

**POLITECNICO DI BARI**

**DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELL’INFORMAZIONE (DEI)**

**Corso di Laurea Magistrale in Ingegneria dell’Automazione - Robotics**

Progetto in

**MOBILE AND FIELD ROBOTICS**

**IMPLEMENTAZIONE SU AMBIENTE ROS DI UN ROBOT MOBILE COME ATOMIZZATORE PER IRRIGAZIONE DI UN VIGNETO**

**ROS IMPLEMENTATION OF A WHEELED MOBILE ATOMIZER ROBOT FOR VINEYARD IRRIGATION**

|  |  |
| --- | --- |
| Studenti:  **Carmelo CARAMUSCIO**  **Michele DAMONE** | Prof.s:  **Prof. Ing. Luca De Cicco**  **Ing. Carlo Croce** |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_­­­\_\_\_\_\_\_\_\_\_\_\_\_

**A.A. 2020-2021**

Sommario

[**INTRODUZIONE** 3](#_Toc88216486)

[**MODELLI** 4](#_Toc88216487)

[**Robot** 4](#_Toc88216488)

[Husky A200 4](#_Toc88216489)

[Sensoristica 5](#_Toc88216490)

[**World File** 6](#_Toc88216491)

[**LAUNCH FILE** 10](#_Toc88216492)

[husky\_atom\_gazebo.launch 10](#_Toc88216493)

[husky\_atom\_rviz.launch 12](#_Toc88216494)

[**NAVIGATION** 13](#_Toc88216495)

[Gmapping 14](#_Toc88216496)

[Amcl e Move Base 16](#_Toc88216497)

[**ASSEGNAZIONE DI PUNTI DI PASSAGGIO** 19](#_Toc88216498)

[**CONCLUSIONI** 21](#_Toc88216499)

# **INTRODUZIONE**

L’obiettivo del progetto è di automatizzare l’irrorazione dei vigneti per mezzo di un robot mobile in maniera da ridurre il tempo impiegato da operatori agricoli in questa attività e anche l’esposizione prolungata degli stessi a sostanze chimiche potenzialmente nocive.

È stata effettuata una simulazione in ambiente ROS utilizzando il robot mobile Husky A200 generando un modello 3D di un piccolo vigneto con l’introduzione anche di ostacoli.

# **MODELLI**

# **Robot**

## Husky A200

Il robot mobile scelto per questo progetto è il modello Husky A200 sia per l’ampio supporto in ambiente ROS, sia per la compatibilità con le operazioni richieste. Il robot è un robot di medie dimensioni con ottime capacità di carico e una discreta sensoristica di supporto che implementa la guida differenziale.

Il sistema di guida differenziale risulta essere ottimale per gli ambienti di lavori previsti data la necessità di potersi districare in ambienti circoscritti con pochi margini di manovra. Il robot Husky A200, infatti, è prodotto e ottimizzato proprio per ambienti esterni e accidentati.

Ai fini del progetto il robot è stato dotato di una cisterna e di due sensori che sono montati su un’asta prevista in dotazione nei modelli originali forniti da Clearpath Robotics.

Sulla piattaforma metallica posizionata sul robot è stata prevista la cisterna adibita all’irrorazione delle vigne.

Immagine che contiene erba, esterni

Descrizione generata automaticamente

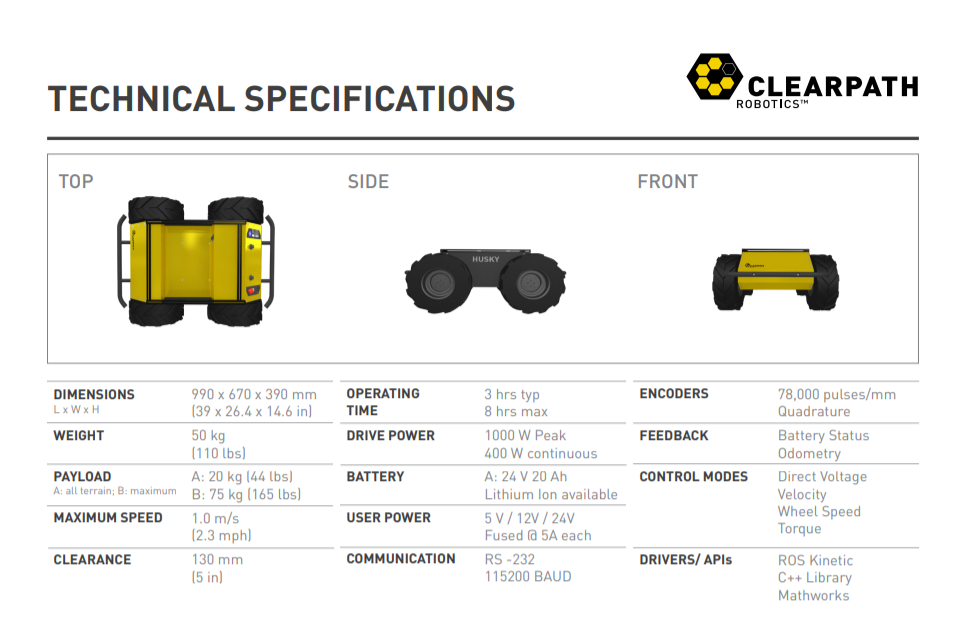


Figura 1 Husky A200 Technical Specifications

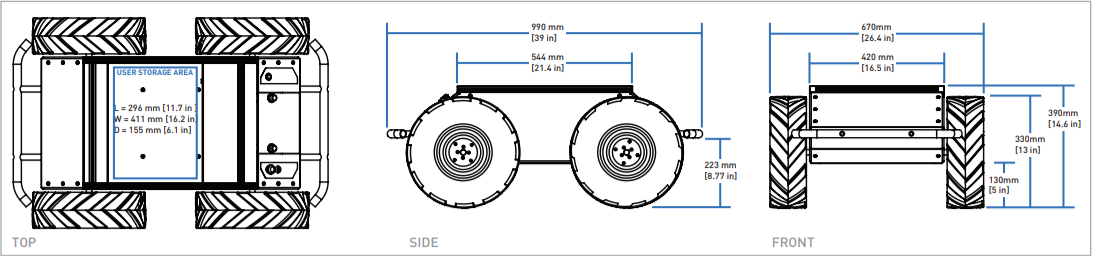


Figura 2 Dimensions of Husky A200

## Sensoristica

Il robot prevede come equipaggiamento sensoristico LIDAR, GPS e videcamere. Nello specifico sono stati utilizzati il LIDAR e la videocamera entrambi montate su un supporto a forma di arco che poggia sulla piattaforma del robot.

* Sensori 2D LiDAR LMS111

Fa parte della famiglia di sensori LMS1xx ed è caratterizzato da un angolo di apertura di 270°, campo di lavoro che va da 0,5-20 metri e frequenza di scansione 25/50Hz

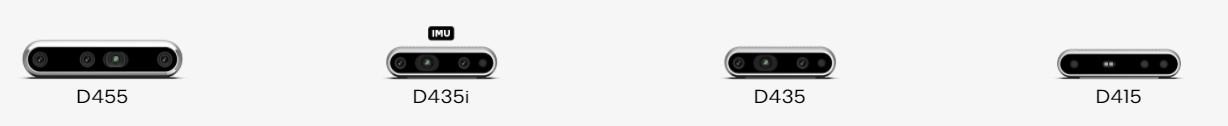
Immagine che contiene interni, mescolatrice, fornello

Descrizione generata automaticamente

Figura 3 LiDAR LMS111

* CAMERA Intel RealSense Dxxx

La videocamera montata sul robot è una Intel RealSense ed è stata prevista per obiettivi futuri di riconoscimento di immagini per l’analisi del vigneto.



# **World File**

L’ambiente è caratterizzato, come anticipato, da uno scenario reale dove è presente un piccolo vigneto con un muretto a secco a delimitare l’area di lavoro.

Il terreno è stato realizzando applicando una texture al piano di appoggio.

Per implementare il vigneto è stato utilizzato un modello di un in intero filare da quattro vigne.



L’ambiente di lavoro è circoscritto da quattro muretti a secco scalati affinchè potessero essere rilevati dal LiDAR posizionato in cima al robot.

Immagine che contiene catena, linea

Descrizione generata automaticamente

Per la realizzazione del vigneto sono state posizionate i filari delle vigne distanziati di 2,5m utilizzando il metodo population per la replica di più modelli in maniera ordinata.

Immagine che contiene testo

Descrizione generata automaticamente

Infine, è stato aggiunto il modello di una persona tra i filari per simulare la presenza di ostacoli nel vigneto.

Per assolvere a questa funzione si è utilizzato il modello “standing\_person” già presente nella libreria di modelli fornita da Gazebo



La descrizione completa dell’ambiente in cui è destinato ad operare il robot è contenuta all’interno di un file .world così composto che include tutti i modelli sopra descritti.

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

E si presenta come segue

Immagine che contiene testo

Descrizione generata automaticamente

# **LAUNCH FILE**

## husky\_atom\_gazebo.launch

Il luanch file di seguito descritto, come suggerisce il nome, è stato implementato per definire i parametri necessari a definire il modello del robot e del mondo all’interno di Gazebo.

Sono stati definiti a tal proposito i parametri di simulazione e visualizzazione del sistema e il modello del robot, assegnando al parametro *robot\_descirption* il file URDF che ne descrive nel dettaglio la struttura e il payload.

Infine, è stato definito il *joint\_state\_publisher.*

Immagine che contiene testo

Descrizione generata automaticamente

Il modello husky, sviluppato e pubblicato da *clearpathrobotics,* prevede un file nel quale vengono, inoltre, definiti i nodi attraverso cui è possibile spawnare il robot all’interno di gazebo nel pieno delle sue funzionalità e mettere in atto il controllo dello stesso.

L’organizzazione descrittiva è organizzata secondo una logica che prevede una serie di launch file richiamati a catena fino al file oggetto di questo paragrafo.

All’interno di husky\_atom\_gazebo.launch, infatti, viene richiamato il file *spawn\_husky.launch*



Che oltre a definire la posizione con cui è possibile visualizzare il robot nel mondo, predispone la possibilità di controllarlo attraverso una ricostruzione di un joystick, da noi non utilizzata in questo progetto.

All’interno di *spawn\_husky.launch* viene poi richiamato un ulteriore launch file, il *control.launch,* attraverso cui l’autore definisce i parametri del joint\_publisher e del velocity controller per mezzo del file *control.yaml*.

Immagine che contiene testo

Descrizione generata automaticamente



Inoltre, vengono richiamati i nodi necessari al controllo del movimento e la localizzazione.

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

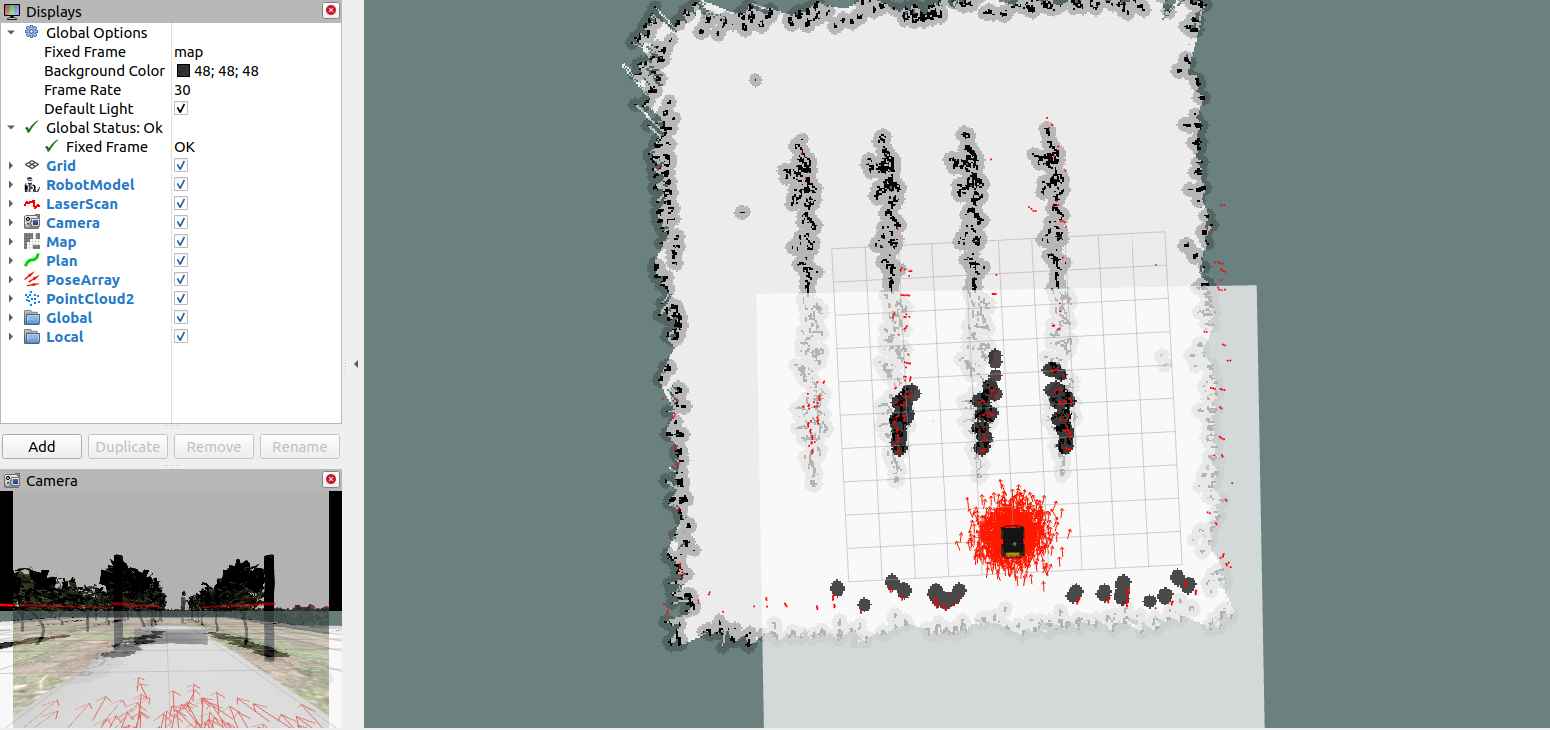
## husky\_atom\_rviz.launch

Il secondo launch file sviluppato prevede la visualizzazione del modello del robot all’interno di rviz. Strumento, questo, indispensabile per lo sviluppo delle dinamiche di autonomuos drivig e obstacle avoidance, oggetto di questo lavoro di progetto.

Immagine che contiene testo, screenshot, schermo, argento

Descrizione generata automaticamente

Attraverso rviz, infatti, è stato possibile implementare, testare e definire completamente tutti gli aspetti del navigation stack, utilizzati per il raggiungimento dei goals iniziali del presente lavoro.



Attraverso rviz, infatti, è stato possibile monitorare tutti i topic attivi in fase di simulazione e di seguito riportati

LaserScan /scan

Camera /realsense/color/image\_raw o /realsense/depth/image\_rect\_raw

Map /map

Plan /move\_base/NavfnROS/plan

PoseArray /particlecloud

Global:

Global\_Plan /move\_base/TebLocalPlannerROS/global\_plan

Global Map /move\_base/global\_costmap/costmap

Local:

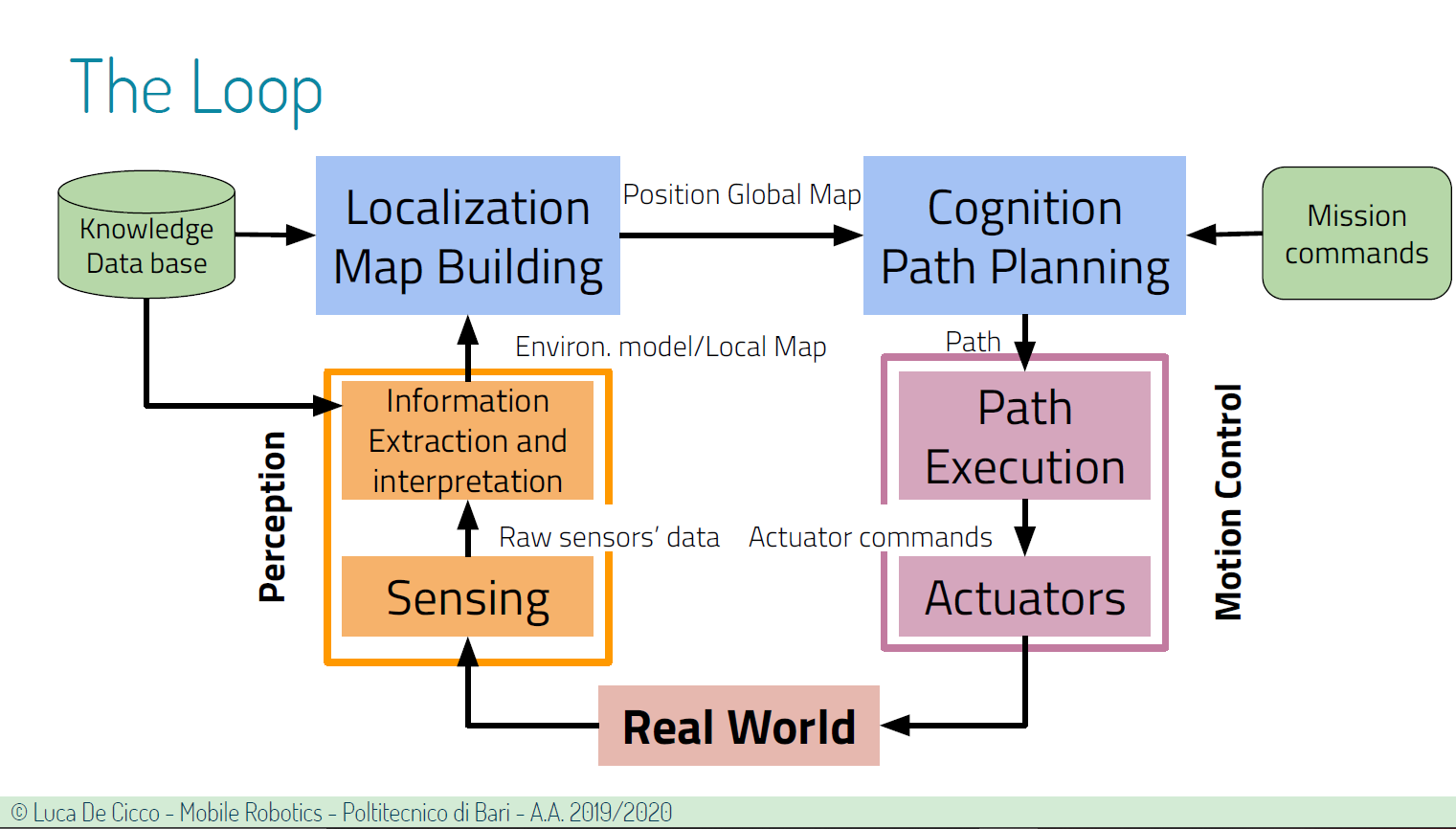
Local\_Plan /move\_base/TebLocalPlannerROS/local\_plan

Local Map /move\_base/local\_costmap/costmap

# **NAVIGATION**

L’obiettivo del progetto definito in fase preliminare consisteva nell’assegnare al robot un path predefinito, e questo venisse eseguito evitando gli ostacoli statici e dinamici presenti all’interno di una mappa fedele alla realtà applicativa a cui fa riferimento il presente lavoro.

Questo compito rientra pienamente negli obiettivi che il *Navigation Stack* si propone di assolvere come è possibile apprezzare attraverso la seguente immagine che descrive per mezzo un diagramma a blocchi l’operato di questo set di nodi e algoritmi atti al compimento di una missione di autonomous moving.



Il *Navigation Stack* si sviluppa attraverso tre step fondamentali ovvero:

**Map generation** per mezzo dei cosiddetti metodi SLAM (simultaneous localization and

mapping), tra *cui Cartographer, Hector, Karto, Frontier Exploration* e *Gmapping* da noi utilizzato.

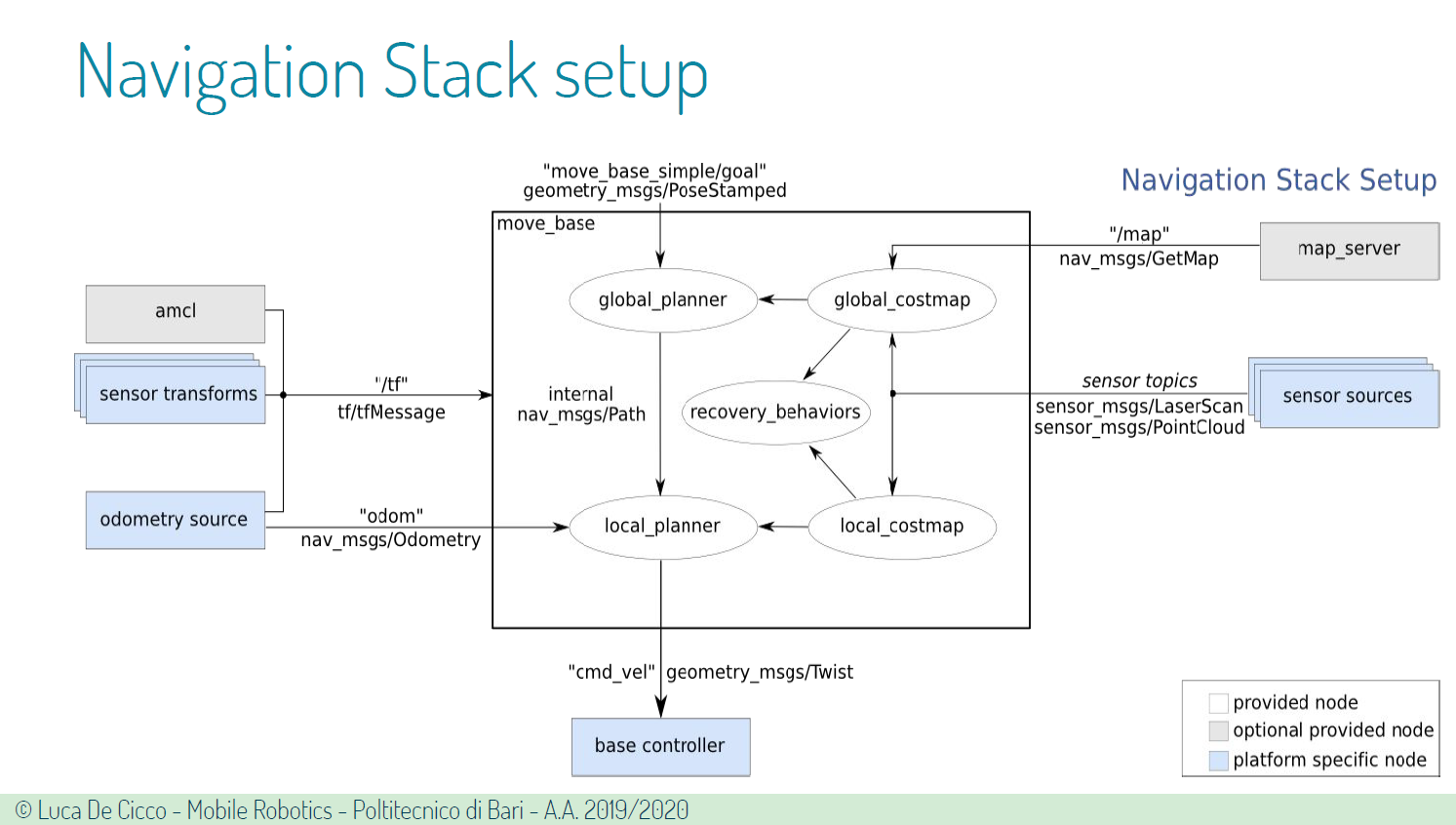
**Localization** attraverso cui si individua la posizione del robot nel mondo. Il metodo utilizzato per questa funzione sfrutta la teoria delle probabilità per definire la posa del robot ed è definito AMCL (Adaptive Monte Carlo Localization).

Le informazioni sull’ambiente utilizzate dall’algoritmo in ROS sono fornite dalla sensoristica di bordo del modello, nello specifico dai sensori laser. Questa informazione è resa disponibile per mezzo dei messaggi *sensor\_msgs/LaserScan*.

L’ultimo step è il vero e proprio **Path Planning**, costituito da una fase di **Global Plannig** ove viene definito una path generale sulla base della morfologia del mondo in cui il robot si trova a operare e della presenza di ostacoli statici.

A questa fase segue quella di **Local Planning** in cui viene ricalcolato real-time un path sulla base delle informazioni dei sensori relativamente agli ostacoli dinamici da evitare.

Di seguito è possibile apprezzare il setup del *Navigation Stack*



## Gmapping

La mappa è stata, quindi, creata servendosi di due nodi fondamentali. Il primo è proprio il gmapping, messo a disposizione da ROS, il secondo è il teleop\_twist\_keyboard necessario per l’esplorazione dell’ambiente circostante.

Concretamente quindi, la mappa è state generata eseguendo il file *husky\_atom\_gmapping.launch* che richiama questi due nodi.

Immagine che contiene testo

Descrizione generata automaticamente



In questo modo è stato possibile, movendo il robot su tutto l’ambiente virtuale, acquisire dati su quest’ultimo per mezzo del laserscan. La mappa risultante a processo compiuto e salvata con loppurtuno comando



appare come in figura

Immagine che contiene testo

Descrizione generata automaticamente

## Amcl e Move Base

Lo step di localizzazione è permesso dall’opportuno ROS pkg Amcl, che opportunamente parametrizzato e richiamato all’interno del file *husky\_atom\_amcl.launch* appare come segue

Immagine che contiene testo

Descrizione generata automaticamente

Gli step di localizzazione e quello di path planning vengono eseguiti run time, infatti questo file viene richiamato nel launch file *husky\_atom\_navigation.launch* assieme ad un altro file, responsabile della movimentazione autonoma del robot, il file *move\_base\_dynamic.launch.*

Il sistema husky prevede di default l’utilizzo del pkg move\_base in cui il base\_global\_planner è costituito dall’algoritmo navfn/NavfnROS mentre il base\_local\_planner si poggia sull’algoritmo dwa\_local\_planner/DWAPlannerROS.

Per motivi di natura strutturale del mondo e delle dimensioni del robot nello stesso, in fase progettuale è risultato di più facile parametrizzazione un algoritmo differente per il base\_local\_planner. Questo è il teb\_local\_planner/TebLocalPlannerROS. Il file *husky\_atom\_navigation.launch* si conclude con il nodo *map\_server* al quale viene assegnato il file .map generato attraverso il gmapping.

Immagine che contiene testo, monitor, schermo, televisione

Descrizione generata automaticamente

Nell’immagine seguente è possbile apprezzare nel dettaglio il file *move\_base\_dynamic.launch*

Immagine che contiene testo

Descrizione generata automaticamente

I parametri del nodo move\_base, sono stati opportunamente tarati sulla base delle esigenze prestazionali del robot all’interno dell’applicazione oggetto del presente lavoro.

In prima istanza viene caricato il file *planner\_teb.yaml,* questo è un file di configurazione necessario per gestire il comportamento del robot di fronte agli ostacoli e non solo. Nell’immagine seguente è possbile apprezzarne la struttura

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

In prima istanza si è definita l’impronta del robot con l’istruzione *footprint\_model*, settata come un rettangolo con stessi vertici della base del modello.

Successivamente, a seguito di numerose simulazioni, si è notato come, in presenza di ostacoli dinamici, il robot non riuscisse a superarli se questi erano posizionati nel mezzo della vigna. Questo problema era da attribuirsi al parametro *min\_obstacle\_dist,* il quale risultando troppo grande impediva al robot ti poter considerare percorribile un passaggio tra l’ostacolo e la stessa vigna nonostante fosse fisicamente possibile.

Considerando che per il planner teb le distanze fisiche sono considerate in metri, settare questo parametro al valore di 0.1 ha permesso di risolvere il problema.

La configurazione delle costmap è da ricondursi al file *costmap\_common.yaml* così definito

Immagine che contiene testo

Descrizione generata automaticamente

La local costmapè descritta, oltre che dal file appena illustrato, da un altro file, il *costmap\_local.yaml,* così definito e configurabile in dimensione

Immagine che contiene testo

Descrizione generata automaticamente

Mentre per la global costmap, questa è definita dal file *costmap\_global\_static.yaml*

Immagine che contiene testo

Descrizione generata automaticamente

# **ASSEGNAZIONE DI PUNTI DI PASSAGGIO**

Per la creazione di un path prestabilito da assegnare al modello, si è optato per l’assegnazione di una serie di punti di passaggio, opportunamente scelti sulla base delle esigenze di progettazione.

È stato individuato un nodo predisposto ad assolvere a questa funzione.

Si tratta del *move\_base\_seq*, uno script che permette l’assegnazione iterativa di punti di passaggio come messaggi *MoveBaseAction e* *MoveBaseGoal* di *move\_base\_msgs.msg.*

Nello specifico il nodo provvede a salvare la sequenza di pose assegnata come parametri ROS nel ROS Parameter Server, questi poi vengono inseriti come goal dal nodo python in modo da essere correttamente interpretati ed eseguiti dall'Action Server.

Di seguito si riporta il launch file *movebase\_seq.launch* e il relativo nodo python

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

# **CONCLUSIONI**

A conclusione di questo lavoro di progetto è possibile affermare che il modello così definito assolve in pieno alle funzioni definite in fase preliminare.

Come possibili sviluppi futuri si potrebbe automatizzare l’intero processo, ad esempio si potrebbe comprendere la fase di rifornimento di acqua/pesticidi da parte del robot, oppure sarebbe possibile sviluppare un sistema di exploration nel quale il robot naviga su tutto il mondo generando alla fine una mappa autonomamente.