effector rispetto al sistema del riferimento della base tramite l'oggetto *p_out* di tipo `KDL::Frame`, che rappresenta una rototraslazione nello spazio 3D. Il valore intero restituito indica l'esito del calcolo della cinematica diretta (zero se il calcolo è stato effettuato con successo). Questo metodo appartiene alla classe `KDL::ChainFkSolverPos_recursive` che implementa un algoritmo ricorsivo per il calcolo della cinematica diretta. Il costruttore di questa classe

`ChainFkSolverPos_recursive (const Chain &chain)`

richiede come parametro un oggetto `KDL::Chain`.

2.6 Cinematica inversa

La cinematica inversa permette di calcolare la configurazione dei giunti a partire dalla posizione dell'end-effector espressa nel sistema di riferimento della base. La libreria Orocos KDL mette a disposizione diversi risolutori che implementano differenti algoritmi per il calcolo della cinematica inversa tramite metodi iterativi. In questo progetto è stato utilizzato il risolutore `KDL::ChainIkSolverPos_NR_JL` che implementa un metodo generale per la risoluzione della cinematica inversa utilizzando il metodo di Newton-Raphson tenendo in considerazione i limiti dei giunti. Il costruttore del risolutore ha la seguente segnatura:

`KDL::ChainIkSolverPos_NR_JL::ChainIkSolverPos_NR_JL(const Chain & chain, const JntArray & q_min, const JntArray & q_max, ChainFkSolverPos & fksolver, ChainIkSolverVel & iksolver, unsigned int maxiter = 100, double eps = 1e-6)`,

dove *chain* è un oggetto di tipo `KDL::Chain` che corrisponde alla descrizione della catena cinematica del robot, *q_min* e *q_max* contengono i limiti dei giunti, *fksolver* rappresenta un riferimento a un risolutore di cinematica diretta, *iksolver* un riferimento a un risolutore di cinematica inversa differenziale (nel progetto è stato utilizzato un risolutore di tipo `KDL::ChainIkSolverVel_pinv`), *maxiter* rappresenta il numero massimo di iterazioni e *eps* rappresenta il massimo errore espresso in metri della posizione raggiunta dall'end-effector con i valori degli angoli ottenuti rispetto alla posizione richiesta. Nel progetto sono stati scelti un numero di iterazione massimo pari a 1000 e una precisione di 1 mm. La classe `KDL::ChainIkSolverPos_NR_JL` fornisce il metodo

`int KDL::ChainIkSolverPos_NR_JL::CartToJnt(const JntArray & q_init, const Frame & p_in, JntArray & q_out)`

che, data la posizione dell'end-effector tramite il parametro *p_in*, restituisce i valori dei giunti calcolati nella variabile *q_out*. Il parametro *q_init* rappresenta la configurazione iniziale dei giunti dalla quale il metodo iterativo inizia la ricerca della soluzione della cinematica inversa e può influenzare la configurazione finale dei giunti. Il valore intero restituito indica l'esito dell'operazione (zero se il calcolo è andato a buon fine).

2.7 Traiettoria lineare

La classe `KDL::Path_Line`, che estende `KDL::Path`, permette di creare un oggetto che rappresenta un percorso lineare da una posizione iniziale ad una finale. Di seguito si riporta il costruttore di un oggetto di tipo `KDL::Path_Line`:

`Path_Line (const Frame &F_base_start, const Frame &F_base_end, RotationalInterpolation *orient, double eqradius, bool _aggregate=true)`.

I parametri `F_base_start` e `F_base_end` rappresentano i sistemi di riferimento della posizione iniziale e

finale del percorso lineare che si vuole calcolare: in questo progetto coincidono con la posizione iniziale e finale dell'end-effector. Il parametro *orient* di tipo `KDL::RotationalInterpolation` è utilizzato dalla classe per rappresentare la rotazione associata a un percorso geometrico. La documentazione della libreria suggerisce di inizializzare questo parametro attraverso il metodo

`KDL::RotationalInterpolation_SingleAxis()`

che ha il vantaggio di essere indipendente dal sistema di riferimento nel quale si esprime il percorso. Il parametro *eqradious* è di tipo `double` e rappresenta l'*equivalent radious*. Questo parametro serve a comparare le rotazioni con le traslazioni, definendo la "quantità di movimento" di una rotazione come lo spazio percorso da un punto a distanza *eqradious* dall'asse di rotazione. Nel progetto, a questo parametro è stato assegnato il valore arbitrario 1. Tuttavia, sviluppi futuri richiederanno di approfondire il valore ottimo da assegnare a questo parametro. Per il progetto sono stati utilizzati anche i seguenti metodi della classe `KDL::Path_Line`:

- `double KDL::Path_Line::PathLength()`: restituisce la lunghezza del percorso. Tuttavia, la lunghezza restituita non è da considerarsi sempre come una lunghezza fisica, ad esempio quando la componente di rotazione è dominante;
- `Frame KDL::Path_Line::Pos(double s)`: ritorna il sistema di riferimento lungo il percorso alla lunghezza s attuale. Questo metodo permette di ricavare i *waypoints* all'interno del percorso lineare.

Implementazione in STAR

In figura 3.1 è riportato il diagramma dei componenti inseriti nel framework STAR.

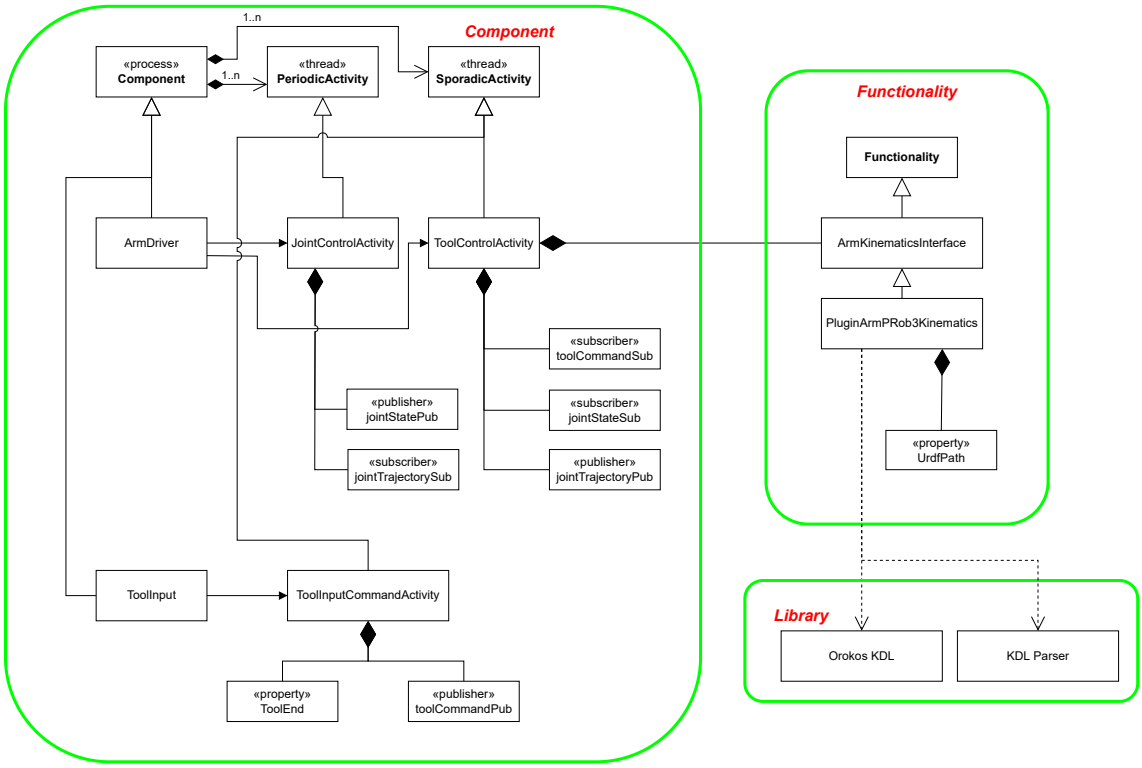

Figura 3.1: Diagramma dei componenti STAR.

Sono stati sviluppati due *Component*:

- *ArmDriver*: implementa l'attività *JointControlAcivity* che si occupa di comunicare con il robot inviandogli i comandi di movimento e ricevendo la posizione corrente dei giunti, e la *ToolControlActivity* che si occupa di elaborare la traiettoria lineare che l'end-effector dovrà seguire.
- *ToolInput*: implementa l'attività *ToolInputCommandActivity* che si occupa di inviare la posizione target che il tool deve raggiungere a partire dalla posizione corrente.

L'attività *ToolControlActivity* utilizza le funzionalità della classe *PluginArmPRob3Kinematics*, la quale fornisce le funzioni per eseguire la cinematica diretta, la cinematica inversa ed il calcolo di un percorso lineare utilizzando la libreria Orocos KDL.

3.1 Plugin: PluginArmPRob3Kinematics

Il plugin *PluginArmPRob3Kinematics* implementa l'interfaccia *ArmKinematicsInterface* e fornisce le funzionalità per il calcolo della cinematica diretta, della cinematica inversa e per la computazione di un percorso lineare. Per fare ciò utilizza le funzioni della libreria Orocos KDL descritte nella sezione precedente. In figura 3.2 è riportato il diagramma della classe in questione.

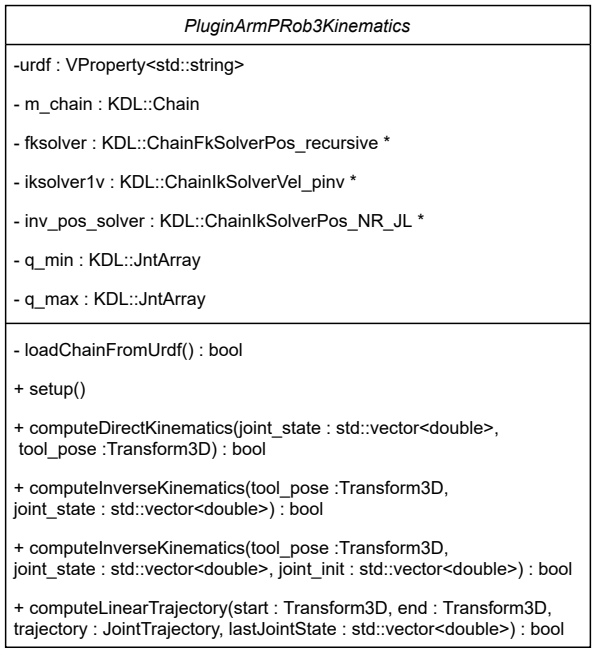

Figura 3.2: Diagramma UML della classe `PluginArmPRob3Kinematics`.

VProperty<std::string>urdf Campo che rappresenta il nome del file urdf da utilizzare. Il file deve essere presente al seguente path:
`AutonomousRobots/Functionalities/plugins/arm_prob3_kinematics/cfg`.

KDL::Chain m_chain Oggetto che incapsula la catena cinematica del robot.

KDL::JntArray q_min, q_max Rappresentano i limiti inferiori e superiori dei valori degli angoli dei giunti del robot. Vengono estratti dall'urdf quando viene chiamata la funzione `loadChainFromUrdf()`.

bool loadChainFromUrdf() Funzione che si occupa di estrarre la `KDL::Chain` ed i limiti dei giunti a partire dall'urdf del robot attraverso le funzioni fornite dalla libreria KDL Parser descritte nella sezione precedente (cap.2.3).
Return:

- `true`: se è stato estratto un oggetto `KDL::Chain` dall'urdf;
- `false`: altrimenti.

void setup() Metodo in cui vengono inizializzati tutti gli oggetti necessari al funzionamento del plugin. In particolare, vengono inizializzate le classi della libreria Orocos KDL necessarie al calcolo della cinematica diretta ed inversa e viene letto l'urdf tramite il metodo `loadChainFromUrdf()`.

bool computeDirectKinematics(std::vector<double>&joint_state, Transform3D &toolPose) Metodo che si occupa di eseguire il calcolo della cinematica diretta utilizzando la funzione `KDL::ChainFkSolverPos_recursive::JntToCart(const JntArray &q_in, Frame &q_out)` (cap.2.5).
Parametri:

- `joint_state`: input, valore dei giunti da cui si vuole calcolare la posizione dell'end-effector;
- `toolPose`: output, oggetto che rappresenta la posizione dell'end-effector rispetto alla base del robot.

Return:

- `true`: la cinematica diretta è stata calcolata senza errori;
- `false`: altrimenti.

bool computeInverseKinematics(Transform3D &tool_pose, std::vector<double>&joint_state) Metodo che si occupa di eseguire il calcolo della cinematica inversa utilizzando la funzione `ChainIkSolverPos_NR_JL::CartToJnt(const JntArray &q_init, const Frame &p_in, JntArray &q_out)` (cap.2.6).
In questo caso, il valore iniziale dei giunti da cui si parte a calcolare la soluzione della cinematica inversa (parametro *q_init*) è pari a 0, che corrisponde alla posizione del robot P-Rob 3 completamente disteso con la pinza rivolta verso l'alto.
Parametri:

- `toolPose`: input, oggetto che rappresenta la posizione dell'end-effector rispetto alla base del robot di cui si vogliono conoscere i rispettivi valori dei giunti;
- `joint_state`: output, valore dei giunti associati alla posizione dell'end-effector rappresentata da `toolPose`.

Return:

- `true`: la cinematica inversa è stata calcolata senza errori;
- `false`: altrimenti.

bool computeInverseKinematics(Transform3D &tool_pose, std::vector<double>&joint_state, std::vector<double>&joint_init) Metodo che si occupa di eseguire il calcolo della cinematica inversa utilizzando la funzione `ChainIkSolverPos_NR_JL::CartToJnt(const JntArray &q_init, const Frame &p_in, JntArray &q_out)` (cap.2.6). A differenza del metodo precedente, è possibile specificare la posizione iniziale dei giunti del robot da cui si parte a calcolare la cinematica inversa.
Parametri:

- `toolPose`: input, oggetto che rappresenta la posizione dell'end-effector rispetto alla base del robot di cui si vogliono conoscere i valori dei giunti;
- `joint_state`: output, valore dei giunti associati alla posizione dell'end-effector.
- `joint_init`: input, valore iniziale dei giunti da cui partire per il calcolo della cinematica inversa con metodi incrementali.

Return:

- `true`: la cinematica inversa è stata calcolata senza errori;
- `false`: altrimenti.

bool computeLinearTrajectory(Transform3D &start, Transform3D &end, JointTrajectory &trajectory, std::vector<double>&lastJointState) Metodo tramite cui è possibile computare, a partire da un punto iniziale ed un punto finale nello spazio, un percorso lineare che l'end-effector dovrà seguire (cap.2.5). In figura 3.3 è mostrato il funzionamento dell'algoritmo che computa il percorso lineare. Si osservi che, nel caso in cui non la cinematica inversa non venga calcolata correttamente per un punto lungo il percorso, l'algoritmo riparte dall'inizio perturbando in modo incrementale il valore iniziale dei giunti nel calcolo del primo *waypoint* della traiettoria.
Parametri:

- `start`: input, oggetto che rappresenta la posizione iniziale dell'end-effector;
- `end`: input, oggetto che rappresenta la posizione finale che l'end-effector dovrà raggiungere seguendo un percorso lineare;
- `trajectory`: output, oggetto che contiene un array di valori dei giunti che rappresentano i punti di passaggio lungo la retta che il robot deve seguire;
- `lastJointState`: input, vettore contenente i valori dei giunti della posizione corrente del robot.

Return:

- `true`: un percorso lineare è stata trovato senza errori;
- `false`: altrimenti.

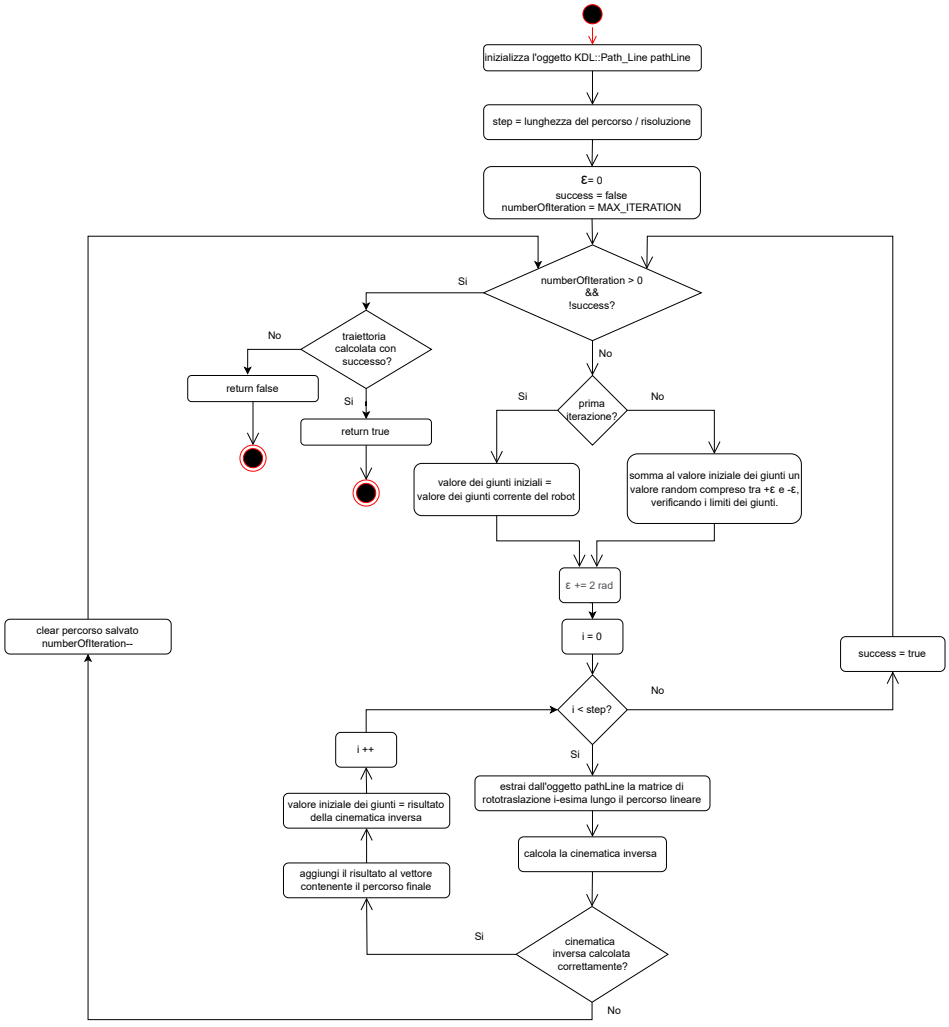

Figura 3.3: Diagramma di flusso che mostra il funzionamento dell'algoritmo che calcola il percorso lineare.

3.2 Componenti: ArmDriver e ToolInput

La comunicazione con il robot è gestita dal componente *ArmDriver* che implementa due attività: la *JointControlActivity* e la *ToolControlActivity*. Queste due attività vengono inizializzate e registrate nel metodo `ArmDriver::startup()` della classe `ArmDriver`.

L'attività *JointControlActivity* è una *periodic activity*, che riceve una traiettoria dal topic *joint_trajectory* e pubblica lo stato corrente dei giunti sul topic *joint_state*. Inoltre, si interfaccia con la *functionality* `ArmInterface` che si occupa della comunicazione con il nodo ROS integrato nel robot. Quando viene pubblicata una traiettoria, viene chiamata la funzione di callback `JointControlActivity::jointTrajectoryCallback(VariantActivity* va)`, che si occupa di estrarre ogni punto di passaggio del percorso ricevuto e inviarlo al robot tramite l'interfaccia `ArmInterface`.

L'attività *ToolControlActivity* è una *sporadic activity* che riceve la posizione target dell'end-effector sul topic *tool_command* e lo stato attuale dei giunti sul topic *joint_state* mentre pubblica la traiettoria calcolata sul topic *joint_trajectory*. Inoltre, si interfaccia con la *functionality* `KinematicsInterface` che permette di richiamare le funzioni presenti nel plugin `PluginArmPRob3Kinematics`. Ogni volta che l'attività riceve lo stato dei giunti dalla *JointControlActivity* li memorizza nella variabile *std::vector<double>lastJointState*. Invece, quando riceve dal componente *ToolInput* la posizione target dell'end-effector viene richiamata la funzione di callback `ToolControlActivity::toolCommandCallback (VariantActivity *va)` che si occupa di pubblicare la traiettoria. In particolare, inizialmente viene calcolata la posizione attuale dell'end-effector grazie alla cinematica diretta calcolata a partire dall'ultimo stato dei giunti ricevuto. Se il calcolo va a buon fine e il comando ricevuto richiede il calcolo di una traiettoria lineare, si procede al calcolo del percorso lineare dalla posizione attuale calcolata e la posizione target ricevuta (tramite la funzione `computeLinearTrajectory`). Se viene calcolato in modo corretto un percorso lineare, si procede alla pubblicazione della traiettoria sul topic *joint_trajectory*, che sarà poi ricevuta ed eseguita dalla *JointControlActivity*.

Il componente *ToolInput* si occupa invece di indicare all'attività *ToolControlActivity* la posizione target che l'end-effector deve raggiungere e implementa l'attività *ToolInputCommandActivity*.

L'attività *ToolInputCommandActivity* è una *sporadic activity* che si occupa di leggere la proprietà `ToolEnd` definita nel file di configurazione del componente. Questa proprietà definisce la posizione target dell'end-effector tramite il seguente formato:

```
1  <ToolEnd>
2    Value>x y z roll pitch yaw</Value>
3  </ToolEnd>,
```

dove *x, y, z* sono espressi in metri mentre *roll, pitch, yaw* sono espressi in radianti. Una volta letta, la posizione sarà inserita in un `star::manipulation_msgs::ToolMotionCommand` e sarà pubblicata sul topic *tool_command*. Essa verrà ricevuta dall'attività *ToolControlActivity* che procederà al calcolo della traiettoria.