

# PROJECT 1 ROBOTICS

## Team members:

- 10612055 Filippo Tallon;
- 10575715 Pietro Bosoni;
- 10600844 Guido Sassaroli;

## Archive description:

Our archive is a Catkin Package named “first\_project” and consists of the following folders:

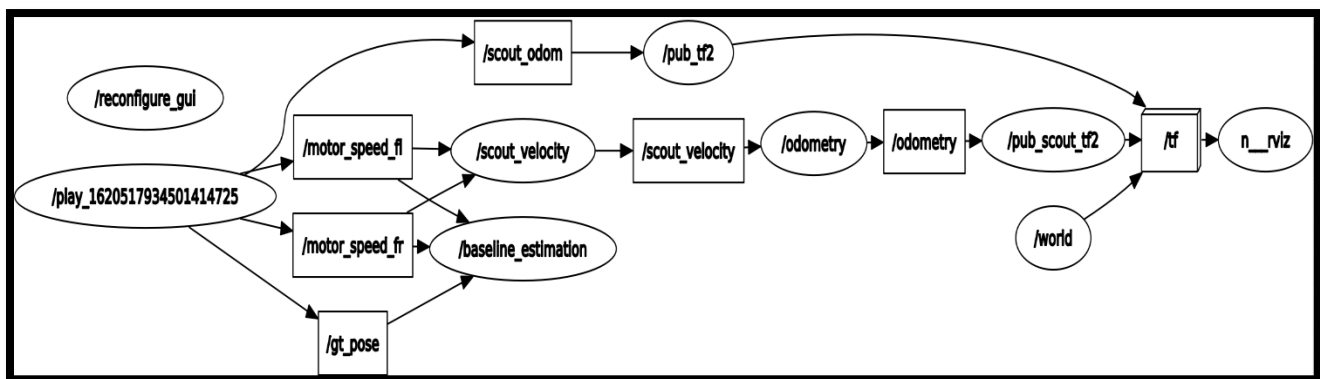
- The **src** folder contains all our code in C ++:
  - “**scout\_velocity.cpp**”: generates a node that subscribed to “/motor\_speed\_fl” and “/motor\_speed\_fr” topics and calculates the longitudinal and angular speed of the robot according to the differential kinematics formulas. Remark: using the definition of a skid steer robot we can assume that the two wheels on the same side of the robot have equal speed. Finally, the node publishes the results of the calculation on the topic “/scout\_velocity” with message of type “geometry\_msgs :: TwistStamped”.
  - “**parameters\_estimation.cpp**”: it is used to compute parameters (gear ratio and apparent baseline) of our project from scout\_odom.
  - “**baseline\_estimation.cpp**”: computes the apparent baseline from gt\_pose (see last paragraph for more info).
  - “**odometry.cpp**”: it subscribes to the topic “/scout\_velocity” from which you can read the speeds (longitudinal and angular). Then it uses the speeds to calculate the odometry of the skid steer robot, integrating with the Euler or Rugen-Kutta method. Finally, the odometry is published on two topics: “/odometry” (of type nav\_msgs::Odometry) and “/custom\_odometry” (our custom message of type first\_project::CustomOdometry).
  - “**pub\_tf2**”: it reads the coordinates of the robot from the odometry provided by the manufacturer (topic /scout\_odom) and creates the roto-translation transformation between the reference frame (header frame) “odom” and the “base\_link” frame fixed on the robot. Remark: we have applied a static transformation from “world” frame to “odom” in order to verify the truthfulness of our results.
  - “**pub\_scout\_tf2**”: it reads the odometry calculated by us (topic /odometry) and creates the transformation between the “scout\_link” reference frame and the “world” reference frame. Remark: the initial coordinates of the robot have been set by looking at the topic /gt\_pose when the bag starts (explained later).
- The **cfg** folder contains the “**methods.cfg**” file which manages the dynamic reconfigure. Inside there is an enumeration that allows you to switch between Euler (0) and Runge-kutta (1). Remark: the default integration method is Runge\_Kutta.
- The **srv** folder contains the “**ResetOdometryToPose.srv**” file, used in “odometry.cpp” to create the “/reset\_odometry\_to\_pose” Service.

The second service `"/reset_odometry"` instead uses a standard service `Empty (std_srvs::Empty)` since it does not need input parameters, but it takes the initial pose of the robot from Parameter Server.

- The **msg** folder contains the two Custom Messages used in the project. "MotorSpeed.msg" was provided to us at the beginning of the project, while we defined "CustomOdometry.msg" respecting the specific requests.
- The **launch** folder contains the file `"first_project.launch"` used to launch all project nodes. Inside, we can also find the initialization of the parameters that define the initial position and orientation of the robot (calculated via `gt_pose`) and the static transformation between "world" and "odom" used to verify the correct functioning of our odometry.

Remark: it is present another launch file `"baseline_estimation.launch"` that launches nodes required to estimate parameters.

- **Rviz** folder contains the configuration we use to view the simulation.

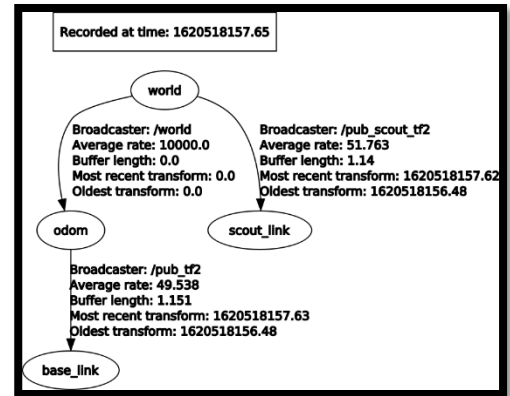


## ROS Parameters:

- **x0**, "double" type: it is the initial abscissa of our odometry with respect to the "world" reference frame.
- **y0**, "double" type: it is the initial ordinate of our odometry with respect to the "world" reference frame.
- **theta0**, "double" type: it is the initial orientation of our odometry with respect to the "world" reference frame. (rotation around the z axis).
- **GEAR\_RATIO**: it is defined as a macro in "scout\_velocity.cpp". In order to estimate it, we considered the robot in the instants in which it had a straight trajectory, when the speed of the center of the wheels is equal to the linear speed of the robot. Then we found the angular velocity of each wheel (knowing the radius) and compared it with the angular velocity of the electric motor.
- **APPARENT\_BASELINE**: it is the constant parameter used in the "odometry.cpp" file.
- We use also a "method\_enum" in the dynamic\_reconfigure in which we indicate the Euler method with 0 and Runge-Kutta with 1 (set as the default integration method, since it is more precise).

## Tf-tree:

- *world*: it is the fixed reference triad, generated by the Optitrack sensor system.
- *odom*: it is the triad in which we calculate the odometry.
- *base\_link*: it is the triad fixed with the robot, it is given by the manufacturer.
- *scout\_link*: it is the triad fixed with the odometry computed by us, with respect to the reference triad "world", offset by the initial coordinates (see "first\_project-launch").



## Structure of the Custom Message:

Our custom message "CustomOdometry.msg" contains two fields:

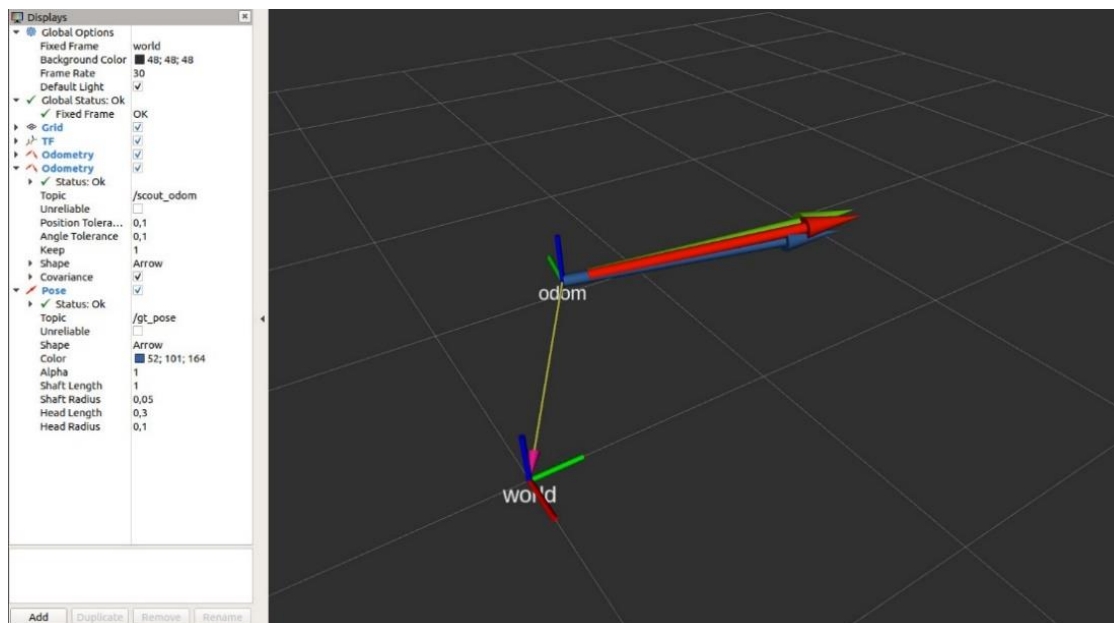
- "Method", of type `std_msgs::String`, containing the name of the selected integration method;
- "Odom", of type `nav_msgs::Odometry`, it contains the odometry;

## Description of how to start the nodes:

After the command `catkin_make` in your workspace, launch our project with `roslaunch first_project first_project.launch`: starts all the required nodes, including rviz opened in the configuration used by us to verify the correctness of our odometry.

In Rviz you can see:

- The topic `/gt-pose` with the positions measured by Optitrack (blue arrow).
- The topic `/odometry` with the positions calculated by us (green arrow).
- All reference triads. The *scout\_link* triad is expressed directly on the basis of the world triad; while *base\_link* is linked first to *odom* and then to *world* through a static transformation.



Remark: the file `.launch` starts also the "reconfigure\_gui", from which we can change the integration method.

Remark: In order to call our services you can use the following commands:

- `rosservice call /reset_odometry`
- `rosservice call /reset_odometry_to_pose [float] [float] [float]` (the pose is referred to the “world” reference frame).

### Things we believe are important:

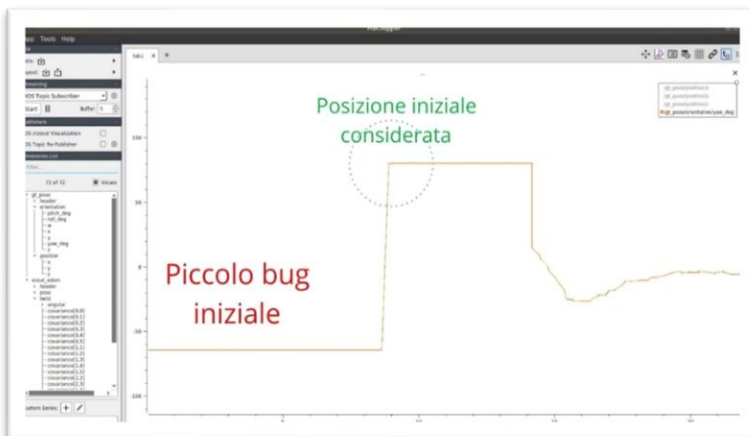
Estimation of the gear ratio and apparent baseline from scout odom: this computation is present in `parameters_estimation.cpp`. Notice that in order to estimate the gear ratio we focus when the robot is going straight, instead for the apparent baseline when is turning. We obtain the following results: **GEAR\_RATIO = 38**, **APPARENT\_BASELINE = 1.04**

*Remark:* it's a very gross estimate, parameters are time varying.

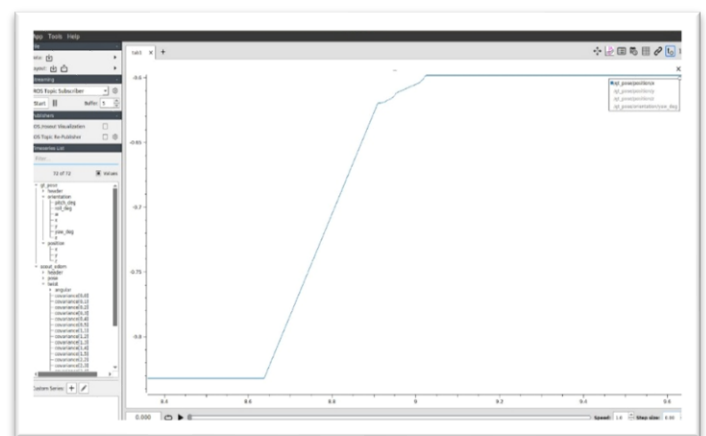
Estimation of the apparent baseline from gt\_pose: as it is possible to see in `baseline_estimation.cpp`, in order to compute the baseline from `gt_pose`, we try to replicate the odometry with different baseline and at each time we compute a cost function that minimize the difference between the two curves (`gt_pose` and our odometry). By doing so we obtain the following result: **APPARENT\_BASELINE = 0.762**.

*Remark:* we use the same gear ratio obtained from `scout_odom` (38) but probably the right way is to have a minimization with respect to both the baseline and the gear ratio.

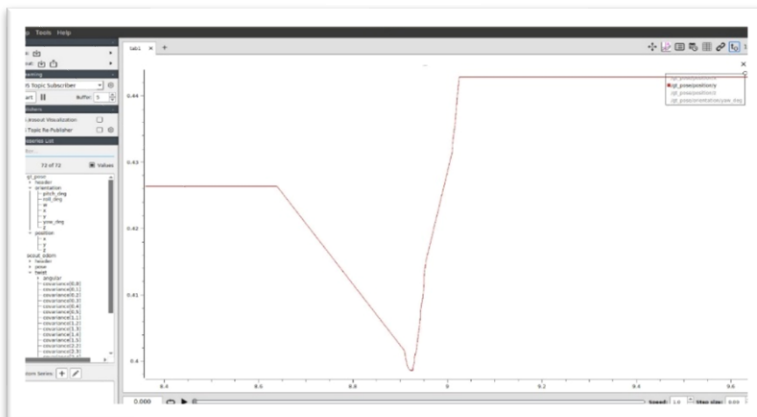
Computation of the starting position of our odometry: in order to calculate the initial coordinates of the odometry we used the topic "`gt_pose`", looking at its position and orientation in the initial instant.



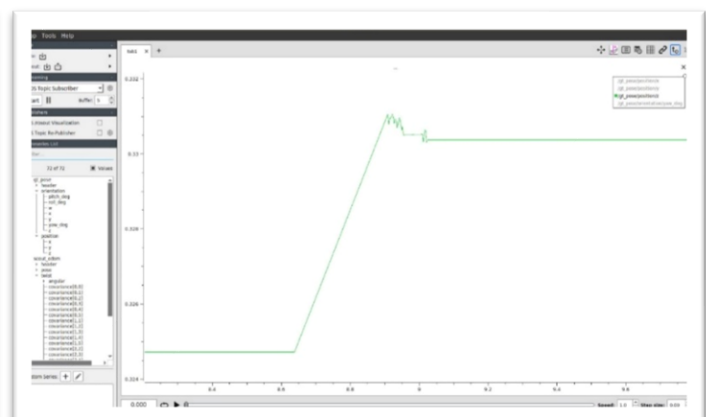
Yaw0= -80.234



x0= -0.598297894001



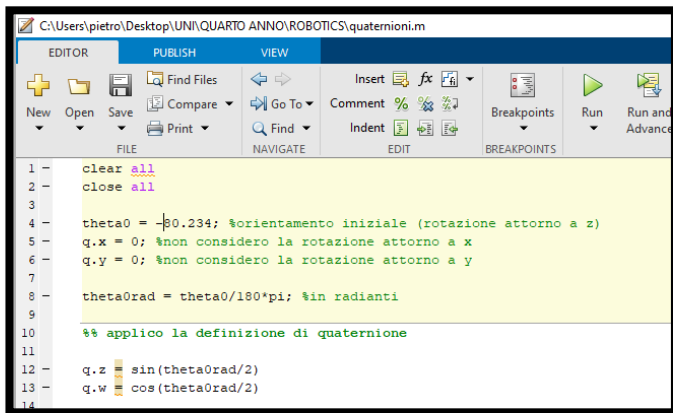
y0= 0.442829757929



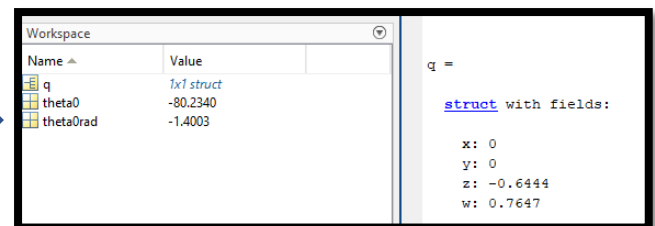
z0= 0.330364078283

Remark: the first seconds are blocked on an untrue (rather unnatural) position, so we used the first instant in which the robot begins its linear movement (look at the images above).

### Transformation of the initial Yaw angle in quaternions (used in the static transform)



```
1 clear all
2 close all
3
4 theta0 = -80.234; %orientamento iniziale (rotazione attorno a z)
5 q.x = 0; %non considero la rotazione attorno a x
6 q.y = 0; %non considero la rotazione attorno a y
7
8 theta0rad = theta0/180*pi; %in radianti
9
10 %% applico la definizione di quaternione
11
12 q.z = sin(theta0rad/2)
13 q.w = cos(theta0rad/2)
```



Name	Value
q	1x1 struct
theta0	-80.2340
theta0rad	-1.4003

q =

struct with fields:

x: 0  
y: 0  
z: -0.6444  
w: 0.7647

Finally with the command `roslaunch first_project baseline_estimation.launch`, it is possible to see how we compute the gear ratio and apparent baseline from `/scout_odom` and the cost function used to estimate the baseline from `/gt_pose`.