



Zeid Kootbally
2022
College Park, MD

version 1.0

ENPM809Y: Final Project (Fall 2022)

Due date: **Friday, December 16, 2022, 11 pm**

Contents

1 Updates	3
2 Document Conventions	3
3 Programming Style Guide	3
4 Syllabus Changes	4
5 Prerequisites	4
6 Fiducial Markers	5
7 Environment	6
8 Proportional Controller	7
9 Deliverable	9
10 Tasks	10
10.1 Broadcaster	10
10.2 Parameters	11
10.3 Locate the Fiducial Marker	13
10.3.1 Move the Robot to Goal #1	13
10.3.2 Find the Fiducial Marker	13
10.4 Move the Robot to Goal #2	14
10.4.1 Retrieve the Goal Coordinates	14
10.4.2 Compute the Goal in the Odom Frame	14
10.4.3 Move the Robot to the Final Destination	14
11 Report 75 pts	15
11.1 Report Guidelines	16
12 ROS Package 125 pts	16

1 Updates

This section describes updates added to this document since its first released. Updates include addition to the document and fixed typos. The version number seen on the cover page will be updated accordingly.

- 12/03/2022 – **v1.0**: The document has been rewritten and now provides more detail explanation. One of the biggest additions can be found in Section 10.4.2. It is highly suggested to read this document one more time.
 - Make a backup of your current packages and clone a new version of the packages. If you already started working on this assignment you should definitely back up the packages `odom_updater`, `target_reacher`, and the launch file `final.launch.py`. To be safe, make a backup of the whole `final` folder.
 - Next, `git clone https://github.com/zeidk/enpm809y_FinalFall2022.git`
- 12/01/2022 – **v0.3**: One important information was missing in the document. When the robot reaches a goal, a Message is published on the Topic `goal_reached`. This information was added to step #4 on page 7.
- 11/30/2022 – **v0.2**: Fixed typos, added section 4, changed the submission deadline, and added new figures describing the proportional controller.
- **v0.1**: Original release of the document.

2 Document Conventions

In this document, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

- This is a [link](#).
- `file.txt` `folder` `package` `tool`
- `>_ terminal command`
- `Message` `Topic` `Node` `Frame` `Parameter`
- Code snippets are described as follows:

```
int main(){
    std::cout << "Hello\n";
}
```

3 Programming Style Guide

Students must use the following programming style guide and conventions.

- C++ programming follows the [C++ Core Guidelines](#).

4 Syllabus Changes

- Presentation has been removed.
- **50 pts** for presentation have been split.
 - **25** points added to the ROS package.
 - **25** points added to the report.

5 Prerequisites

- Create a new workspace (e.g., `>_ mkdir -p ~/final_ws/src`).
- Clone the git package in `final_ws/src`
 - `>_ cd ~/final_ws/src`
 - `>_ git clone https://github.com/zeidk/enpm809y_FinalFall2022.git`
- In your `.bashrc` add the following commands:
 - `export GAZEBO_MODEL_PATH=$GAZEBO_MODEL_PATH:<path>/<to>/tb3_gazebo/models`
 - `alias frames='cd /tmp && ros2 run tf2_tools view_frames && evince frames.pdf'`
- Close and reopen the terminal.
- Install `pip3` if not already installed.
 - `>_ sudo apt update`
 - `>_ sudo apt install python3-pip`
 - `>_ pip3 --version` (to check it was installed)
- Install package for OpenCV.
 - `>_ pip3 install opencv-contrib-python` (67.1 MB will be installed)
- In the package `target_reacher`
 - Edit `package.xml`
 - ◊ Edit the **maintainer** fields.
- Build the workspace.
 - `>_ cd ~/final_ws`
 - `>_ colcon build`
 - `>_ source install/setup.bash`
- Run the project.
 - `>_ ros2 launch tb3_bringup final.launch.py`

6 Fiducial Markers

In robotics, fiducial markers are mainly used for localization or object detection. For instance, you can stick a fiducial marker on an object and the robot will pick up the object if it detects the specific marker. Fiducial markers are used in this assignment differently. They are used to retrieve information on the final destination the robot has to reach. There is only one marker at a time in the environment and each marker has a unique ID (0, 1, 2, or 3). The marker used in the environment can be one of the 4 markers depicted in Figure 1.

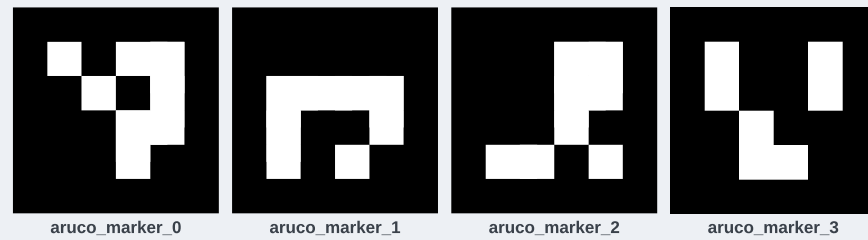


Figure 1: One of the 4 possible fiducial markers used in the environment.

7 Environment

The Gazebo environment is shown in Figure 2. There are 5 large dots and 16 smaller dots on the floor to help the user visualize where the robot is going.

- The large red dot shows the coordinates of a static frame named $\mathcal{F}_{\text{origin1}}$
- The large yellow dot shows the coordinates of a static frame named $\mathcal{F}_{\text{origin2}}$
- The large green dot shows the coordinates of a static frame named $\mathcal{F}_{\text{origin3}}$
- The large blue dot shows the coordinates of a static frame named $\mathcal{F}_{\text{origin4}}$
- The large white dot helps the user visualize where the marker is located in the environment.

The static Frames $\mathcal{F}_{\text{origin1}}$, $\mathcal{F}_{\text{origin2}}$, $\mathcal{F}_{\text{origin3}}$, and $\mathcal{F}_{\text{origin4}}$ can also be seen in RViz, as depicted in Figure 3. These static Frames will be used in retrieving the final destination of the robot. More information on these static frames is provided in Section 10.2.

The smaller dots represent all possible final destinations. The robot will reach one of these smaller dots if this assignment is properly implemented.

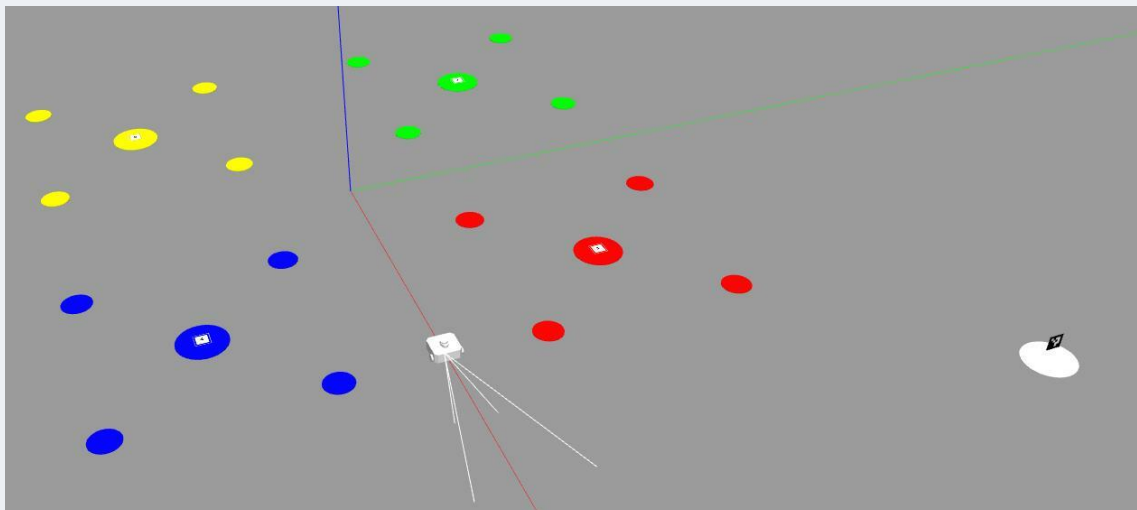


Figure 2: Gazebo environment.

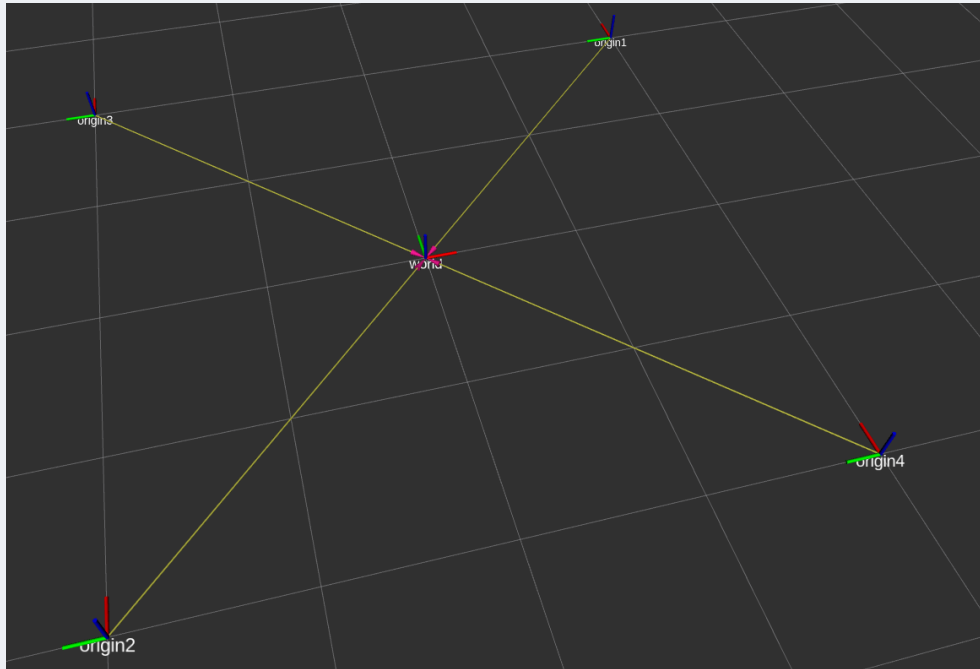


Figure 3: Static frames used in the final destination.

8 Proportional Controller

The method `go_to_goal()` from the package `bot_controller` uses a closed-loop (or feedback system) to make the robot reach a goal. A closed-loop control system looks at the current output and alters it to the desired condition. The control action in these systems is based on the output.

A proportional-integral-derivative (PID) controller (see Figure 4) is a control loop mechanism employing feedback that is widely used in industrial control systems and a variety of other applications (e.g., temperature control, the head positioning of a disk drive) requiring continuously modulated control. The controller used in `go_to_goal()` is a proportional controller (see Figure 5).

Moving to a point $G(x_G, y_G)$ in a 2D plane:

- Linear velocity: $v = K_v \sqrt{(x_G - x)^2 + (y_G - y)^2}$, $K_v > 0$
- Angular velocity: $\gamma = K_h(\theta_G \ominus \theta)$, $K_h > 0$

With the proportional controller used in this assignment, the error and the velocity of the robot are large at the beginning. While the errors decrease the velocity decreases as well (see Figure 6).

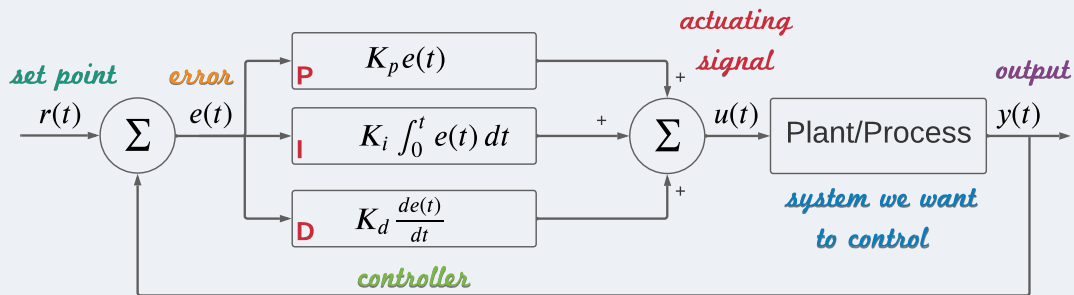


Figure 4: PID controller.

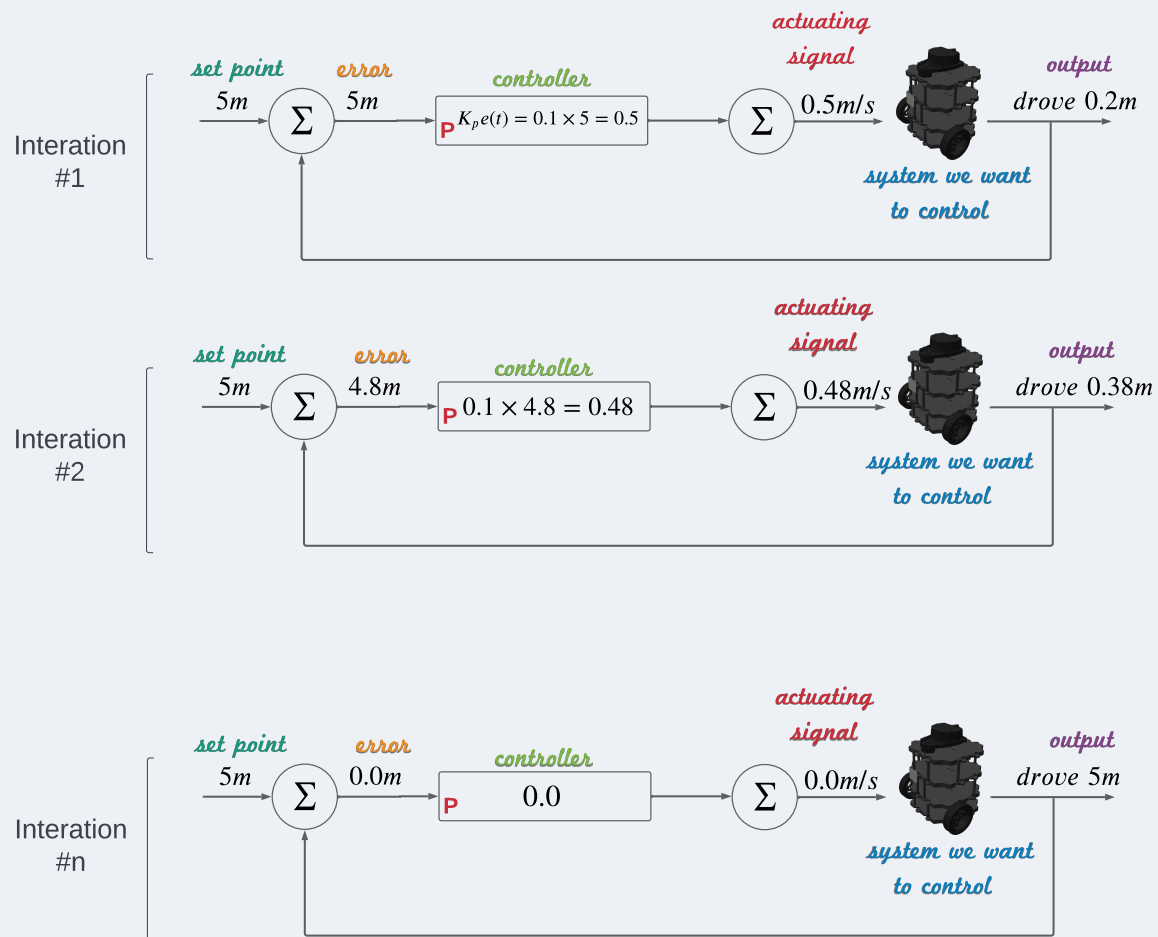


Figure 5: Proportional controller.

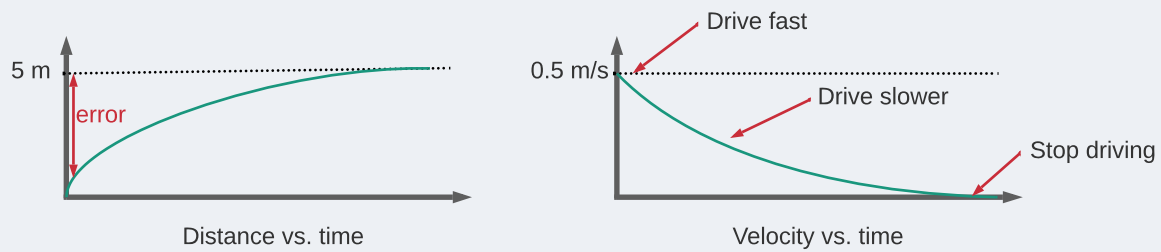


Figure 6: Errors and velocity over time.

9 Deliverable

1. A package `odom_updater` to broadcast `/robot1/base_footprint` in `/robot1/odom`
2. An augmented version of the package `target_reacher`. This is the only package that students are allowed to modify.
3. HTML Doxygen documentation should be generated in the `doxygen` folder located in the package `target_reacher`
4. A report describing your work.

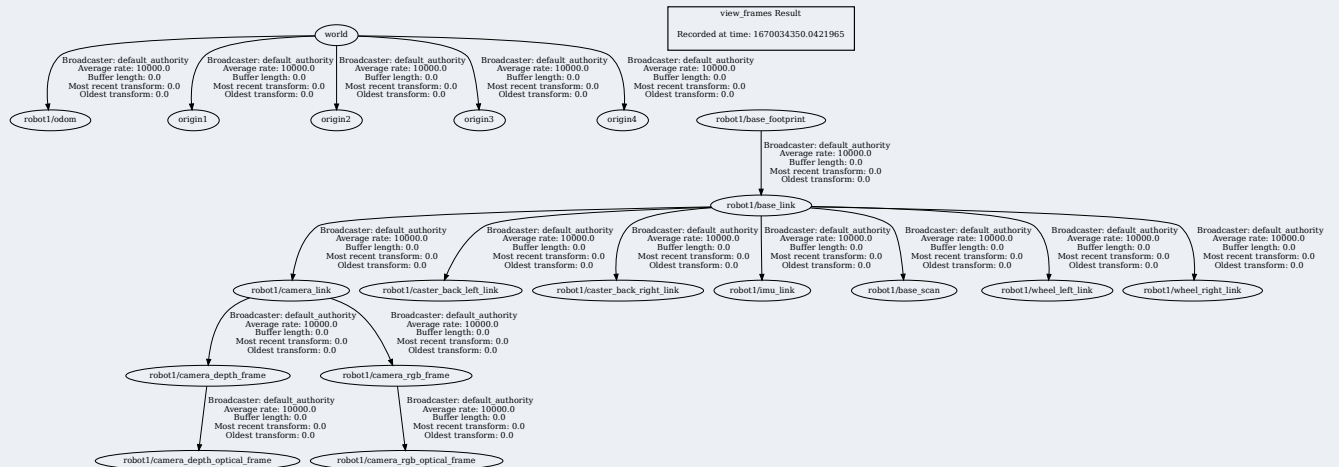


Figure 7: Disconnected trees.

10 Tasks

The main tasks to perform in this assignment are described in the following subsections.

10.1 Broadcaster

After launching the application, a look at the TF tree (use the command `>_ frames`) shows disconnected trees (see Figure 7). Since all the frames have to be connected to properly perform transforms, your first task is to connect these two trees into one tree using the steps described in Listing 10.1.

10.1: Steps for the broadcaster Node.

1. Create the package `odom_updater` with the following dependencies:
 - `rclcpp`, `geometry_msgs`, `nav_msgs`, and `tf2_ros`
2. Create the Node `N odom_updater` in this package to broadcast `/robot1/base_footprint` as a child of `/robot1/odom`
 - Since `/robot1/base_footprint` is a movable Frame, the broadcaster cannot be a static broadcaster. An example of a non-static broadcaster can be seen at [Writing a broadcaster](#).
 - A subscriber to the Topic `/robot1/odom` is needed to retrieve the pose of the robot in `odom`. In the callback method for the subscriber, call a function which does the broadcast.
3. Edit `final.launch.py` to start `N odom_updater`

Once this task is completed, you should get a tree similar to the one depicted in Figure 8.

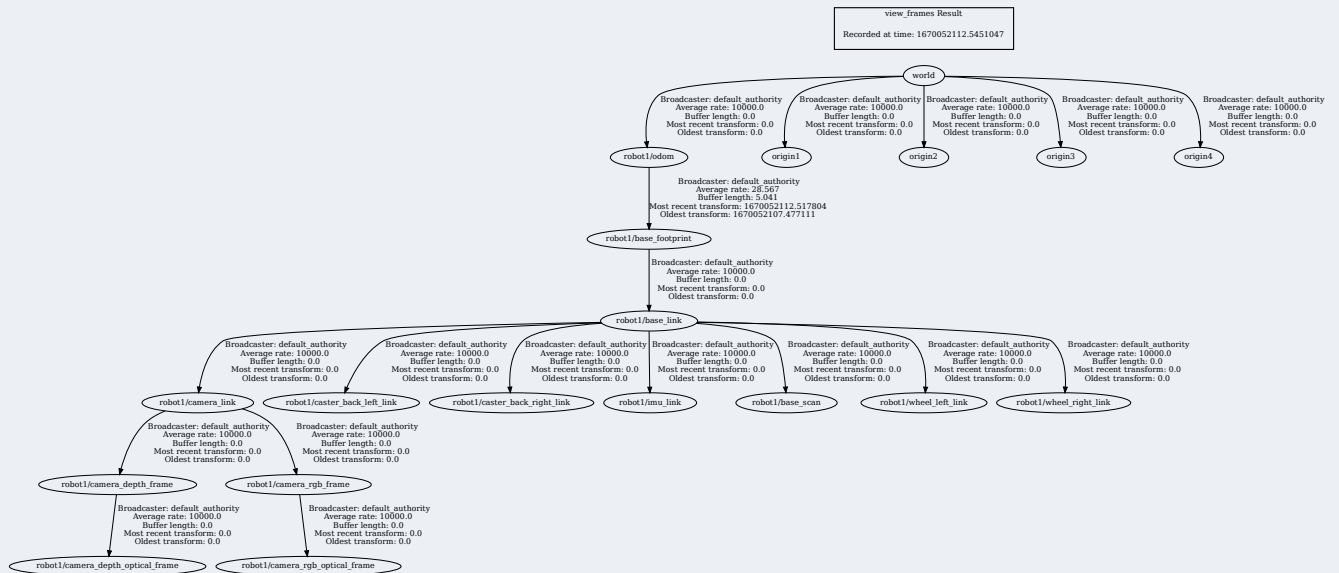


Figure 8: Connected tree.

10.2 Parameters

This task consists of starting `target_reacher` with Parameters located in

`final_params.yaml`, which is written in the YAML language. Listing 10.1 shows the content of this Parameter file.

Listing 10.1: Parameters for the Node `target_reacher`

```

1 target_reacher: # node Name
2   ros_parameters:
3     aruco_target:
4       x: 6.5
5       y: 5.0
6     final_destination:
7       # "origin1", "origin2", "origin3", or "origin4"
8       frame_id: "origin1"
9     aruco_0:
10      x: 1.0
11      y: 1.0
12     aruco_1:
13      x: 1.0
14      y: -1.0
15     aruco_2:
16      x: -1.0
17      y: -1.0
18     aruco_3:
19      x: -1.0
20      y: 1.0

```

- In YAML, comments start with `#`
- Line 1: This is the name of the Node for which Parameters in this file will be loaded.
- Line 2: `ros__parameters` tells ROS that all the fields that come afterwards are Parameters.
- Lines 3–5: These parameters indicate where the robot should go to be able to find the fiducial marker.
- Line 6: Everything under the field `final_destination` is related to the second goal that the robot has to reach.
 - Line 8: `frame_id` describes the Frame in which the pose of the goal is referenced. This field can take one of the 4 following values: `"origin1"`, `"origin2"`, `"origin3"`, or `"origin4"`. All these static Frames are already broadcast on `tf_static` and are depicted in Figure 3.
 - Lines 9–20: The field `aruco_i` corresponds to the fiducial marker with the ID `i`. The fields `x` and `y` represent the coordinates of the final destination in the Frame `frame_id`. For instance, in the YAML file provided in Listing 10.1, the coordinates of the final destination are (1.0, 1.0) in the Frame `origin1` if the detected fiducial marker has the ID 0.

The steps to perform this task are described in Listing 10.2.

10.2: Load parameters from a file.

1. In `final.launch.py`, start the Node `target_reacher` with the Parameter file `final_params.yaml`. This was shown in class.
2. In the constructor of `TargetReacher`, write the following for each Parameter (an example was provided in class).

```
this->declare_parameter<Type>(<parameter name>);
```

10.3 Locate the Fiducial Marker

This step consists of moving the robot to a goal and make the robot rotate in place until it can detect the fiducial marker.

10.3.1 Move the Robot to Goal #1

Move the robot to a location where the fiducial marker can be found. To send the robot to a location (x, y), use `m_bot_controller->set_goal(x, y)`; where x and y are `aruco_target.x` and `aruco_target.y`, respectively.

Once a goal is reached, a Message is published to the Topic `goal_reached`. Therefore, you need a subscriber to this Topic.

10.3.2 Find the Fiducial Marker

Goal #1 guarantees the fiducial marker can be found. To do so, make the robot rotate on itself by publishing `Twist` Messages on `/robot1/cmd_vel` (linear velocity is null and angular velocity is 0.2). Once a fiducial marker is found, a Message is published on `aruco_markers` by `aruco_node` from `ros2_aruco`. You need a subscriber to `aruco_markers`. An example of a Message published on `aruco_markers` is depicted in Listing 10.2. The field `marker_ids` is a vector of integers. This vector has at most 1 item since there is only 1 fiducial marker in the environment.

Listing 10.2: A Message published on `aruco_markers`

```
header:
  stamp:
    sec: 30
    nanosec: 549000000
  frame_id: /robot1/camera_rgb_optical_frame
marker_ids:
- 0
poses:
- position:
  x: -0.2393652624689967
  y: -0.04005704515270145
  z: 0.5849345455223367
  orientation:
    x: 0.9908618144447304
    y: -0.0024511286022593664
    z: 0.13481463215468573
    w: -0.003445518954903101
---
```

10.4 Move the Robot to Goal #2

Goal #2 is the final destination of the robot. Once the robot has reached this second goal, there is nothing else to do and you have completed the project.

10.4.1 Retrieve the Goal Coordinates

Using the marker ID retrieved from Section 10.3.2, retrieve the goal coordinates from Parameters `final_destination.aruco_i.x` and `final_destination.aruco_i.y`, where i is the ID of the marker found by the robot.

10.4.2 Compute the Goal in the Odom Frame

The retrieved coordinates from the previous step is not in `/robot1/odom` but in the Frame assigned to `final_destination.frame_id` (refer to Listing 10.1). Therefore, you need to retrieve the value for this Parameter. Based on the example provided in Listing 10.1, we need to transform this goal to `/robot1/odom`. To do so, follow the steps provided in Listing 10.3.

10.3: Broadcast and Transform.

1. Broadcast a frame (name it `final_destination`) in the frame `origin1`. The result should be similar to the one shown in Figure 9. The pose of this frame in `origin1` is:
 - position: (`final_destination.aruco_i.x`, `final_destination.aruco_i.y`, 0)
 - orientation: (0, 0, 0, 1)
2. Write a transform listener to get the transform between `final_destination` and `/robot1/odom`. The result gives the pose of the final destination in the Frame `/robot1/odom`.
3. The result should point to one of the smaller dots in the environment (refer to Figure 2).

10.4.3 Move the Robot to the Final Destination

Use the result from the previous step to send the robot to its final destination.

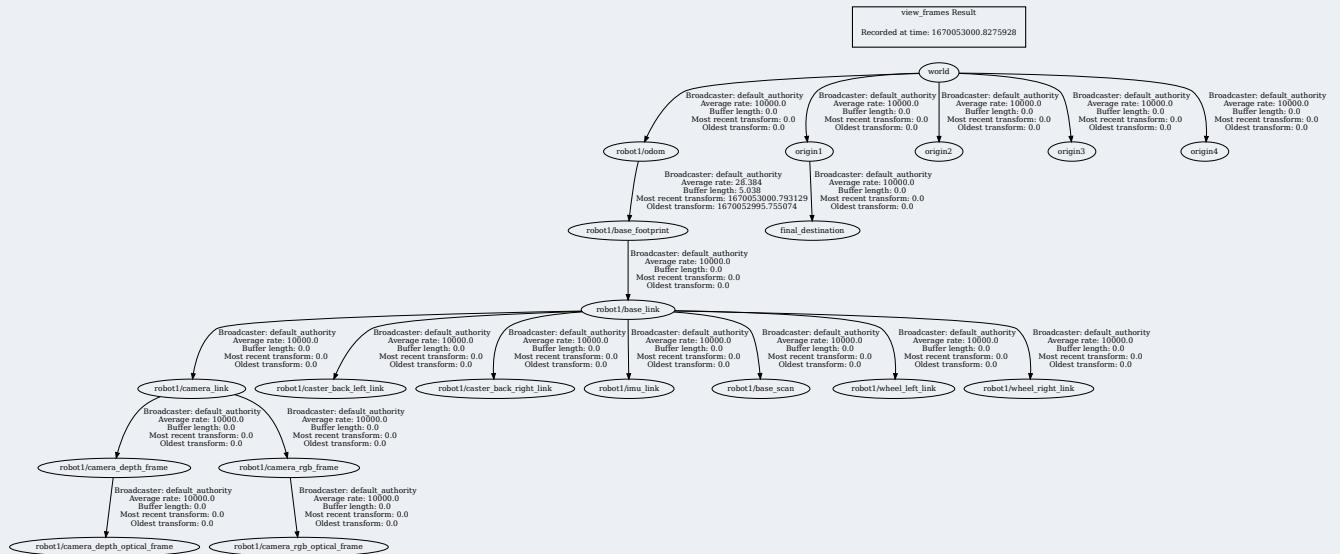


Figure 9: Broadcast of the Frame \mathcal{F} final_destination in \mathcal{F} origin1.

11 Report 75 pts

A LaTeX template can be found on [overleaf](#) (sorry, no Word template). Make a copy of this template and do not edit the original version. Below is a proposed outline for the report.

- **Introduction (5 pts):** Describe the problem you are trying to solve.
- **Approach (45 pts):** This section is the most important one in the report. How did you break down the problem and how did you solve it? This is where you will include flowchart diagrams, pseudo-code, and class diagrams. Do not include any C++ code in this report, this is not a user manual. Points will be deducted if C++ code is included. : If you need any content from my slides or from the instruction files (such as pictures, figures, pseudo-code, etc), ask me and I will provide them to you. Do not take screen captures of my content, the resolutions will not be good for a report.
- **Challenges (20 pts):** What are the main challenges you faced while working on the project? e.g., "Difficulty of making the Turtlebot reach the target". "Computer stopped working" is not a challenge, it is an unfortunate event which tends to have a very high probability of occurring during the final project...for some reasons.
- **Contributions to the project (5 pts):** Who did what in the project? For instance:
 - Student #1: Worked on the broadcaster, code documentation, and report.
 - Student #2: Worked on the report.
 - Student #3: Worked on the code to find the fiducial marker.
- **Resources (optional):** You are free to use any resource. If you found some piece of C++ code which helped you solve part of the problem, make sure you include the website or github project this code was taken from.

- Course feedback (optional): This section is optional. If you can, please provide some feedback on the course. My goal is that you end this semester with C++ and ROS knowledge. If you did not acquire any of those things then there may be some things I need to fix.


11.1 Report Guidelines

This section provides guidelines on what and what not to do in a report. Points will be deducted if those guidelines are not followed.

- Before you start writing a report you need to define an outline. The outline was given to you this time.
- Next, you need to identify the most important parts of each section in the report. For instance, in the Approach section you should at least focus on the methodology used for the Turtlebot to reach each target. This looks like something interesting to discuss.
- Many students do not know how to use figures in a report. All figures must be captioned and referenced in the text. People read your report and look at the figures only if they are referenced in the text. Figures not referenced in the text are usually ignored by the reader. Your figures should be big enough and must have good resolution. Some figures are placed in the reports just to make the reports bigger without providing any value, just remove them if they are used to decorate the report.
- Many of you do not know how to write pseudo codes. Copying and pasting C++ code in your report does not constitute a pseudo code. You have to provide a name for the pseudo code, the parameters, the loops, etc. You can use variable names in the pseudo codes and they can be mixed with plain English.
- Proofreading is not always performed and this results in a large number of typos and inconsistencies.
- Flow chart diagrams always need a start and an end, otherwise we do not know how to read your diagrams. There is a convention for flow charts, for example, diamonds are used for decision with at least one input and with at least two outputs.
- The use of "we" and "our" is frowned upon. Rewrite your sentences to get rid of these words. Also, do not use "Now" in your report, e.g., "now we switch to moving to the final destination".
- Some reports are too verbose and it becomes difficult to read them. Provide diagrams and if more explanations are needed, write them in plain English.
- Very few students showed any results from accomplishing the projects. Results can be of different types (e.g., RViz screen captures, Gazebo screen captures, outputs from Topics).

12 ROS Package **125 pts**

The following steps will be used to grade the final project.

- We will use a different fiducial marker. This is to make sure you are retrieving the correct parameters dynamically and not hardcoding anything. You can use a different one yourself by editing the file  `tb3_bringup/bringup/worlds/empty_world.world`. Edit the following part of the file if you want to use a different fiducial marker (replace

aruco_marker_0 with aruco_marker_1 for instance):

```
<model name="aruco_marker_0">
  <static>1</static>
  <include>
    <uri>model://aruco_marker_0</uri>
    <pose>5.0 5.0 0.2 0 0 0</pose>
  </include>
</model>
```

- We will change the final destinations in `final_params.yaml`. Again, this is to make sure you are retrieving the parameters dynamically. You can change the final destination yourself by switching around the fields `aruco_0`, `aruco_1`, `aruco_2`, and `aruco_3`.
- We will change the value of `frame_id` in `final_params.yaml`.
- We will run your package once to see if the robot does what it is expected (go to marker location, look for marker, and go to the final destination).
- We will look at your code. We want to make sure that you properly used the style guide and conventions.
- We will look at the HTML files generated in the `doxygen` folder.