UNIVERSITÀ
DI SIENA
1240

Dipartimento di Ingegneria
dell'Informazione e Scienze Matematiche

Arduino-based mobile robots controlling in ROS

Marullo Simone

Report on a Human-Centered Robotics project
Academic Year 2019-2020

Contents

Chapter 1

Abstract

In this project I developed a simple navigation system for two-wheel mobile robots and acquired experience with marker recognition, camera-based localization and ROS programming. I used an Arduino-based robot whose only logic is related to the execution of speed commands from a computer through a Bluetooth module. The computer is able to localize the robot in an environment using either its camera (assuming that the robot is tagged with a marker), either exploiting the Vicon localization system.

Once the user decides a navigation goal, the software computes the rotation needed to align the robot axis with the displacement vector and sends the command to the robot. After the alignment phase, the computer asks the robot to perform translation until the target is reached.

Chapter 2

Preliminary stage: camera calibration

In order to perform camera measurements, some prior knowledge of the camera system is required. I adopted the pinhole camera model with RadTan distortion for the gimbal camera of the drone and estimated the related parameters by the means of camera calibration.

2.1 Pinhole camera model

The pinhole camera model is the most widely used camera model in computer vision; it is rather simple but generally quite effective as a first order approximation of the mapping from a 3D scene to a 2D image. It models the mathematical relationship between the coordinates of a point in a 3D space and its projection onto the image plane of an ideal pinhole camera, which is a camera system where the aperture is a point and no lens are involved. The following equation explains the model:

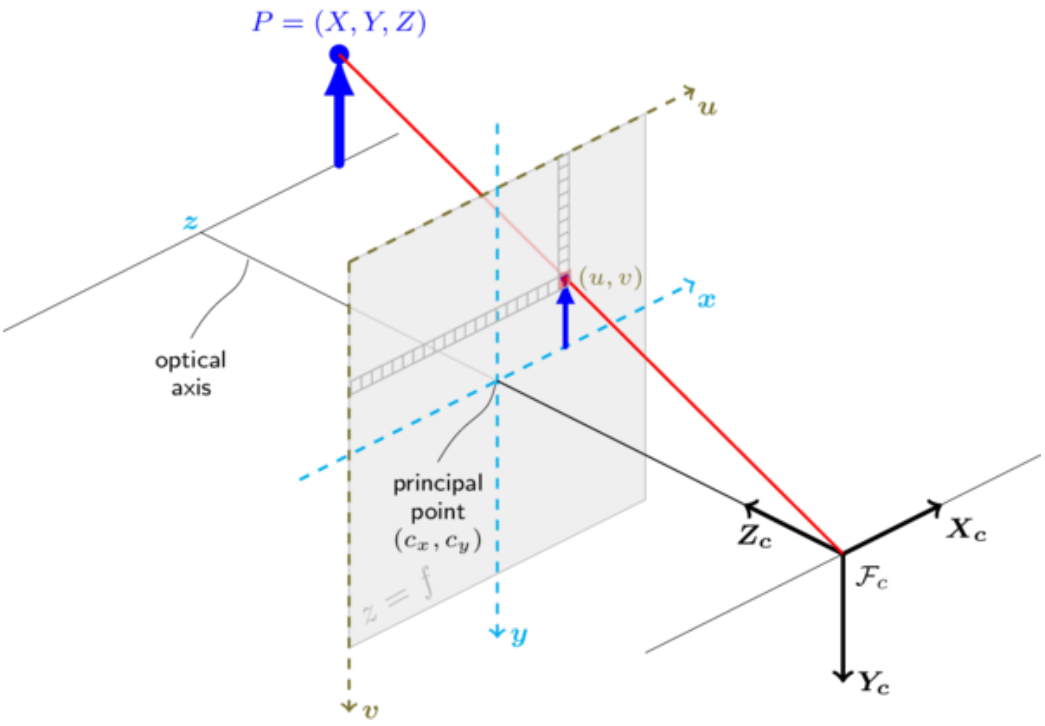


Figure 2.1: Geometric view of the pinhole camera model

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = A \left[R|t\right] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

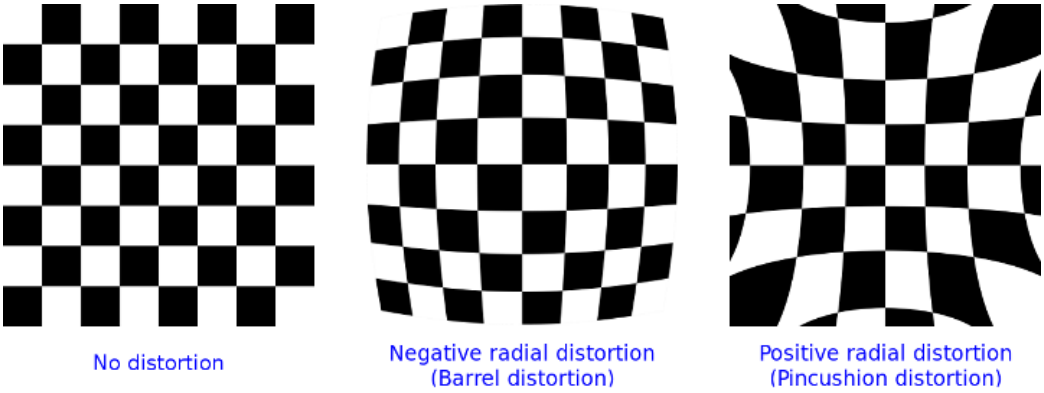
where:

- (X, Y, Z) are the coordinates of a point in the 3D space, while (u, v) are the coordinates of its projection
- s is a scale factor
- $\left[R|t\right]$ is the rototranslation matrix (or extrinsic parameters matrix), which describes camera motion around a static scene
- A is the camera matrix (or intrinsic parameters matrix, so-called because it does not depend on the scene depicted)

– (c_x, c_y): optical center (typically image center) expressed in pixels coordinates.

– f_x and f_y are the focal lengths expressed in pixel units.

Figure 2.2: Typical distortion produced by real lenses



Actually, real lenses produce two main kinds of significant distortion: radial (the so-called "fish-eye" effect) and tangential (caused by lenses not perfectly parallel to the image plane). Hence, the pinhole model can be extended to take into account these effects:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t$$

$$x' = x/z$$
$$y' = y/z$$
$$x'' = x'(1 + k_1 r^2 + k_2 r^4) + 2p_1 x' y' + p_2(r^2 + 2x'^2)$$
$$y'' = y'(1 + k_1 r^2 + k_2 r^4) + p_1(r^2 + 2y'^2) + 2p_2 x' y'$$
$$\text{where} \quad r^2 = x'^2 + y'^2$$
$$u = f_x x'' + c_x$$
$$v = f_y y'' + c_y$$

2.2 Calibration procedure

I used the `camera_calibration` ROS package (which is based on the OpenCV library) to perform the monocular camera calibration with a standard black-white chessboard target.
The process of calibration consists in estimating camera matrix and distortion coefficients. While a wide range of objects could be used to perform calibration, chessboards are a popular choice because they are simple to construct, and their planar grid structure defines many natural interest points in the image. The procedure takes as input 3D coordinates of object points and their corresponding 2D projections (which are easily achievable exploiting the known geometry of the chessboard) and in so doing is able to characterize the distortion produced by the specific camera system.

Chapter 3

Robot upgrade and characterization

The Arduino Robot is a very simple robot produced by Arduino. It comes with two processors, one on each of its two boards. The Motor Board controls the motors, and the Control Board reads sensors and decides how to operate. Each of the boards is a full Arduino board programmable using the Arduino IDE.

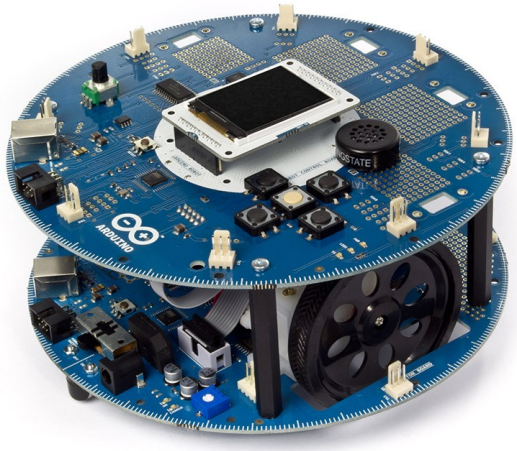


Figure 3.1: Arduino Robot

The robot is not equipped with any communication module, so I added a cheap Bluetooth module (RN-42, by Roving Networks) to provide a convenient communication channel with the computer. It is also very easy to integrate into the Arduino platform, since it is controllable through serial interface.

Figure 3.2: RN-42 Bluetooth module

The robot does not have encoders at the wheels or any other sensor useful to implement odometry and motion control onboard. Hence, it is not possible to tell the robot to execute a specific rotation or translation and the speed commands are very qualitative (e.g., writing some speed values at a certain time may produce a motion or not, because of the different charge levels of the battery while time passes by). Since it is quite difficult to model the behavior of such a system, I proceeded with an approximate empirical approach. I fixed a specific velocity (i.e., 40% of the maximum value) and performed a number of experiments both for rotational movement and translational one. Using the camera system to get meaurements, I collected pairs of command duration and obtained displacement (angle or length) and performed simple polynomial (quadratic) regression.

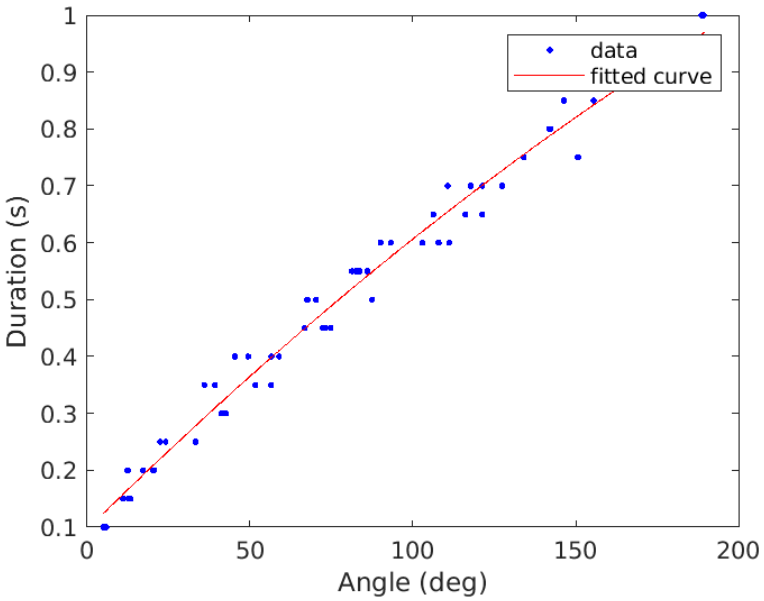


Figure 3.3: Regression for `rotate` commands

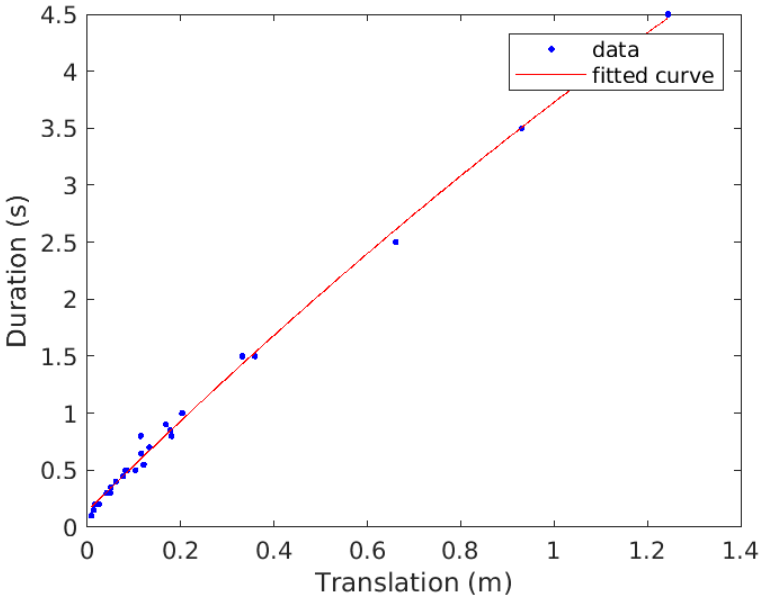


Figure 3.4: Regression for `go_straight` commands

Chapter 4

Software development

I decided to use ROS (Robot Operating System, originally developed by Stanford Artificial Intelligence Laboratory) as software environment, in order to become familiar with a widely popular framework for programming heterogeneous clusters of computers, smart devices, robots and autonomous vehicles. It features a graph architecture where processing takes place in nodes that are able to receive and post sensor data or control messages.

For marker recognition I used the `aruco_ros` library. In the implemented approach, two markers are used: one to be placed on the floor as a fixed reference system and one to be attached on top of the robot.

The control system is composed of the following parts:

- node `bluetooth_proxy.py`: it just subscribes a specific topic and forwards the command via Bluetooth to the robot
- node `aruco_tf2_broadcaster.py`: it is responsible of processing marker detections from the `aruco_ros` module and publishing the result coherently in the `tf` channel
- node `robot_positioning.py`: it exploits the information already available in the `tf` channel to compute the desired orientation `robot_steering` of the robot in order to reach the target. In particular, with $\vec{p}$ being the 2D position of the robot and $\vec{q}$ being the 2D navigation goal, the

desired angle with the x-axis of the reference frame is given by

$$\theta = \arctan \frac{(\vec{p} - \vec{q}) \cdot \hat{y}}{(\vec{p} - \vec{q}) \cdot \hat{x}}$$

- node `robot_controller.py`: it generates command in order to align the actual orientation the the desired one and finally translate to the target. The duration of the *rotate* and *go_straight* commands are decided according to the regression laws previously found; errors are subsequently adjusted until a given threshold (5 cm) is reached

The user can set navigation goal directly in the Rviz interface (using the tool 'Set 2D Navigation goal', which publishes the clicked point into the topic `/move_base_simple/goal`, that is subscribed by the nodes.

To simplify the launch of the system, the following launch files are created:

- `usb_cam_two_arucos.launch`, which specifies the use of a USB camera for marker recognition and the features of the printed markers
- `arduino_robot.launch`, which launches the control system with the Rviz interface

Also, a specific config file is created to control Rviz panels. The control system can be seen in action here.

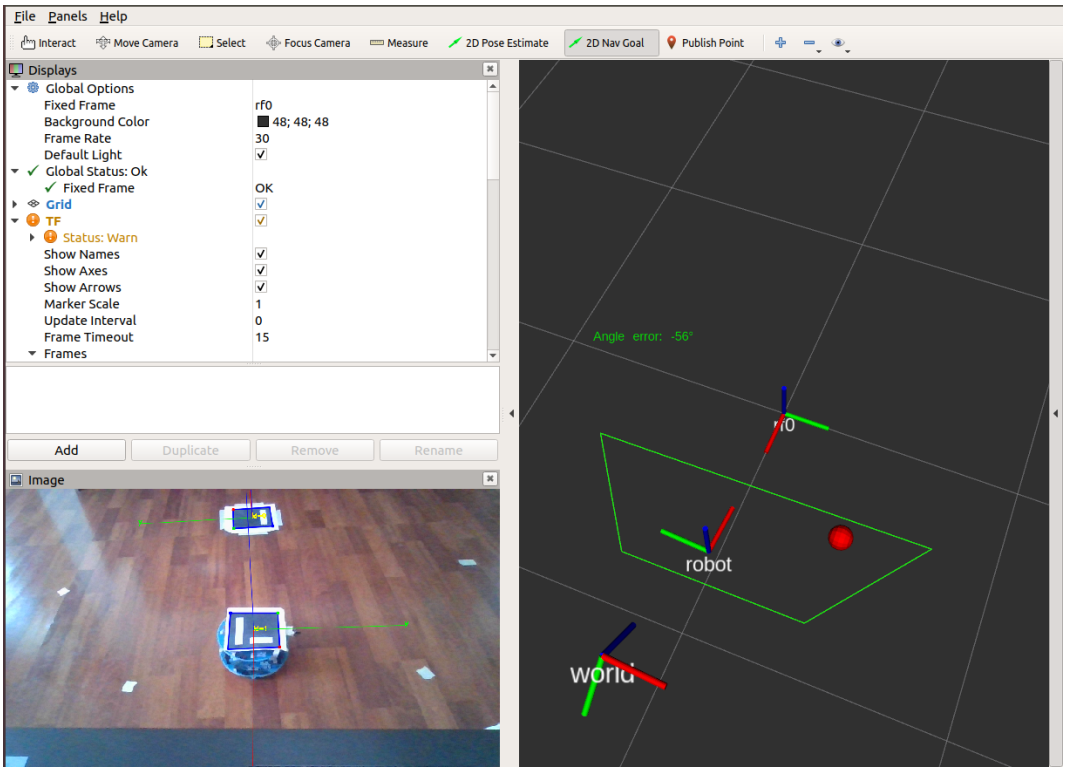


Figure 4.1: Rviz control interface: in the right panel, the trapezoidal polygon corresponds to the robot workspace (chosen by the user

. *The red marker corresponds to the robot target and was acquired by clicking on the grid. In the lower left panel it is shown the live stream from the camera with superimposed markers.*