Autonomous Vehicle Competition, Advanced Robotics – CSCI 5302

Vibhor Mishra, Raghunath Reddy and R P Sharat

***Abstract*— As a part of the class Advanced Robotics which involves Automatic Vehicle Competition, we made autonomous bot which can move along the corridors and stop in the presence of a stop sign and as a part of the challenges we implemented a rolling ball avoider, SLAM, sparse Map. Keywords: autonomous, ROS, SLAM, Odriod, OCam, Polulu**

I. Introduction

The components provided to achieve the tasks are Odroid-XU4 (main CPU), Pololu Driver, IR Sensors, O-Cam, Phidgets IMU, 1/10th Scale monster truck with DC Motor and servo motor, Lithium Polymer battery. The Autonomous Vehicle competition includes a closed loop circuit timed race which includes a stop sign during the race where the bot is required to stop and then continue with the race. In the next stage, we perform SLAM/Sparse Map using ORB_SLAM_2, which will be displayed on the screen using a HDMI dongle on the Odroid present on the bot. In the next step, we would implement an obstacle avoider / rolling ball avoider using data from camera and IR sensor placed in the front of the camera. Throughout all the challenges and the race, we used data from IR sensor and camera, as an important factor for making decisions. We used our own control loop to control the motion of the bot i.e. IR Sensor, D.C. Motor.

II. Hardware and Software used

The various hardware and software modules used in the bot are as follows.

A. Mechanical - Hardware

In the given chassis of the 1/10 AMP 2WD Monster truck [1], we had a very good damper system and a very good protection chassis. Keeping the safety of the components into account, the components prone to physical contact are protected with cases. The bot's bottom extrusion was extended to protect the IR sensor and the camera in the front, changed the bot in such a way so that the base plate on which the components are placed have minimal or no stress concentration. Because of low speeds, there was no requirement of sturdiness. Hence an acrylic sheet was selected as a base plate. Initially all the components were fixed on the baseplate with Velcro when the positions of all the components were not fixed but was later fixed properly. The amount of wires and cables on the baseplate also was taken into consideration.

B. Electronics - Hardware

With some components like the ESC (Electronic speed control) having little or no documentation, though it was a bit difficult to figure out the electronic connections in the beginning, it was later figured out and was easily controllable. Because of usage of Robot operating system, the hardware was chosen such that the ROS software packages of each of the hardware is readily available and can be easily installed. The following circuitry for the bot was made.

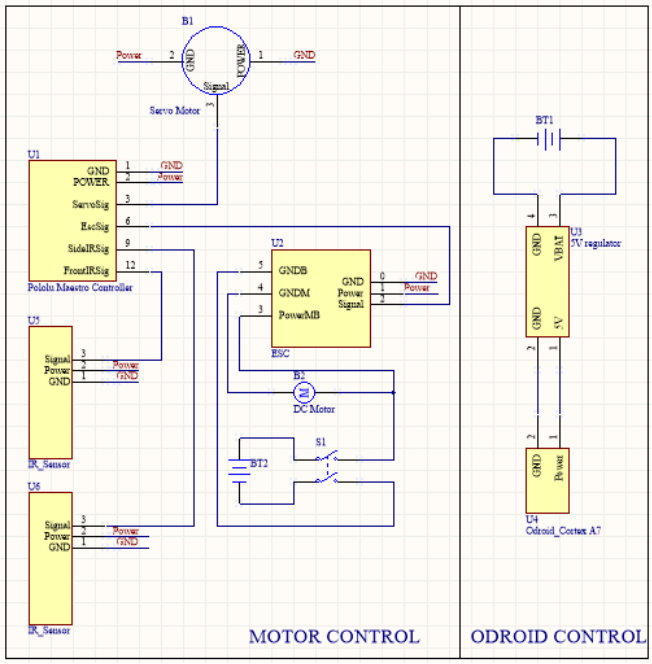


Figure 1: Schematic of the wiring on the bot.

The hardware described below explains the usage of each component.

- Battery:
  The initial given battery was a 1800mAh LiPo 7.2V rated battery. But the capacity wasn't enough to run the given DC Motor and to run the Cortex A7 oDROID board. Hence another 3000mAh battery was added to support the motor separately. With this configuration, there were no current deficiency issues. The battery stays charged for half hour which is more than enough for good time of testing. But it takes around 2.5 hours to charge the battery. And there is no battery level indicator.

- IR-Sensor:
  The IR Sensors were put up on the car in two positions to identify the distance from the front and distance from the side. These IR sensors have a range of 100 cm to 500 cm. Because of this the sensors are to be placed on the edge of the bot to reduce the distance from the wall as much as possible. The IR sensors are very much dependent on the reflectivity of a surface and hence sufficient care had to be taken to avoid that while testing.

- DC-Motor:
  The DC Motor being used is the stock DC Motor from the chassis given. It has a high inrush current of around 20A and hence a separate battery had to be used in the system as the ODroid was getting reset when the Motor had to just turn on. The Motor's rating was found out after testing to be 2V to 8V and the starting voltage should be greater than 5V.
- Servo-Motor:
  The servo motor used is the stock servo motor given along with the chassis.
- Maestro Pololu 18 Channel Servo motor controller:
  The Maestro Pololu controller is used to control the DC motor and the servo motor. The front and side IR sensor data are also being taken from through this board. The IR sensors are connected on the channels 2 and 4, the servo Motor on channel 6 and the DC motor on channel 8. A few things like pulse width control is don't through the Maestro controller user interface.
- ESC:
  The ESC was used to control the DC motor and hence the thrust given to the bot. With two MOSFET's in parallel and timer IC which would help to control the Ton time of the ESC and hence control the motor. With a little bit of testing, it was figured out that 1.2ms was required to turn on the motor. The MOSFET's provide a ground to the motor whenever the ESC is provided with a signal.
- O-Cam:
  The Monocular camera that is given to be used is used mainly as there are ROS packages available for the same. The camera has a high resolution and helps in detecting contours for stop sign detection.
- ODROID:
  The ODROID is the main brains of the car. It is connected to a separate battery and all the components are controlled by the Odroid. It is connected to the Pololu and the camera through USB and commands are sent through the same.

C. Software Packages

The following packages are being used with the functionality as given.

- Ros_pololu_servo:
  Adapting from the "geni-lab/ros_pololu_servo[2]", this package provides a ROS is installed on the ODROID to help for the control of the bot's DC motor, Servo motor and to take in the IR sensor values. Here we publish the data on the topic "/pololu/command". Once the scripts are written and changes are made in the .yaml file according to what we need and then the launch files present help to start the publishers, once the installation is done. It is good to first install the user interface and then the package installation as it helps to install certain drivers which might be required.
- libuvc_camera:
  This package provides a ROS interface for digital cameras meeting the USB Video Class standard (UVC). Adapting from the "ktossell/libuvc[3]", this package is mainly used to take in the image data for stop sign detection and for contour detection required to make the "jump". Here we are publishing the data on the topic "image_raw". This data is taken and filtered to detect for contours of hexagon or circle based on our required application of stop sign detection or circle detection. During installation, make sure the user is added to the dialer group else the camera will not be detected.
- ORB_SLAM2:
  ORB-SLAM2 is a real-time SLAM library for **Monocular**, **Stereo** and **RGB-D** cameras that computes the camera trajectory and a sparse 2D/3D reconstruction. It can detect loops and re-localize the camera in real time. Adapting from the "raulmur/ORB_SLAM2[4]", this package helps in building our own sparse map from the scratch by detecting features on the image frame and building the map on the detected features. Just launching the launch file, opens a window and the package starts building up the sparse map from that position.

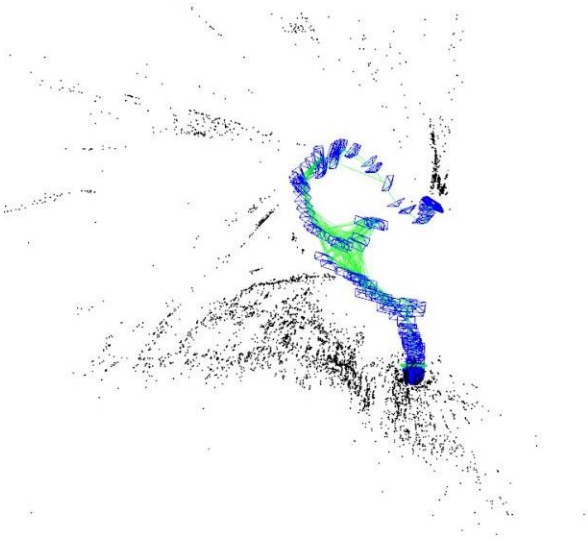


Figure 2: The sparse map drawn of the Norlin Library's Ground floor

The sparse map shown isn't very clear as the camera had a large amount of viewing area.

- robot_pose_ekf:
  The Robot Pose EKF package [5][6] is used to estimate the 3D pose of a robot, based on (partial) pose measurements coming from different sources. It uses an extended Kalman filter with a 6D model (3D position and 3D orientation) to combine measurements from wheel odometry, IMU sensor and visual odometry. The basic idea is to offer loosely coupled integration with different sensors, where sensor signals are received as ROS messages.

The sensor signals received are that of IMU and visual odometry. The package publishes on the topic “robot_pose_ekf/odom_combined” and subscribes from various topics based on the data required. Here the usage of “odom” topic is done as IMU and visual odometry is not used in the developed algorithm.

III. METHODOLOGY

The race is divided into different parts and hence different methodologies are used for different scenarios. First is the main event, second is the rolling ball avoider and the third is the ramp. As the race has different scenarios on its own, many of them are integrated into the same algorithm.

A. The Race

The race objective is for the bot to follow the wall and take a left or right depending on the turn there is to be taken and stop at the stop sign, whenever it comes up.

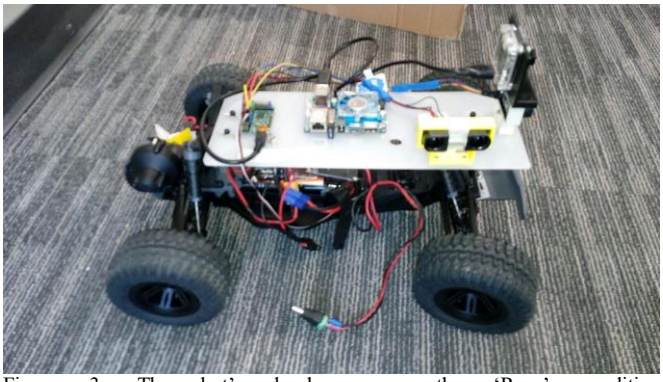
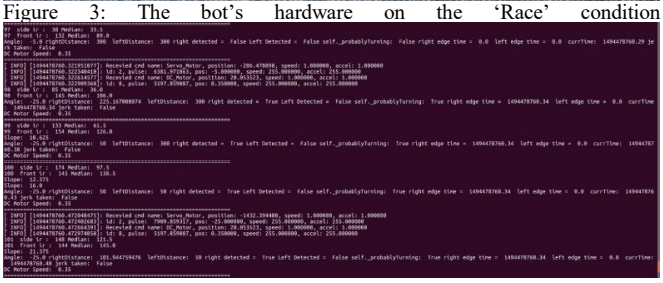


Figure 3: The bot’s hardware on the ‘Race’ condition



Our hardware setup for this is as follows. The IR sensors are put perpendicular to each other. One pointing towards the front, and the second pointing towards the right wall. The camera is also put in the front. All other PC Boards are put on the baseplate.

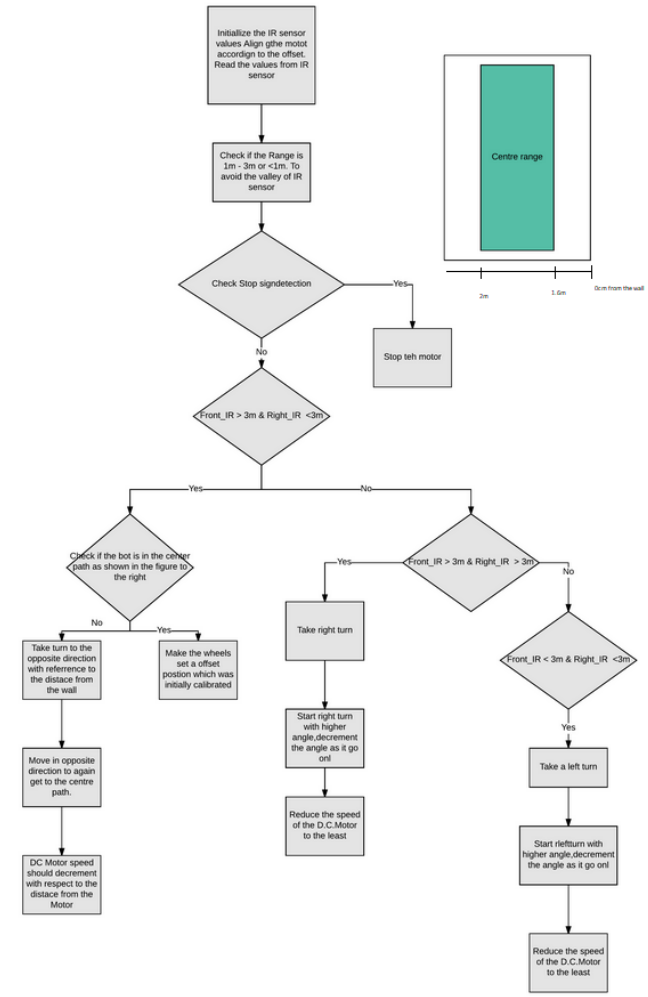


Figure 4: The flow chart of the control module

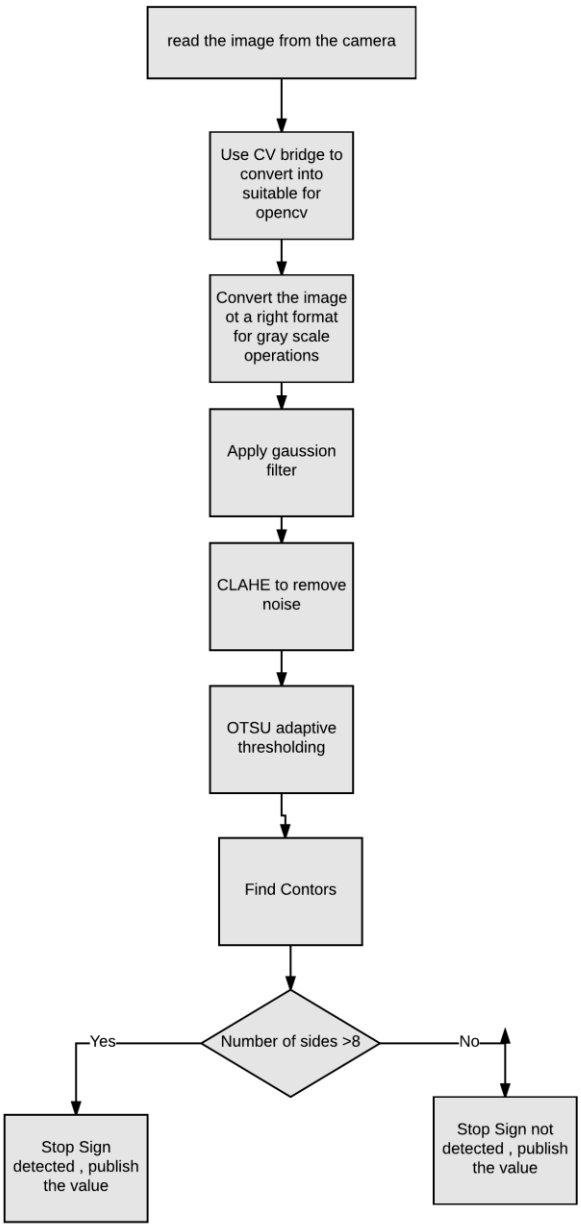


Figure 4: The flowchart of the stop sign detection.

The algorithm of the bot movement is as follows. First the straight paths are divided into 3 areas taking into the fact that the path will not be wider than 300cm. The first area is between the wall and 160cm from the wall and the second part is 160cm to 200cm from the wall. When the bot is started, the DC motor is turned off till the servo motor is calibrated which is time based of 2.5 seconds. During this calibration time, the IR sensor is also fed a dummy value to set up the median filter to filter out the noise from the IR sensor values. Once the DC motor is turned on, two topics publish the data. One is the IR sensor along with time stamp and other is the detected object along with time stamp. The IR sensor sends the value of the front and side IR and the image sends a Boolean value of image detected or not. The image is looked for 5 times to make sure and then the image detected flag is set high and sent to the motor controller. The IR sensor value is also sent along with the timestamp so as to make sure we are not looking at the same value twice. Once the data is received by the motor controller, if image detected is true, it gives maximum value to the DC motor and because of that and the ack EMF, the motor is forced to a halt. It keeps checking till the flag becomes false. If image detected is false, it checks for the IR values. If the front IR sensor value is less than 300cm, the DC motor is put to a minimum possible speed and once the front IR reaches the value of 270cm, the side IR value is looked for. If the side IR, which is on the right side gives a value of less than 300cm, the bot knows it should turn to the left and the servo motor is given the max value to turn right. If the side IR detects a value more than 3000cm, maximum value is given to the servo motor to take a turn towards right. One the bot has taken a turn, if the bot is in the first area (i.e. less than 160cm from the wall), the servo is given a slight left side command till it reaches the second area and when the side IR sees that it is inside the range of the second area, the bot is given a slight jerk for 50ms to turn to the opposite side and then is set to move in a straight line irrespective of anything as long as it is in the second area. Along with all this the map is to be formed using the ORB_SLAM2 package and be displayed on the monitor.

B. Ramp

The objective of this part is to make the car go over a slide and jump, being in control even in the air and land properly on the other side of the slide.

The hardware setup for this is as follows. The bot is equipped with the camera in the front. The IR sensors are not used in this method so they are on the bot with no functionality. The PC Boards are secured onto the baseplate.

The libuvc package detects for a circle. When the circle is detected, in the coordinates of the center of the circle the X axis value is published onto the topic “ObjectDetected”. The motor controller which is subscribing from this topic, receives the coordinate value and compares it with the center of the frame value. The resolution of the image being 1280x980, if the coordinate value is less than 460, that implies that image is on the left of the frame and the bot should turn towards left. Then the DC motor is given a slight angle to turn to left based on the difference of the center of the circle and the center of the frame. But then again if the coordinate value is more than 500, that implies that image is on the right of the frame and the bot should turn towards right. Then the DC motor is given a slight angle to turn to left based on the difference of the center of the circle and the center of the frame. Once the center of the circle comes into the buffer region of 460 to 500, the servo is made straight and the DC motor is given the highest speed and the servo motor is locked till after 5 seconds of the image not visible in the frame and then the power to the DC motor is cut.

C. Rolling Ball avoider

The main objective of this leg of the race is for the bot to be able to avoid an oncoming ball and to keep going while maintaining its path.

The hardware setup of this is to place the two IR sensors in the front. The Camera is not used for this and all the PC Boards will be on the base plate.

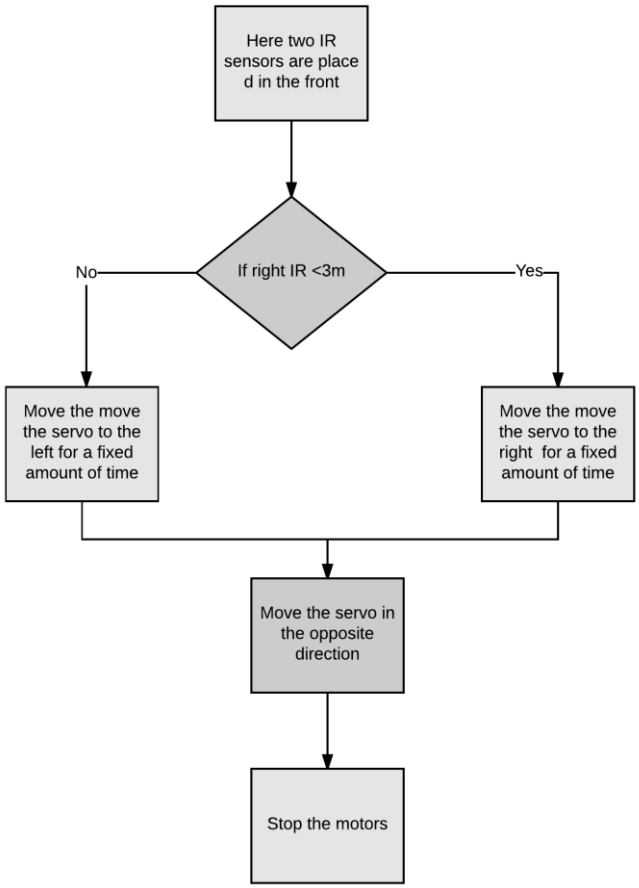


Figure 5: Flow chart of the rolling ball avoider algorithm.

The algorithm for the rolling ball avoider is something like this. The two IR sensors placed on the front acts as a basic gesture sensor. The two IR sensor values are published along with their respective time stamp. The right IR sensor and the left IR sensor values are compared, within 2ms of their time stamp. If the right IR sensor value is less than 300cm and the left IR sensor value, it means that the ball is coming from the right and the bot needs to turn right. The servo is then given a sharp right turn for 2.5 seconds, then sharp left for 2.5 seconds. But if the left IR sensor value is less than 300cm and the right IR sensor value, it means that the ball is coming from the left and the bot needs to turn left. The servo is then given a sharp left turn for 2.5 seconds, then sharp right for 2.5 seconds. After this the DC motor is made to run for 1 second and then its stopped.

IV. Results

The car was run in various modes. The various terminal logs are as follows.

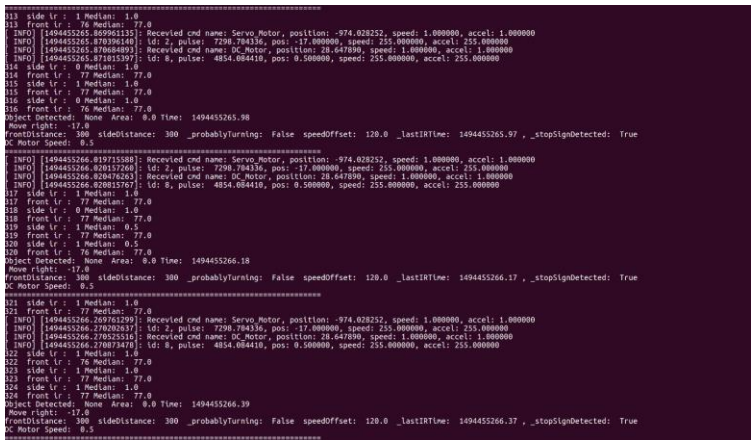


Figure 6: The log file of a normal run scenario

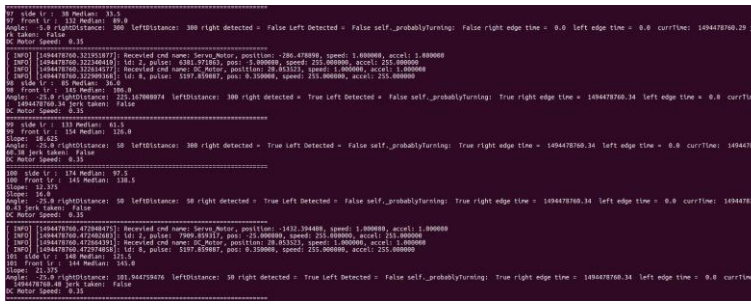


Figure 7: The log file of a ball avoider scenario

V. Summary

The given components helped us to make the bot for the requirements. The bot has the objective to move in the specified course, stop at a stop sign, and form a sparse map. That was achieved with the Bot having a hardware required to run on ROS which was the main subject of the course. The given electronics, some had some problems like the regularity of values in IR sensor, irregular performance of the ESC. But at the end after a lot of filtering and a lot of testing, they were made to work well. The Bot following the wall was achieved with the help of the side the front IR sensor and Pololu controller's packages. The Stop sign was achieved with the help of libuvc packages. The ball avoider was done using two IR sensors in the front.

VI. Acknowledgment

We would like to thank Professor Christoffer Heckman for the wonderful opportunity to help us make this bot and would also like to thank the University of Colorado Boulder for this great opportunity.

VII. Appendix

Attached along with the report in the end.

VIII. References

[1] The monster truck chassis. Components and parts: https://www.horizonhobby.com/product/cars-and-trucks/cars-and-trucks-14524--1/electric-cars-and-trucks/1-10-amp-mt-2wd-monster-truck-rtr--black-green-p-ecx03028t2

[2] Geni-labs, ROS Pololu servo package git repository: https://github.com/geni-lab/ros_pololu_servo

[3] Libuvc package. A recommendation in the ROS_wiki:https://github.com/ktossell/libuvc
[4] Raul Mur-Artal, Juan D. Tardos, J. M. M. Montiel and Dorian Galvez-Lopez, The ORB slam package's git repository: https://github.com/raulmur/ORB_SLAM2
[5] Wim Meeussen, the git repository of the robot pose ekf: https://github.com/ros-planning/navigation.git
[6] Wim Meeussen, the ROS wiki page of robot_pose_ekf: http://wiki.ros.org/robot_pose_ekf