UNIVERSITY OF PADUA

INFORMATION ENGINEERING DEPARTMENT (DEI)

MASTER'S DEGREE IN COMPUTER ENGINEERING

Report Assignment 2

Prof.
Emanuele Menegatti

Students:
Francesco Caldivezzi : 2037893
Manuel Barusco : 2053083
Riccardo Rampon : 2052416

Contents

1 Assumptions **2**
1.1 Navigation Assumptions . . . 2
1.2 Manipulation Assumptions . . . 3
1.3 Detection Assumptions . . . 5

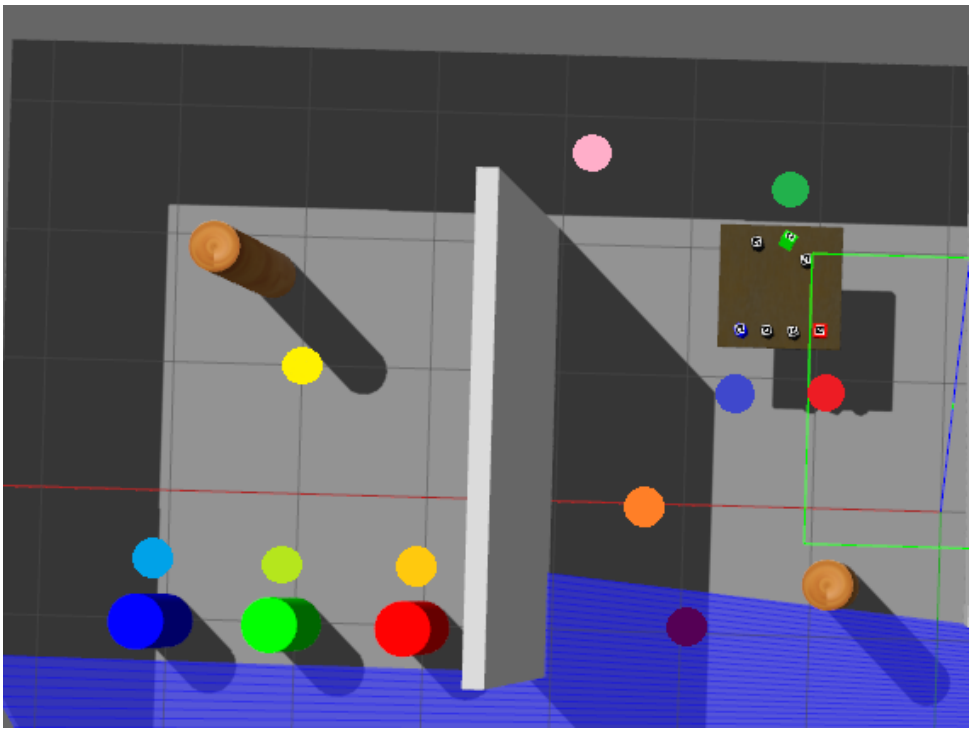
2 Packages **5**
2.1 Package Structure . . . 5
2.2 detection package . . . 5
2.3 detection_msgs package . . . 7
2.4 manipulation package . . . 7
2.5 manipulation_msgs package . . . 8
2.6 navigation_automatic_ros package . . . 8
2.7 solution package . . . 8

1 Assumptions

In this section we are going to describe the various assumptions that were made in order to solve this assignment.

1.1 Navigation Assumptions

For moving around the robot we have decided to use 10 waypoints. To be precise, such waypoints are described with the following image :



Notice that :

- **red**, **blue**, **green**, pink, **orange**, **dark violet** and yellow colored waypoints are used for the solution of the Assigment with and without the extra point part.
- **light blue**, **light green** and **light orange** colored waypoints are used in addition to the previous for solving the part of the Assigment without extra points.

Moreover, the meaning of the single waypoints is the following :

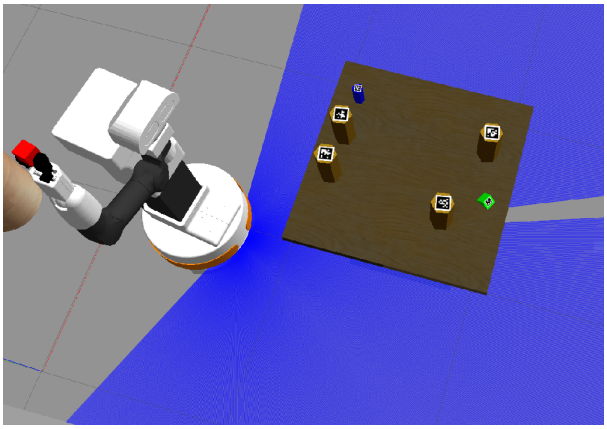
- **red** waypoint : is the pick-position for the red cube.
- **blue** waypoint : is the pick-position for the blue hexagon.
- **green** waypoint : is the pick-position for the green tetrahedron.
- **dark violet** waypoint : is the waypoint used to avoid that the robot try to go through the circular obstacle after the corridor and the wall of the room.
- **orange** waypoint : is the waypoint used only after having picked the red cube or blue hexagon for avoiding to hit the table if the robot tries to move directly from **red** or **blue** waypoints to the pink waypoint.
- pink waypoint : is the waypoint reached after having picked an object. Notice that when the robot picks the green tetrahedron it is reached from the **green** waypoint, while when the robot picks the blue hexagon or the red cube it is reached from the **orange** waypoint.

- yellow waypoint : is the docking position used for obtaining the centers of the place tables (It is used both for solving the Assigment in the extra point and no extra point version, but in the no extra point version it determines only the centers of the cylindrical tables for building the pose of where to place the object, while in the extra version of the assigment, in addition to that it is used also for determining the position of the table of where to place the current picked object).
- light blue waypoint : is the position of the robot for placing the blue hexagon in the corresponding cylindrical place-table of blue color. (Used only for no-point extra version of the Assigment)
- light green waypoint : is the position of the robot for placing the green tetrahedron in the corresponding cylindrical place-table of green color. (Used only for no-point extra version of the Assigment)
- light orange waypoint : is the position of the robot for placing the red cube in the corresponding cylindrical place-table of red color. (Used only for no-point extra version of the Assigment)

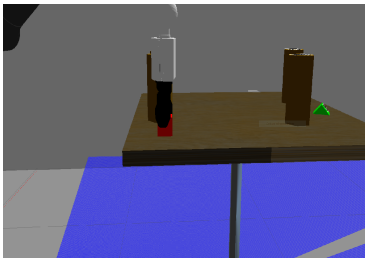
1.2 Manipulation Assumptions

The assumptions made in the manipulation part are mainly concerned the following aspects: intermediate pose of TIAGo's arm and gripper after the pick phase, the different picking poses of the objects and the approach pose used to pick the objects. We provide also some images for better understanding our assumptions:

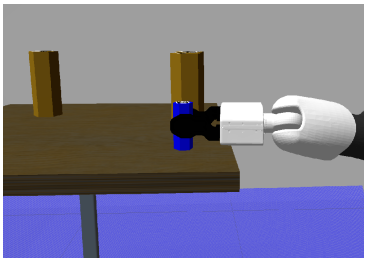
1. We used a predefined pose for TIAGo's arm and gripper before and after the pick phase, so that it is away from the planning scene in front of the robot and does not obstruct its view. We have chosen a slightly elevated side position, as can be seen in the figure:



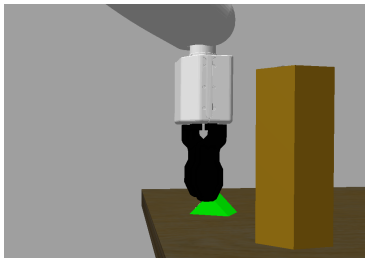
2. Next, we engineered a specific picking pose for each object, in particular, a pose from above for the green tetrahedron and the red cube, and a side one, instead, for the blue hexagon, as can be seen in the figures:



Pick red



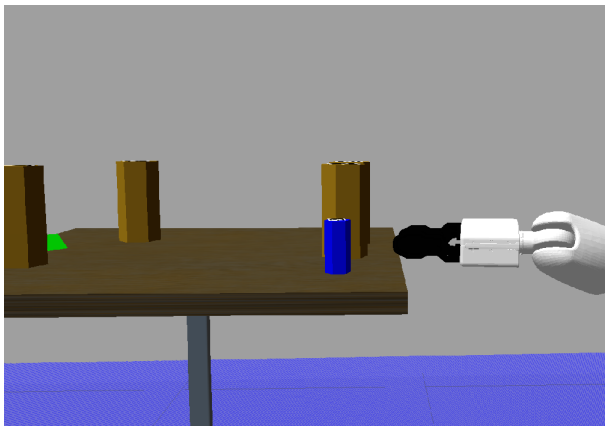
Pick Blue



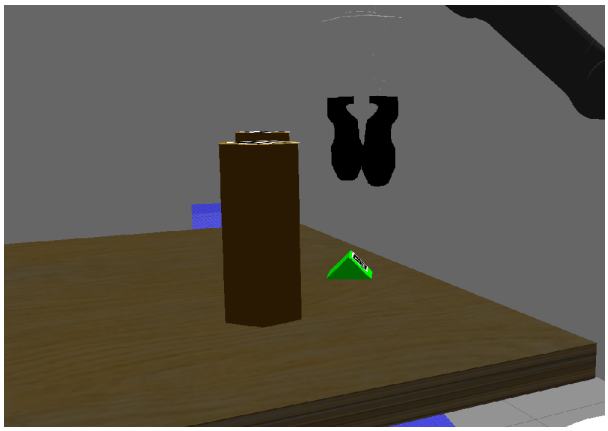
Pick Green

3. Finally, we decided to use different approach distances to grasps the different object, i.e. :

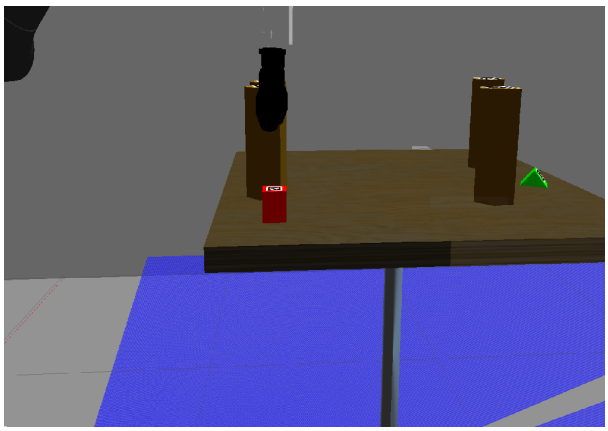
- for the Blue Hexagon, we modified the actual pick pose, by :
  - subtracting on the x axis of the pick pose $GRIPPER\_LENGTH + (HEXAGON\_DIAMETER/2.0) + offset$ where offset is equal to 0.1 (10 cm);
  - adding on the z axis of the pick pose $HEXAGON\_HEIGHT/3.0$;



- for the Green Tetrahedron, we modified the actual pick pose, by :
  - adding on the z axis of the pick pose $GRIPPER\_LENGTH + (TRIANGLE\_HEIGHT/2.0) + offset)$ where offset is equal to 0.1 (10 cm);



- for the Red Cube, we modified the actual pick pose, by :
  - adding on the z axis of the pick pose $GRIPPER\_LENGTH + offset$ where offset is equal to 0.1 (10 cm);



1.3 Detection Assumptions

During the detection of :

- the position of the pick-table;
- the position of the objects on the pick-table;

we have assumed that the robot is placed in one of the picking positions i.e. either **red**, **blue** or **green** positions (as illustrated before in the Navigation Assumption part) and is already lifted up. With these assumptions we can assume that the pick-table is in front of the robot and, in order to look at it, the robot has to move the head a little bit.

2 Packages

In this section we are going to first describe the main structure of each package, and then what each package does, what it contains and the implementation choices made.

2.1 Package Structure

We have used mainly two structures for the packages :

1. **"Implementation and Node Package"**: this package structure is used to contain, as suggested by the name, the code i.e. classes, node implementations and declarations (files ".cpp" and ".h").

```
package_name
├── include
│   └── package_name
│       ├── constants
│       │   └── constants.h
│       └── functionality_name
├── src
│   └── functionality_name
├── CMakeLists.txt
└── package.xml
```

2. **"Message Package"** : this package structure is used to contain, as suggested by the name, the messages, actions and services of the **package_name** 's package.

```
package_name_msgs
├── action
├── msg
├── srv
├── CMakeLists.txt
└── package.xml
```

2.2 detection package

The **detection** package contains everything concerning the detection of the objects of interest in the task, such as the collision objects in the table, the colored objects and the colored cylinders. This package is divided into five parts:

1. **obstacle_extractor**, that includes all the components useful for the detection of walls and cylindrical obstacles using the laser_scan measurements. This part was developed during the Assignment 1.
2. **pose_table_detector**, that includes all the components useful for the detection and recognition of the colored cylinders. This part starts from the cylinder detections of the obstacle_extractor part and choose the correct cylinder where TIAGo has to place the picked-object.

3. **poses_detection**, that includes all the components useful for the April Tags poses detection. The poses are detected using the ROS April Tag detection package provided by ROS and are transformed from the camera reference frame (**xtion_rgb_optical_frame**) to the **base_link** frame with the utilities of the poses_transformer part (following point).

4. **poses_transformer**, that includes all the components useful for transforming the April Tags poses from the camera reference frame (**xtion_rgb_optical_frame**) to the **base_link** reference frame using the TF package provided by ROS.

5. **table_objects_detection**, that includes all the components that are useful for the table detection and for the detection of the colored and collision objects that are present on it. This part gives all the directives to the head_movement part of the manipulation package (see the section 2.4 of the report) in order to move the head in the correct way.

In particular:

1. **obstacle_extractor** contains :

   - **obstacle_extractor_server.cpp** and **obstacle_extractor_server.h** that defines a service server that detects circles i.e. position (x and y) and radius, given the laser range finder data, in the **base_laser_link** reference frame. Those circles are published through the topic **/obstacle_extractor**;
   - **obstacle_extractor_node.cpp** which is the node that instantiate and start the service server previously defined;

2. **pose_table_detector** contains :

   - **pose_table_detector_server.cpp** and **pose_table_detector_server.h** that defines a service server that is used only for solving the extra point of the Assigment and that computes the pose of the table where the picked object must be placed by analyzing a frame captured with the camera and understanding on which place-table to go given the color of the picked object;
   - **pose_table_detector_node.cpp** which is the node that instantiate and start the service server previously defined;

3. **poses_detection** contains:

   - **poses_detection_publisher.cpp** and **poses_detection_publisher.h** that define a publisher that publishes in the topic **/table_objects_detection** all the April Tags poses already transformed, thanks to the **poses_transformer** part tools, to the base_link frame with the relative April Tag id;
   - **poses_poses_detection_node.cpp** which is the node that instantiate and start the publisher previously defined;

4. **poses_transformer** contains:

   - **poses_transformer_server.cpp** and **poses_transformer_server.h** that define a service server that transform a given pose (in the request message) from a given reference frame to another reference frame, both specified in the request;
   - **poses_poses_detection_node.cpp** which is the node that instantiate and start the service server previously defined;

5. **table_objects_detection** contains:

   - **table_objects_detection_server.cpp** and **table_objects_detection_server.h** that define an action server that detect the table and the colored and collision objects on the table. The implementation of this server is the following. First, it detects the table by tracking its color from the images acquired from the TIAGo camera, then, when the table is well detected and pointed from the TIAGo camera, it points the head toward the colored object (by searching its color) in order to acquire the April Tag pose of the tag placed on the object, and then start to search and track all the others April Tags that are placed on the collision objects present on the table.
   - **table_objects_detection_node.cpp** which is the node that instantiate and start the action server previously defined;

2.3 detection_msgs package

detection_msgs package contains all the messages that are used in the components previously defined. In particular it contains:

- **Detection.msg** simple message that contains a geometry_msgs/Pose field and an integer id associated to the pose. This message is used for managing an April Tag detection pose that is transformed from the camera reference frame to the base_link reference frame in order to save the transformed pose and also the id of the April Tag pose.
- **Detections.msg** simple message that contains an Array of **Detection** messages. This message is used for managing all the April Tags detections, all transformed in the base_link reference frame and with their April Tag id associated, in the **table_objects_detection_server**.
- **Transform.srv** service message that is used with the **poses_transformer_server**. The request contains the source and target reference frames names and the pose that has to be transformed from the source to the target reference frame. The transformed pose is returned in the service response.
- **ObjectsDetection.action** action message that is used with the **table_objects_detection_server**. The request contains the colorID that TIAGo has to pick (and obviously correctly detect) from the table, the response contains a **Detections** field that contains the poses, already transformed to the base_link reference frame, of the detected April Tags on the table. To be more precise, the main purpose of this server is to correctly detect the table and then the colored object April Tag that is on the table, but during the detection is also detecting others April Tags, so it is returning also those poses because are the collision objects poses, that are used during the pick phase. The feedback contains an integer that indicates the status of the action execution: 0 when the robot is searching for the table, 1 when it is searching for the colored object tag, 2 when it is searching for other tags (collision objects).
- **Circle.msg** simple message that contains 3 floating point numbers used for maintaining the x, y, coordinates of a circle and the radius of a circle.
- **Circles.srv** service message that is used with the **obstacle_extractor_server**. The request contains the scan data while the response contains an array of the circles detected.
- **PoseTableDetector.srv** service message that is used with the **pose_table_detector_server**. The request contains an array of circles that corresponds to the positions of the cylindrical table w.r.t. to the map reference frame as well as the id of the object to place. The output is the circle that correspond to the place cylindrical table corresponding to the given id.

2.4 manipulation package

manipulation is the package that contains everything concerning the manipulation of the TIAGo robot. In our implementation it is divided into three parts:

1. **head_movement**, that includes all the components useful for the movement of the TIAGo's head;
2. **pick_place**, that includes all the components useful to complete the pick and place process;
3. **torso_lifter**, that includes all the components useful to lift the TIAGo's torso, helpful during the detection phase;

In particular:

1. **head_movement** contains:
    - **head_movement_node.cpp** which is the node that implements a ServiceServer that advertise the topic **/head_movement**;

- **head_movement_server.cpp** and **head_movement_server.h** which are respectively the implementation and the declaration files of the ServiceServer cited previously, i.e. the HeadMovementServer class perform the movement of TIAGo's head, and the movement of TIAGo's head with an image pixel pointing by using the control_msgs::PointHeadAction client;

2. **pick_place** contains:

   - **pick_place_node.cpp** which is the node that implements an ActionServer that waits that a client sends to it the goal through the topic **/pick_place**;
   - **pick_place_server.cpp** and **pick_place_server.h** which are respectively the implementation and the declaration files of the ActionServer cited previously, i.e the PickPlaceServer class that perform the pick and place procedure, after the construction of properly Planning Scene;

3. **torso_lifter** contains:

   - **torso_lifter_node.cpp** which is the node that implements a ServiceServer that advertise the topic **/torso_lifter**;
   - **torso_lifter_server.cpp** and **torso_lifter_server.h** which are respectively the implementation and the declaration file of the ServiceServer cited previously, i.e. the TorsoLifterServer class that performs TIAGo's torso lift by going to properly set the values of the joint variables regarding the torso;

2.5 manipulation_msgs package

manipulation_msgs package contains :

- **HeadMovement.srv** which is the Service message used to make a request to the HeadMovementServer (of manipulation package) and providing a response;
- **PickPlace.action** which is the Action message used for requesting a goal to the PickPlaceServer (of manipulation package);
- **TorsoLifter.srv** which is the Service message used to make a request to the TorsoLifterServer (of manipulation package) and providing a response;

2.6 navigation_automatic_ros package

navigation_automatic_ros is the package that was developed during the Assignment 1 and reused here to move the robot around.
In particular it contains :

- **tiago_server_node.cpp** which is the node that implements an ActionServer that waits that a client sends to it a goal through the topic **/tiago_server**.
- **tiago_server.cpp** and **tiago_server.h** which are the implementation file and declaration file of the ActionServer cited previously i.e. TiagoServer class, that actually act as a proxy, because it accepts goals from clients and, use inside of it an ActionClient to send a goal to the MoveBase ActionServer through the topic **/move_base**.
- **MoveDetect.action** which is the Action message used for requesting a goal to the TiagoServer.

Notice that, this package do not satisfy the structure used for the package previously mentioned.

2.7 solution package

solution is the package that contains the clients for connecting to the servers of the various packages as well as the main files and launch files to run the solution in the no-extra point version and extra-point. So, the content, divided for which package will each file interacts to is :

- For **detection** package :

  - **obstacle_extractor_client.h** and **obstacle_extractor_client.cpp** which are respectively the declaration and implementation files of the ObstacleExtractorClient class which is a ServiceClient that is used to send a request through the topic **/obstacle_extractor** to the corresponding server in order to obtain the obstacles detection (cylinder obstacles and walls);
  - **pose_table_detector_client.h** and **pose_table_detector_client.cpp** which are respectively the declaration and implementation files of the PoseTableDetectorClient class which is a ServiceClient that is used to send a request through the topic **/pose_table_detector** to the corresponding server in order to obtain the pose of the correct cylinder where to pose the colored object;
  - **poses_transformer_client.h** and **poses_transformer_client.cpp** which are respectively the declaration and implementation files of the PosesTransformerClient class which is a ServiceClient that is used to send a request through the topic **/poses_transformer** to the corresponding server in order to transform the pose provided in the service request from a given reference frame to another (provided in the request);
  - **table_objects_detection_client.h** and **table_objects_detection_client.cpp** which are respectively the declaration and implementation files of the TableObjectsDetectionClient class which is an ActionClient that is used to send a request through the topic **/table_objects_detection** to the corresponding server in order to obtain the poses of the colored object and collision objects on the table (w.r.t. base_link frame);

- For **manipulation** package :
  - **head_movement_client.h** and **head_movement_client.cpp** which are respectively the declaration and implementation files of the HeadMovementClient class which is a ServiceClient that is used to send a request through the topic **/head_movement** to the corresponding server in order to move the TIAGo's head;
  - **pick_place_client.h** and **pick_place_client.cpp** which are respectively the declaration and implementation files of the PickPlaceClient class which is an ActionClient that is used to send a goal through the topic **/pick_place** to the corresponding server in order to perform a pick or place operation;
  - **torso_lifter_client.h** and **torso_lifter_client.cpp** which are respectively the declaration and implementation files of the TorsoLiferClient class which is a ServiceClient that is used to send a request through the topic **/torso_lifter** to the corresponding server in order to lift up or down the TIAGo's torso;

- For **navigation_automatic_ros** package :
  - **tiago_client_client.h** and **tiago_client_client.cpp** which are respectively the declaration and implementation files of the TiagoClient class which is an ActionClient that is used to send a goal through the topic **/tiago_server** to the corresponding server in order to move TIAGo robot in a certain position;

- For **tiago_iaslab_simulation** package :
  - **human_client_client.h** and **human_client_client.cpp** which are respectively the declaration and implementation files of the HumanClient class which is a ServiceClient that is used to send a request through the topic **/human_objects_srv** to the corresponding server in order to move obtain the randomized order of the list of ids that must be picked;

While the main files and the corresponding files for launching the execution of the two different solution (no-point extra and point-extra) are :

- **main_no_extra.cpp** and **main_extra.cpp** which are the two executable files that contains the logic on how the robots moves in terms of navigation and manipulation point of view. In fact, to do so, inside of those files there are a lot of instances of the previously mentioned clients that contact the opportune server to perform a task.

- **no_extra.launch** and **extra.launch** which are the two launch files that are used to execute the nodes of the packages previously mentioned.