bug0_robo_driver_ROS

1.0

Generated by Doxygen 1.8.17

Chapter 1

Research Track 1 - Assignment 2 (UniGe)

This is a client package for the package assignment_2_2022 that provides an implementation of an action server, that moves a robot in the environment by implementing the bug0 algorithm. This is the 2nd asignment as part of the course: **Research Track 1** at the University of Genoa, Italy.

1.1 Goals and Overview:

Three (or four) principle nodes have been developed as part of this package:

- **Node 1a** (`target_info_client_node`)
  - Implements an action client, allowing the user to set a target position (x, y), or cancel it.
- **Node 1b** (`robot_status_pub_node`)
  - This node subscribes to `/odom` topic and publishes the retrieved position and velocity values using a custom ROS message (`RoboStatusMsg.msg`) to the topic `/robot/robo_stats`.
- **Node 2** (`goal_info_srv_node`)
  - This ROS service node serves the number of goals reached by the robot and cancelled by the user. To get the number of goals, this subscibes to the topic `/reaching_goal_result` and increments each value accordingly, depending on the returned status of the robot reaching goal or not.
- **Node 3** (`robo_info_node`)
  - This node subscribes to the topic `/robot/robo_stats` to obtain robot's position and velocity using our custom message and displays these parameters along with robot's distance from goal and average speed.

In addition to these, in order to show the implementation of a ROS Service-Client, a node: (`disp_goals_node`) has also been created, that calls the Service Server node `goal_info_srv_node` and prints the number of goals reached and cancelled.

1.2 Requirements

The overall project has been well tested using ROS Noetic and requires RViz and Gazebo for simulation and physics.

1.3 Directory Structure [Package: unige_rt1_assignment2]

```
.
|-- CMakeLists.txt
|-- LICENSE
|-- README.md
|-- config
|   `-- robot.yaml
|-- images
|   `-- rosgraph.png
|-- launch
|   `-- complete_sim.launch
|-- msg
|   `-- RoboStatusMsg.msg
|-- package.xml
|-- src
|   |-- disp_goals.cpp
|   |-- goal_info_srv.cpp
|   |-- robo_info.cpp
|   |-- robot_status_pub.cpp
|   `-- target_info_client.cpp
`-- srv
    `-- GoalInfo.srv
```

1.4 Running the full simulation:

1.4.0.1 Update and upgrade packages:

```
sudo apt update && sudo apt upgrade
```

1.4.0.2 First, we need to build a catkin workspace for the project. Navigate to preferred directory, then:

```
mkdir -p bug0robo_catkin_workspace/src/
cd bug0robo_catkin_workspace/src/
```

1.4.0.3 Initialize the catkin workspace:

```
catkin_init_workspace
```

1.4.0.4 Now clone this repo and the action server repo (assuming user has ssh-auth setup):

```
git clone git@github.com:amanarora9848/unige_rt1_assignment2.git
git clone git@github.com:CarmineD8/assignment_2_2022.git
```

1.4.0.5 Navigate back to base directory <tt>bug0robo_catkin_workspace</tt>

```
cd ..
```

1.4.0.6 Run <tt>catkin_make</tt> to build the package:

```
catkin_make
```

1.4.0.7 Source <tt>setup.bash</tt> to set the environment variables used by ROS

```
source devel/setup.bash
```

1.4.0.8 Launch the complete simulation. The launch file for this can be found in <tt>/src/unige_rt1_assignment2/launch/complete_sim.launch</tt> relative to the current diretory:

```
roslaunch unige_rt1_assignment2 complete_sim.launch
```

Using this command launches the simulation (particularly the node-C at the default rate of 5 /sec). This can be changed by also passing an argument parameter, for example:

```
***Here is an example of the simulation running:
[Implementation Video](https://youtu.be/TDBa1oudBaU)***
## Project Explanation and Implementation
### ROS rqt-graph (with nodes and topics) for the whole project (client-server)
<img src="images/rosgraph.png" alt="rqt-graph">
<!-- [rqt-graph](images/rosgraph.png) -->
### ROS Nodes in detail:
This package uses 4 + 1 nodes for the task to be done effectively and holistically. They are
       `target_info_client_node`, `robot_status_pub_node`, `goal_info_srv_node`, `robo_info_node` and 1
       extra node: `disp_goals_node`.
- As seen from the rqt-graph, the action client node `target_info_client_node` interacts with the action
       server defined in the package `assignment_2_2022` using the `Planning.action` file, in which are
       defined the action goal and feedback messages.
    - The pseudocode for this node is given below:
```

```
while ros is running:
    take user input to proceed or cancel goal

    if input is proceed:
        if cancel command passed before (is in stack):
            pop cancel command from stack
        end
        take user input for x and y
        send goal to action server
        set goal in parameter server variables
    else if input is cancel:
        if cancel not passed before: // stack is empty
            push cancel command to stack
            if action server status is executing goal:
                send cancel to action server
            else:
                prohibit cancel command
            end
        else if cancel in stack:
            prohibit cancel command
        end
    else:
        tell user that input is invalid
    end

```
```

- The `robot_status_pub_node` subscribes to the `/odom` topic for position and velocity of the robot and publishes them at the topic `/robot/robo_stats`.
  - The pseudocode for this node is given below: ``` subscribe to /odom topic check for /odom callbacks set x, y, vel_x, vel_y in callbacks while ros is running: set user x, y, vel_x, vel_y obtained from /odom publish these as custom message to /robot/robo_stats topic at user defined rate

  ```

  - Given below is the custom message definition. ```

## 1.5 A custom message for robot position and velocity

float32 pos_x float32 pos_y float32 vel_x float32 vel_y ```

- The node `robo_info_node` recieves these robot position and velocity values by subscribing to the `/robot/robo_stats`, along with obtaining the user-given goal position from the parameter server. The parameter server variables have been configured in the configuration file `robot.yaml` present in the ***config*** folder. ```yaml robot: { goal_pos_param: { x_goal: 0, y_goal: 0 } } ```
  - Using these values, the distance to the goal and theaverage velocities are calculated.
- The node `goal_info_srv_node` also interacts with the action server messages and receives the status message (`assignment_2_2022::PlanningActionResult::ConstPtr& msg`) from it, to determine the number of goals reached or cancelled.
  - The pseudocode for this node: ``` Declare service goal_service

  Subscribe to /reaching_goal/result topic from action server Check for /reaching_goal/result callbacks

  if goal is completed: goals_reached_number = goals_reached_number + 1 else if goal is cancelled: goals↩
  _cancelled_number = goals_cancelled_number + 1 end

  Define service response message Advertise /goals_service service ```
  - Given below is the service (srv) file definition.

### 1.5.1 ```

string msg_feedback ```

- The extra node `disp_goals_node` simple sends requests to the service server node `goal_info_↩ srv_node` at the specified rate defined in the launch file (which can be set by the user, the same for the node `robo_info_node`) to get the number of goals reached or cancelled and prints them on the terminal. The simple class definition for the node is given below: ```cpp class GoalInfoClient { private: // Define the service client ros::ServiceClient goal_info_client; // Define the service message unige_rt1_assignment2::↩
  GoalInfo goal_info_srv;

  public: GoalInfoClient(ros::NodeHandle *n, double freq) { // Set the frequency of the loop ros::Rate *loop_↩
  rate; loop_rate = new ros::Rate(freq);

  // Create the service client goal_info_client = n->serviceClient<unige_rt1_assignment2::GoalInfo>("/goals↩
  _service");

  while(ros::ok()) { // Call the service if (goal_info_client.call(goal_info_srv)) { ROS_INFO_STREAM(goal_↩
  info_srv.response.msg_feedback); } else { ROS_ERROR("Failed to call service /goals_service"); } // check for incoming messages ros::spinOnce(); // sleep for the time remaining loop_rate->sleep(); }

  delete loop_rate; } }; ```

### 1.5.2 Programming style

This package uses C++ classes for implementation of the ROS Nodes, since the use of classes not only provides structure and clarity to the code, but also aids in proper management of data and methods. We also avoid using unnecessary global variables in the code. The programs simply become more modular, with reusable chunks which are beneficial in larger projects.

### 1.5.3 Possible Improvements

- The client console can be made even more robust, error-free and interactive, since some of the erroneous user-inputs have not been handled.
- An interactive GUI on the console can be created.
- Currently, there is no exit button / command, and the easiest way to exit simulation is using `Ctrl-C` on the main console, where `roslaunch` command is used. Better mechanism can be developed.

### 1.5.4 Helpful references / documentation / answers:

- http://wiki.ros.org/actionlib/DetailedDescription
- http://docs.ros.org/en/api/actionlib_msgs/html/msg/GoalStatusArray.↩
html

# Chapter 2

# Class Index

## 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# Chapter 3

# File Index

## 3.1 File List

Here is a list of all documented files with brief descriptions:

# Chapter 4

# Class Documentation

## 4.1 GoalInfoClient Class Reference

Classd defining the service client.

### Public Member Functions

- GoalInfoClient (ros::NodeHandle ∗n, double freq)

### 4.1.1 Detailed Description

Classd defining the service client.

This class defines the service client that sends a request to the service server /goals_service displays the number of goals reached or cancelled by the user.

### 4.1.2 Constructor & Destructor Documentation

#### 4.1.2.1 GoalInfoClient()

```
GoalInfoClient::GoalInfoClient (
            ros::NodeHandle * n,
            double freq) [inline]
```

Constructor of the class

This constructor initializes the service client and the service message definition.

Parameters

n	Pointer to the node handle
freq	Frequency of the loop

Returns

None

This constructor initializes the service client and the service message definition.

$<$ variable to set the frequency of the loop

$<$ Initialize the loop rate variable

Initialize the service client

This statement initializes the service client to send request to the service /goals_service server.

The documentation for this class was generated from the following file:

- src/disp_goals.cpp

4.2 GoalsService Class Reference

Goals service class.

Public Member Functions

- GoalsService (ros::NodeHandle ∗n)
- void target_result_callback (const assignment_2_2022::PlanningActionResult::ConstPtr &msg)
- bool goals_service_callback (unige_rt1_assignment2::GoalInfo::Request &req, unige_rt1_assignment2::↩ GoalInfo::Response &res)

4.2.1 Detailed Description

Goals service class.

This class is used to create a node that subscribes to the topic /reaching_goal/result and obtains the result of the action done by the user to send a goal. It also provides a service /goals_service to provide the number of goals reached and cancelled.

Member functions:

- GoalsService(ros::NodeHandle ∗n)
  - This is the constructor of the class.
- target_result_callback(const assignment_2_2022::PlanningActionResult::ConstPtr& msg)
  - This is the callback function for the subscriber.
- goals_service_callback(unige_rt1_assignment2::GoalInfo::Request &req, unige_rt1_assignment2::GoalInfo::Response &res)
  - This is the callback function for the service advertiser.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 GoalsService()

```
GoalsService::GoalsService (
             ros::NodeHandle ∗ n )  [inline]
```

Constructor for the class.

Parameters

n	Pointer to the node handle.

Returns

None

This constructor creates a subscriber for the topic /reaching_goal/result and a service server for the service /goals↩
_service.

target_result_subscriber

This variable is a subscriber for the topic /reaching_goal/result for the result of the action.

goals_service_server

This variable is a service server for the service /goals_service.

4.2.3 Member Function Documentation

4.2.3.1 goals_service_callback()

```
bool GoalsService::goals_service_callback (
            unige_rt1_assignment2::GoalInfo::Request & req,
            unige_rt1_assignment2::GoalInfo::Response & res ) [inline]
```

Callback function for the service advertiser.

Parameters

req	The request message of type GoalInfo received on the service.
res	The response message of type GoalInfo to be sent on the service.

Returns

bool

This function gets called whenever a new service request arrives on the /goals_service topic. It sets the response message and returns true.

res.msg_feedback

This variable is the response message of type GoalInfo to be sent on the service.

4.2.3.2 target_result_callback()

```
void GoalsService::target_result_callback (
            const assignment_2_2022::PlanningActionResult::ConstPtr & msg )  [inline]
```

Callback function for the subscriber.

Parameters

msg	The message of type PlanningActionResult received on the topic.

Returns

None

This function gets called whenever a new message arrives on the /reaching_goal/result topic. It updates the variables goals_reached and goals_cancelled according to the status of the message.

The documentation for this class was generated from the following file:

- src/goal_info_srv.cpp

4.3 RoboClient Class Reference

Implements an action client, allowing the user to set a target position (x, y), or cancel it.

Public Member Functions

- RoboClient (ros::NodeHandle ∗n)

4.3.1 Detailed Description

Implements an action client, allowing the user to set a target position (x, y), or cancel it.

This class is used to create a client node that sends goals to the action server defined in package assignment_2←
_2022. The user can input a goal position after entering 'p' and pressing [ENTER] and the robot will move to that position. The user can also cancel the current goal by entering 'q'.

The class contains the following functions:

- RoboClient(ros::NodeHandle ∗n)
  - This is the constructor of the class.

The class contains the following variables:

- double x, y: The x and y position of the goal
- std::string goal_input: The user input for the goal command
- std::stack<char> goal_stack: The stack to store the user commands

The class contains the following actionlib client:

- actionlib::SimpleActionClient<assignment_2_2022::PlanningAction> target_ac
  - This is the actionlib client that sends goals to the action server.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 RoboClient()

```
RoboClient::RoboClient (
            ros::NodeHandle * n )  [inline]
```

Constructor for RoboClient class.

Parameters

n	The node handle object.

Returns

None

This constructor is used to create the action client, and send goals to the action server. It also lets the user set a goal position (x, y) or cancel the current goal, which is given to the action server for execution and simulation.

create the action client, true causes the client to spin its own thread

< Wait for the action server to start, will wait for infinite time

send the initial goal to the action server

< Declare goal object

Check if user input is p or q

Check if q was previously entered and remove it from the stack

< Get x and y position from user

Send goal to action server

< Set x position in goal object

< Set y position in goal object

Set the goal positions in the parameter server variables

< Set x position in parameter server

< Set y position in parameter server

< Error message if parameter not found

Ensure that q has not already been pressed by user

Check if there is a goal currently executing

Check for new messages from the subscribed topics

The documentation for this class was generated from the following file:

- src/target_info_client.cpp

4.4 RoboInfo Class Reference

Robot information class.

Public Member Functions

- RoboInfo (ros::NodeHandle ∗n, double freq)
- void robo_info_callback (const unige_rt1_assignment2::RoboStatusMsg::ConstPtr &msg)
- void show_info ()
- double distance_from_goal (double x, double y)
- void average_speed ()

4.4.1 Detailed Description

Robot information class.

This class is used to create a node that subscribes to the topic /robot/robo_stats and obtains the position and velocity of the robot. It also obtains the user given goal position from the parameter server.

Member functions:

- RoboInfo(ros::NodeHandle ∗n, double freq)
  - This is the constructor of the class.
- robo_info_callback(const unige_rt1_assignment2::RoboStatusMsg::ConstPtr& msg)
  - This is the callback function for the subscriber.
- distance_from_goal(double x, double y)
  - This function calculates the distance of the robot from the goal position.
- average_speed()
  - This function calculates the average speed of the robot in the x and y directions.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 RoboInfo()

```
RoboInfo::RoboInfo (
            ros::NodeHandle * n,
            double freq )  [inline]
```

Constructor of the RoboInfo class.

Parameters

n	Pointer to the node handle.
freq	Frequency of the node.

Returns

None

This function created a subscriber to the topic /robot/robo_stats to get the robot position and velocity. It also obtains the user given goal position from the parameter server. The Subscriber calls the callback function 'robo_info_↩ callback'.

4.4.3 Member Function Documentation

4.4.3.1 average_speed()

```
void RoboInfo::average_speed ( )  [inline]
```

Function to calculate average speed.

Returns

None

This function calculates the average speed of the robot in the x and y directions.

< Average speed in x direction

< Average speed in y direction

4.4.3.2 distance_from_goal()

```
double RoboInfo::distance_from_goal (
             double x,
             double y )  [inline]
```

Function to calculate distance from goal.

Parameters

x	Robot x position.
y	Robot y position.

Returns

distance Distance from goal.

This function calculates the distance of the robot from the goal position.

< Distance from goal

4.4.3.3 robo_info_callback()

```
void RoboInfo::robo_info_callback (
             const unige_rt1_assignment2::RoboStatusMsg::ConstPtr & msg )  [inline]
```

Callback function for the subscriber.

Parameters

msg	Pointer to the message.

Returns

None

This function gets the x and y pos, and linear x and y velocities from RoboSatusMsg message.

< Robot x position set from message

< Robot y position set from message

< Robot x velocity set from message

< Robot y velocity set from message

< Total x velocity

< Total y velocity

4.4.3.4 show_info()

```
void RoboInfo::show_info ( )  [inline]
```

Function to print robot information.

Returns

None

This function prints the robot position and velocity, goal position, distance from goal and average speed.

< Function call to calculate average speed

The documentation for this class was generated from the following file:

- src/robo_info.cpp

4.5 RobotStatusPublisher Class Reference

Publishes the position and velocity of the robot.

Public Member Functions

- RobotStatusPublisher (ros::NodeHandle ∗n, double freq)
- void odom_callback (const nav_msgs::Odometry::ConstPtr &msg)

4.5.1 Detailed Description

Publishes the position and velocity of the robot.

This class is used to publish the position and velocity of the robot on the topic /robot/robo_stats.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 RobotStatusPublisher()

```
RobotStatusPublisher::RobotStatusPublisher (
            ros::NodeHandle * n,
            double freq ) [inline]
```

Subscribe to /odom topic to read the position and velocity of the robot with the subscribing queue size of 1 and call the odom_callback function

Inform ROS master that we will be publishing a message of type geometry_msgs::Twist on the robot actuation topic with a publishing queue size of 10

Publish the robot position and velocity on the robot_pos_vel_publisher Publisher object at the rate of freq Hz

< Robot position x

< Robot position y

< Robot velocity x

< Robot velocity y

4.5.3 Member Function Documentation

4.5.3.1 odom_callback()

```
void RobotStatusPublisher::odom_callback (
            const nav_msgs::Odometry::ConstPtr & msg ) [inline]
```

Callback function for the /odom topic.

Parameters

msg	Message received on the /odom topic.

Returns

void

This function gets called whenever a new message arrives on the /odom topic. It reads the position and velocity of the robot from the message and updates the robot position and velocity.

$<$ Robot position x

$<$ Robot position y

$<$ Robot velocity x

$<$ Robot velocity y

The documentation for this class was generated from the following file:

- src/robot_status_pub.cpp

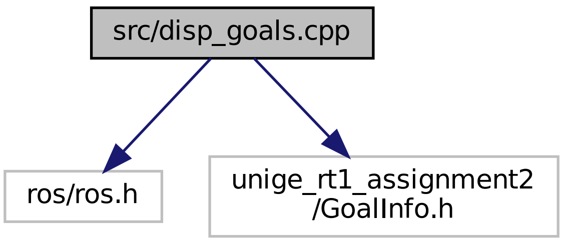
Chapter 5

File Documentation

5.1 src/disp_goals.cpp File Reference

Node to display number of goals reached or cancelled.

```
#include <ros/ros.h>
#include <unige_rt1_assignment2/GoalInfo.h>
```
Include dependency graph for disp_goals.cpp:



Classes

- class GoalInfoClient

  Classd defining the service client.

Functions

- int **main** (int argc, char ∗∗argv)

5.1.1 Detailed Description

Node to display number of goals reached or cancelled.

Author

Aman Arora (aman.arora9848@gmail.com)

Version

1.0

Date

15/03/2023

Services:
° /goals_service

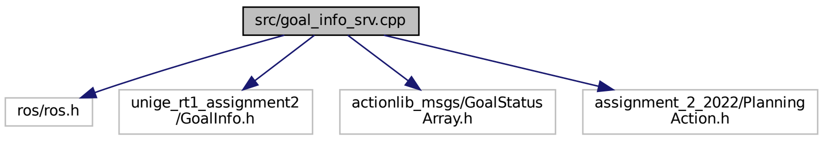
Description:

This node is used to create a service client node that sends a request to the service server /goals_service defined in this package, and, as response, displays the number of goals reached or number of goals cancelled by the user, by pressing 'q' on the user CL-interface.

5.2 src/goal_info_srv.cpp File Reference

Goal information service node.

```
#include <ros/ros.h>
#include <unige_rt1_assignment2/GoalInfo.h>
#include <actionlib_msgs/GoalStatusArray.h>
#include <assignment_2_2022/PlanningAction.h>
```
Include dependency graph for goal_info_srv.cpp:


src/goal_info_srv.cpp
ros/ros.h
unige_rt1_assignment2/GoalInfo.h
actionlib_msgs/GoalStatusArray.h
assignment_2_2022/PlanningAction.h

Classes

- class GoalsService

    Goals service class.

Functions

- int **main** (int argc, char ∗∗argv)

5.2.1 Detailed Description

Goal information service node.

Author

Aman Arora (aman.arora9848@gmail.com)

Version

1.0

Date

15/03/2023

Subscribes to:
° /reaching_goal/result

Services:
° /goals_service

Description:

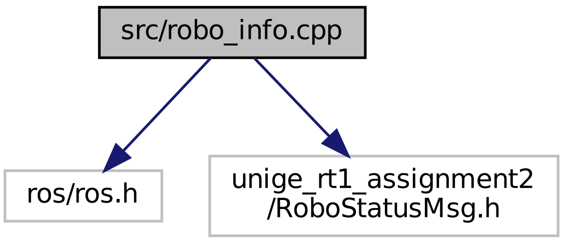
This node recieves the result of the action done by the user to send a goal and obtains this info from the topic /reaching_goal/result. It also provides a service /goals_service to provide the number of goals reached and cancelled.

5.3 src/robo_info.cpp File Reference

Robot information node.

```
#include <ros/ros.h>
#include <unige_rt1_assignment2/RoboStatusMsg.h>
```

Include dependency graph for robo_info.cpp:



Classes

- class RoboInfo

  Robot information class.

Functions

- int **main** (int argc, char **argv)

5.3.1 Detailed Description

Robot information node.

Author

Aman Arora (`aman.arora9848@gmail.com`)

Version

1.0

Date

15/03/2023

Parameters

`in`	*/robot/goal_pos_param/x_goal*	The x position of the goal
`in`	*/robot/goal_pos_param/y_goal*	The y position of the goal

Subscribes to:
° /robot/robo_stats

Description:

This node recieves the position and velocity of the robot from the topic /robot/robo_stats. It also obtains the user given goal position from the parameter server. The parameter server variables have been configured in the configuration file "robot.yaml".

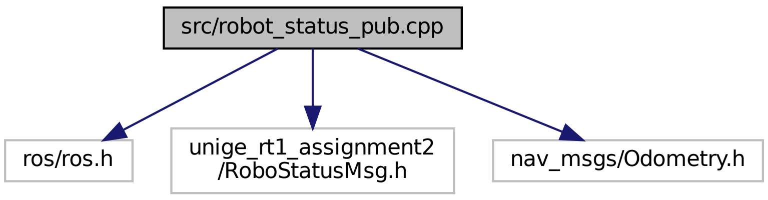
It thus calculates the average speed of the robot in the x and y directions using these values.

5.4 src/robot_status_pub.cpp File Reference

Status publisher node for the robot.

```
#include <ros/ros.h>
#include <unige_rt1_assignment2/RoboStatusMsg.h>
```

`#include <nav_msgs/Odometry.h>`
Include dependency graph for robot_status_pub.cpp:



Classes

- class RobotStatusPublisher

  Publishes the position and velocity of the robot.

Functions

- int **main** (int argc, char ∗∗argv)

5.4.1 Detailed Description

Status publisher node for the robot.

Author

Aman Arora (`aman.arora9848@gmail.com`)

Version

0.1

Date

2023-03-15

Subscribes to:
◦ /odom Publishes to:
◦ /robot/robo_stats

Description:

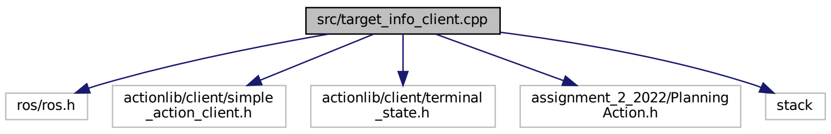
This node is used to publish the position and velocity of the robot on the topic /robot/robo_stats.

It uses the message RoboStatusMsg ROS message definition defined in this package.

5.5 src/target_info_client.cpp File Reference

Controller node for the robot to move to a target position. Implements an action client, allowing the user to set a target position (x, y), or cancel it.

```
#include <ros/ros.h>
#include <actionlib/client/simple_action_client.h>
#include <actionlib/client/terminal_state.h>
#include <assignment_2_2022/PlanningAction.h>
#include <stack>
```
Include dependency graph for target_info_client.cpp:


src/target_info_client.cpp
ros/ros.h
actionlib/client/simple_action_client.h
actionlib/client/terminal_state.h
assignment_2_2022/Planning Action.h
stack

Classes

- class RoboClient

  Implements an action client, allowing the user to set a target position (x, y), or cancel it.

Functions

- int **main** (int argc, char ∗∗argv)

5.5.1 Detailed Description

Controller node for the robot to move to a target position. Implements an action client, allowing the user to set a target position (x, y), or cancel it.

Author

Aman Arora (aman.arora9848@gmail.com)

Version

1.0

Date

15/03/2023

Parameters

in	*/robot/goal_pos_param/x_goal*	The x position of the goal
in	*/robot/goal_pos_param/y_goal*	The y position of the goal

Actions:
° /reaching_goal

Description:

This node is used to create a client node that sends goals to the action server defined in package assignment_2_↩
2022. It interacts wuth the action serverusing the Planning.action file defined in the package "assignment_2_2022".

Usage Instructions:

- After launching the simulation using the launch file:
  - Enter 'p' and press [ENTER] to input a goal position (x, y) for the robot to move to.
  - Enter 'q' and press [ENTER] to cancel the current goal.

Index