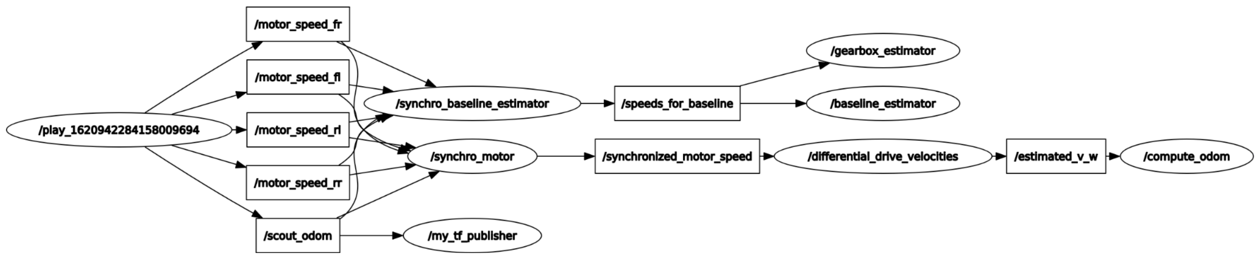
<u>1 - Student ID ("Person Code"), name and surname of all team members</u>

10564189 Massimiliano Manenti
10559982 Riccardo Monaco
10609776 Alberto Ortolan

<u>2 - Small description of the files inside the archive</u>

Before describing the files, a graph of the nodes could be useful to be seen:



We have divided the files in groups:

- file .cpp
- file .srv
- file .launch
- file .cfg

- <u>FILE .CPP</u>

[SYNCHRO_BASELINE_ESTIMATOR]
This file is used only to prepare a message that contains the four speeds synchronized (using a message filter) with the linear velocity (value of the field *twist.twist.linear.x*) and the angular velocity (value of *twist.twist.angular.z)* read on */scout_odom*, topic of the bag. In order to do this, it creates a node (*synchro_baseline_estimator*) and, after the synchronization, it sends a message of type Speeds.msg (custom message) to a topic called */speeds_for_baseline*. In this case, it is not necessary to keep track of the time instants.

[BASELINE_ESTIMATOR]
This file creates a node (*baseline_estimator*) which reads the message of type Speeds.msg on */speeds_for_baseline* and computes the baseline.
The computation is made in the following way: every time the callback starts, if w_z is different from 0, it takes the mean values on the left side (v_l) and on the right side (v_r) and it computes a temporary value for the baseline (at the current iteration). Finally, it updates the mean value of

all the temporary baseline previously computed. The results, at each iteration, is printed on screen, and the correct value is the last one obtained. Note that this file takes as a parameter (from the launch file) the gearbox ratio estimated using the files synchro_baseline_estimator.cpp and gearbox_estimator.cpp.

[GEARBOX_ESTIMATOR]
This file creates a node (*gearbox_estimator*) which reads the message of type Speeds.msg from /*speeds_for_baseline* and computes the gearbox ratio.
The computation is made through a formula that we got manually, starting from the usual equations of the skid steering vehicle. Then the procedure is the same as for *baseline_estimator.cpp* so that, after the bag stops, a mean value for the gearbox ratio is obtained.

[SYNCHRO_MOTOR]
This file creates a node (*synchro_motor*) which takes the four speeds from the bag (in particular from the four topics */motor_speed_..*), synchronizes them and publishes them on the topic */synchronized_motor_speed* along with a header, containing also the time instant of the synchronization. Since the four speeds are synchronized, their stamp.sec (and stamp.nsec) field is supposed to be the same for all of them, so we just picked one.

[DIFFERENTIAL_DRIVE_VELOCITIES]
This file creates a node (*differential_drive_velocities*) which takes the angular velocities (from */synchronized_motor_speed*) of the 4 motors. At every time instant, the mean values of the right and left side are computed (v_l and v_r), and used to compute v and w.
A message of type TwistStamped is assembled and published on */estimated_v_w*.

[TF_PUBLISHER]
This file creates a node (*tf_publisher)* which simply defines a transformation framework which relates "odom" (static framework) with "base_link" (framework moving together with the manufacturer odometry). The purpose of this is to visualize on Rviz (when all file are running) both the "base_link", which follows the trajectory of the manufacturer, and the "estimated_baselink", which follows the estimated trajectory, and compare them. "estimated_baselink" is obtained in the file "compute_odom.cpp".

[COMPUTE_ODOM]
This file computes the odometry, starting from the messages published on */estimated_v_w*.
Then, it publishes a message of type nav_msgs/Odometry, called *output*, on */final_odometry* and broadcasts a TF called *transformStamped*.
Moreover:
- provides two services, */reset_pose_to_origin* which resets the pose to (0,0) without changing the orientation, and */reset_pose_to* which resets the pose to a generic pose described by three ros parameters
- provides the possibility to select the integration method (Euler/Runge-Kutta), even while the

bag is running. This is possible thanks to dynamic reconfigure, with the parameter *E_RK*.
- publishes a custom message on */final_odometry_and_method*, of type OdometryAndMethod.msg

- FILE .srv

[POSE]
It has three values:
float64 x
float64 y
float64 theta
They are needed for the service */reset_pose_to* in the file *compute_odom.cpp*. The service has as a request a Pose type, there the user can set the desired pose.

[NULL]
It is empty but it is needed for the service */reset_pose_to_origin* in the file *compute_odom.cpp*. The service has as a request a Null type, in this case the file is empty because we don't need to pass parameters to the service (since it resets the pose to the origin, which is always (0,0)).

- FILE .launch

[PROJECT_LAUNCHER]
This file is used to:
-define five ros parameters (more details in the parameters section)
-start four nodes:
    -tf_publisher
    -synchro_motor
    -differential_drive_velocities
    -compute_odom (only one opened in a new terminal)

The launcher will NOT start the nodes useful for the estimation of the baseline and the gearbox ratio. To replicate the process of parameters estimation please refer to the paragraph "6 description of how to start/use the nodes".

- FILE .cfg

[DYNAMIC_RECONFIGURE]
This file is used to define a dynamic parameter (E_RK), through enumeration.
Its value can be changed between 0 and 1, using the terminal. (detail on the command in the section "how to use nodes")

----

3 - Name and meaning of the ROS parameters

[Inside the launch file]
Five parameters are defined inside the launch file:

- estimated_baseline: contains the value of the estimated baseline, obtained through the files *synchro_baseline_estimator.cpp* and *baseline_estimator.cpp* . The value is 1.004501
- estimated_gearbox: contains the value of the estimated gearbox, obtained through *synchro_baseline_estimator.cpp* and *gearbox_estimator.cpp*. The value is 38.461319
- initial_pose_x
- initial_pose_y
- initial_pose_theta

The last 3 parameters contain the values used for the initialization of the pose, helpful in order to start the computation of the odometry. Their values are 0, 0, 0.
Using the services, these values can be changed.

[Inside the dynamic_reconfig file]
One parameter is defined inside this file:
-E_RK
It is possible to modify this parameter (0 or 1, corresponding to Euler or Runge-Kutta) writing on the terminal:

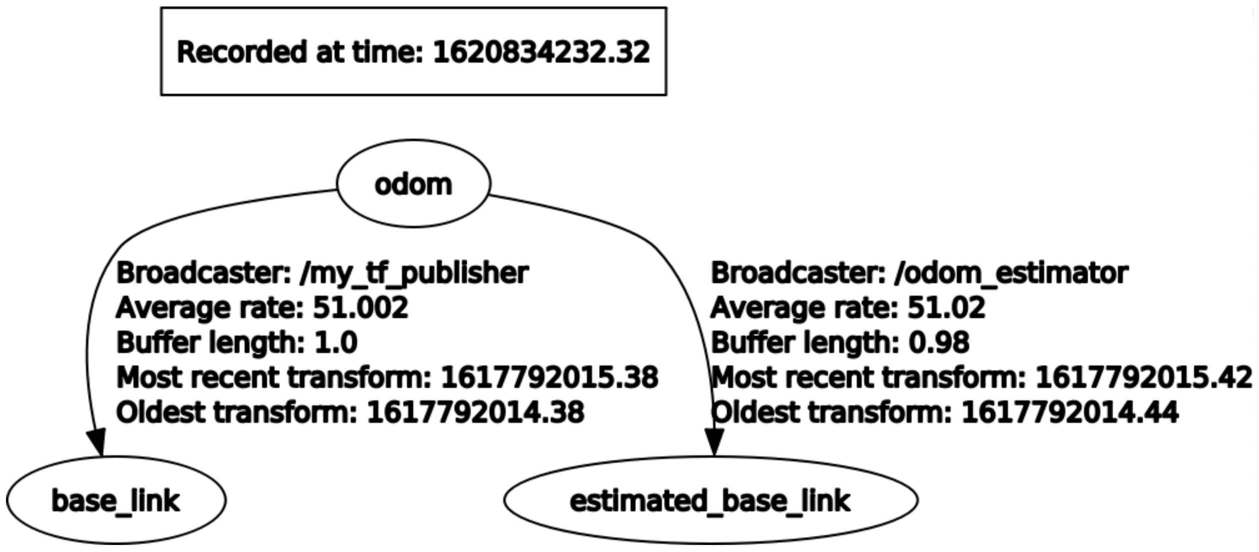
```
$  rosrun dynamic_reconfigure dynparam set /compute_odom E_RK value
```

where value can be either 0 or 1.

Inside the *compute_odom.cpp* file, an "if" switches the integration method depending on the value of this parameter.

----

4 - Structure of the TF tree


Recorded at time: 1620834232.32
odom
Broadcaster: /my_tf_publisher
Average rate: 51.002
Buffer length: 1.0
Most recent transform: 1617792015.38
Oldest transform: 1617792014.38
Broadcaster: /odom_estimator
Average rate: 51.02
Buffer length: 0.98
Most recent transform: 1617792015.42
Oldest transform: 1617792014.44
base_link
estimated_base_link

“odom” is the father frame. Linked to it there is the “base_link” frame (which represents the odometry of the manufacturer) and “estimated_base_link” (which represents the odometry computed by us).

<u>5 - Structure of the custom message</u>

[FourSpeeds.msg]
This custom message contains a header and four float64.
It is defined so that the file *synchro_motor.cpp* can publish, on the topic */synchronized_motor_speed*, a message containing the synchronized speeds of the four wheels of the robot. Inside the header, the field stamp is filled with the time of synchronization (sec e nsec) and the field seq is inherited from the topics */motor_speed*. Moreover the field *frame_id* of the header is inherited from */scout_odom*.

[Speeds.msg]
This custom message contains six float64.
Its only purpose is to contain, in one single message, the four speeds of the motor and the v_x and ome_z read on the topic */scout_odom* and synchronized thanks to *synchro_baseline_estimator.cpp*, so that the baseline could be estimated properly in *baseline_estimator.cpp* (and same for *gearbox_estimator.cpp*). This type of message is read on */speeds_for_baseline*. Since the time instants are not needed for this step, this message does not contain a header.

[OdometryAndMethod]
This custom message contains a field *odom* of tipe nav_msgs/Odometry, and a field *method* of

type std_msgs/String.
It is defined so that a message, containing both the odometry computed and the integration method, could be published on */final_odometry_and_method*. This operation is provided by *compute_odom.cpp* .

----

6 - Description of how to start/use the nodes

The package contains
- a cfg folder
- a launch folder
- a msg folder
- a src folder
- a srv folder
- CMakelist.txt and package.xml

In order to run the nodes, write
```
$ roslaunch robotics_hw1 project_launcher.launch
```
This will open 4 nodes
-tf_publisher
-synchro_motor
-differential_drive_velocities
-compute_odom
The last of them will also appear on screen.

In order to visualize the frames the standard procedure can be followed (rviz on terminal, add TF, run bag, select the fixed frame).
odom is the father frame. Linked to it there is the *base_link* frame (which represents the odometry of the manufacturer) and *estimated_base_link* (which represents the odometry computed by us).

In order to change the value of the dynamic parameter *E_RK*, type on the terminal:
```
$ rosrun dynamic_reconfigure dynparam set /compute_odom E_RK value
```
(where value has to be 0 for Euler, 1 for RK)

We have developed two services with the aim of resetting the odometry. One service resets the pose to a desired one, the other to the origin.
In order to use those services, one needs to write on the terminal:
```
$ rosservice call /reset_pose_to x y th
```
(where x, y th are floats representing the desired pose)
or

```
$ rosservice call /reset_pose_to_origin
```
(in order to set the pose to (0, 0) and current theta)
note that if you want to pass negative values, you need to write “--” before the parameters (x, y, th)

Here below is reported also how to use the files for the estimation of the following parameters:

-estimated_gearbox parameter: contains the value of the estimated gearbox, obtained through *synchro_baseline_estimator.cpp* and *gearbox_estimator.cpp*.
To get the estimation you can play the bag1.bag, pause it and then run:
```
$ rosrun robotics_hw1 synchro_baseline_estimator
```
and in a new terminal:
```
$ rosrun robotics_hw1 gearbox_estimator
```
then play the bag.
When the bag finishes, the last estimation of the gearbox should be 38.461319.

-estimated_baseline parameter: contains the value of the estimated baseline, obtained through the file *synchro_baseline_estimator.cpp* and *baseline_estimator.cpp* .
To get the estimation you can play the bag1.bag, pause it and launch the launcher:
```
$ roslaunch robotics_hw1 project_launcher.launch
```
we need the launcher because the parameter of the gearbox is defined there, then run:
```
$ rosrun robotics_hw1 synchro_baseline_estimator
```
and in a new terminal:
```
$ rosrun robotics_hw1 baseline_estimator
```
then play the bag.
When the bag finishes the last estimation of the baseline should be 1.004501.

----

7 - Info you think are important/interesting

ON THE ESTIMATION OF PARAMETERS:
The idea behind the estimation of the gearbox is:
- we synchronize the useful speeds coming from */scout_odom* and */motor_speed_(f/r)_(l/r)*.
  v_fl = msg->speed_fl/60*2*3.14*radious;
- we compute the auxiliary mean left and right velocity:
  v_l = (v_fl + v_rl)/2;   v_r = (v_fr + v_rr)/2;
  (those are auxiliary since they don’t represent a real velocity of the center of the wheel, since the gearbox radius scaling factor isn’t taken into account)
  (v_fl = msg->speed_fl/60*2*3.14*radious;
  is used instead of

v_fl = msg->speed_fl/60*2*3.14*radious/gearbox_ratio;)
- we estimate the baseline for that single time instant:
  temporary_gearbox_ratio = (v_r - v_l)/2/v_xg; (v_xg is the linear velocity of the robot coming from */scout_odom*)
  (this equation has been found with a system of equations of the robot kinematics)
- we find an average value for all temporary_gearbox_ratio found running the bag:
  est_gearbox_ratio = (est_gearbox_ratio*(k - 1) + temporary_gearbox_ratio)/k;
  (note that k is a counter, it represents the number of times that we have computed temporary_gearbox_ratio).

In order to estimate the baseline we use a similar process:
- we synchronize the useful speeds coming from */scout_odom* and */motor_speed_(f/r)_(l/r)*.
- we compute the mean left and right velocity of the wheels:
  v_l = (v_fl + v_rl)/2; v_r = (v_fr + v_rr)/2;
  (this time
  v_fl = msg->speed_fl/60*2*3.14*radious/gearbox_ratio;
  is used)
- as before, we compute several temporary baseline estimations:
  temporary_base = (v_l + v_r)/w_z; (w_z comes from */scout_odom*)
- and then we average them
  est_base = (est_base*(i - 1) + temporary_base)/i;

To replicate the process of parameters estimation please refer to the paragraph “6 description of how to start/use the nodes”.

ON THE PARAMETER RELATED TO THE INTEGRATION METHOD:
The parameter E_RK can assume only two values: 0 or 1.
It is used, inside the file compute_odom.cpp, in the following way:

-an internal variable (called i) takes the value of E_RK
-inside the formulas of x and y, the parameter “i” multiplies the terms that corrects the integration according to the Runge-Kutta formulas, so that, if i=0, the integration method is Euler, otherwise is RK.
This is useful in order to avoid inserting an “if”, and writing twice similar formulas.

ON THE FRAME IDS AND TIME STAMPS:
All the custom messages (except for Speeds.msg) contains a header.
The first message to be published is of type FourSpeeds, which, at every time instant, contains in the field *stamp* the same values of the corresponding message of type MotorSpeed.
After that, this value is continuously passed through the other messages, until it reaches the last one. In this way the Ts (delta time) used in the integration is the original interval between two measurements of the speeds and also when using rviz the movements of the two reference frame should be synchronous.
The same thing is done for the field *frame_id*, except that the original *frame_id* is taken from

/scout_odom. (we have done it in order to have the estimated pose and velocity expressed wrt the same “odom” reference frame and also in order to have only one father reference frame, the same of the MotorSpeed messages)

ON THE FILES GEARBOX_ESTIMATOR.CPP AND BASELINE_ESTIMATOR.CPP
The reason why the two files are separated is the following:
At the beginning, we used, as value for the gearbox ratio, the one reported in the official website. Therefore, only the files *synchro_baseline_estimator.cpp* and *baseline_estimator.cpp* were created. Then, we decided to estimate the gearbox ratio from the data, and we thought that it would have been easier to simply add another file instead of computing also this estimation in the file *baseline_estimator.cpp* . In fact, the baseline estimation requires a value of the gearbox ratio for it to run.
After we got a more accurate value for the gearbox ratio, we also adjusted the value of the estimated baseline.