Homework - Kinematics Calibration of a Differential drive wheelchair

Francesco Argentieri[*]

University of Trento
francesco.argentieri@studenti.unitn.it

Abstract

For a mobile robot, odometry calibration consists of the identification of a set of kinematic parameters that allow reconstructing the vehicle's relative position and orientation starting from the wheels' encoder measurements. The aim of the homework is to: estimate from the encoder and the camera data the kinematics parameters wheels radius "r_L" and "r_R" and wheelbase "b". Estimate the camera position with respect to the wheelchair reference point (the mid point between the wheels).

I. Introduction

The autonomous navigation is understood as the set of techniques by which it is a system able to move with a certain level of autonomy in a certain type of environment (land, air, underwater, space). The problems in the field of autonomous navigation are addressed first and foremost related to the localization of the system with respect to the environment, planning out their duties and motion control. An interesting class of systems covered by the study of autonomous navigation systems is that of the AGV (Autonomous Guided Vehicles). They are now widely used in industries, ports, airports, hospitals, etc, however, to measure the position and the attitude of a vehicle it is still a problem of substantial interest. The sensor fusion is the process that combines information from a number of different sources to provide a complete and robust description (measure) a set of variables of interest. The sensor fusion is of particular utility in any application where many measures have to be combined together, melted and optimized in order to obtain quality information and integrity suitable for the purpose of the application. The sensor fusion techniques are used in many industrial systems, military, monitoring, civilian surveillance, control processes and systems.

The problem of localization of autonomous vehicles, which in almost all cases, the individual transducers are found insufficient in setting up a comprehensive and robust localization system for autonomous navigation, requires the use of sensor fusion techniques for combining measurements from different types of transducers whose characteristics, if fused together, allow us to obtain a more reliable and accurate measure of the state of the system and the environment surrounding it. The sensor fusion techniques have important applications in the field of autonomous navigation, in which it is necessary to obtain a good estimate of the position measurement and alignment (pose) of a mobile robot. Incremental measuring methods or dead-reckoning, using encoders, gyroscopes, ultrasonic etc., have the considerable advantages of being self-contained within the robot, to be relatively simple to use and provide high refresh rate measure. On the other hand, since these measurement systems integrate related increments, the errors grow considerably increasing the integration time.

The document presented is based on the analysis of a real robot, developed and built at the MIRO - Measurements Instrumentations Robotics Lab at the University of Trento. It is equipped with two rotary incremental en-

[*]ID: 183892

coder keyed to the axes of the two wheels and a chamber pointing upwards for reading specially arranged on the ceiling marker. The two datasets have been collected steering the vehicle clockwise and counterclockwise.
The aim of the homework is to:

- Estimate from the encoder and the camera data the kinematics parameters wheels radius "r_R" and "r_L" and wheelbase "b".
- Estimate the camera position with respect to the wheelchair reference point (the mid point between the wheels).

II. Methods

i. Kinematic Model

For the analysis of wheelchair differential drive parameters is used a simplified model; the reference system, indicated by $RF1$, is placed in the odometric center of the robot. Consider that the robot moves in a reference system fixed to the $RF0$ envoiroment. The robot is subject to pure rolling, then we neglect slippage between wheel and ground. The angular velocities, indicated with ω_r and ω_l, are applied respectively to the right and left wheels in such a way that the components of the fixed body and the speed of the robot are related to the angu- lar velocity of the wheels according to the equation (1). The other two support wheels are considered passive. This schematization is observable in fig. 1.

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = C \begin{bmatrix} \omega_L \\ \omega_R \end{bmatrix} \tag{1}$$

The matrix $\mathbf{C} \in \mathbb{R}^{2x2}$ is defined as (2):

$$\begin{bmatrix} \frac{r_R}{2} & \frac{r_L}{2} \\ \frac{r_R}{b} & -\frac{r_L}{b} \end{bmatrix} \tag{2}$$

in which r_R and r_L are the radii of the right and left wheel, respectively. The odometry of a vehicle is usually implemented by discrete-time integration, such as (3):

$$\begin{cases} x_{k+1} = x_k + T\, v_k\, cos(\theta_k + T\,\omega_k/2) \\ y_{k+1} = y_k + T\, v_k\, sin(\theta_k + T\,\omega_k/2) \\ \theta_{k+1} = \theta_k + T\,\omega_k \end{cases} \tag{3}$$

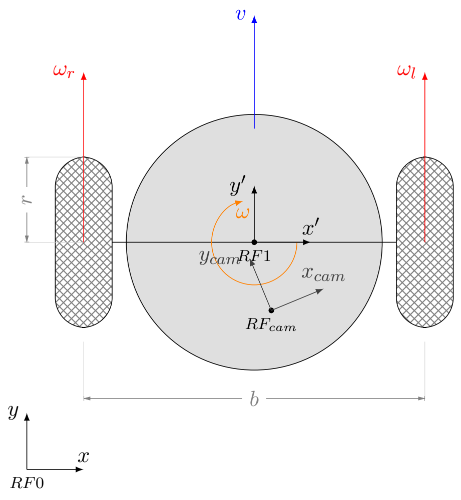


Figure 1: Robot kinematic model

Notice that low sampling frequency and high vehicle velocities can be significant sources of odometric error.

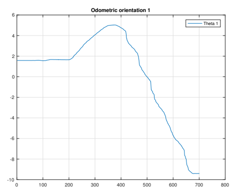
ii. Data Analysis

To achieve the objective of calibration of the geometric parameters of the robot four datasets were provided in each of which the information is collected from the camera and from the incremental encoder. In the data provided there were errors due to their registration, in fact, they are prompted to correct them, then a python script is created[1] which accepts the original files containing the data input; the first part of code is aviable at listing 1. In the first operation, it corrects the sign to the values of the left encoder because it has mounted inverted repetition to the right encoder. The second correction that concerns the robot's angle of orientation changes a sign since this must be considered as a negative sign. It may be noted that in one line the camera information and the other the odometer ticks have changed. Thus, at the same timestamp, in the second row the information is modified. So you chose to eliminate duplicate rows to preserve the last where both information camera pose or covariance and ticks has changed. The second operation was carried out with the second part of the code that can be consulted at

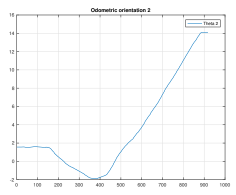
[1]the code is reported at the end of the report

listing 2, which confines the row information while preserving the last one. At the end of the operation we can observe that the number of data has been reduced respect the starting point, the result is available in table. We can observe in figure 3 the path registred by camera in plane xy, in figure 2 the orientation θ and in figure 4 the encoder trends.

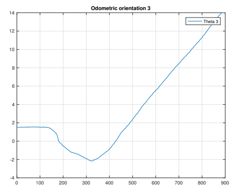
	Original nr. row	Shrink nr. row
dataset 1	1229	701
dataset 2	1617	918
dataset 3	1016	896
dataset 4	884	807



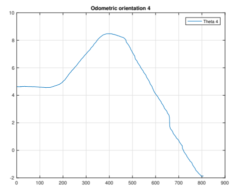
(a) *dataset 1.*



(b) *dataset 2.*

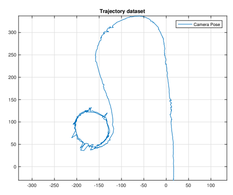


(c) *dataset 3.*

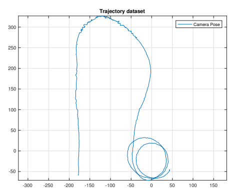


(d) *dataset 4.*

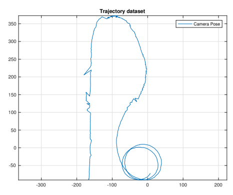
Figure 3: Odometry orientation



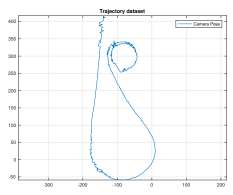
(a) *dataset 1.*



(b) *dataset 2.*

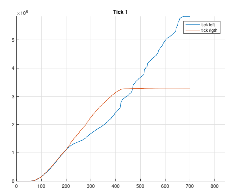


(c) *dataset 3.*

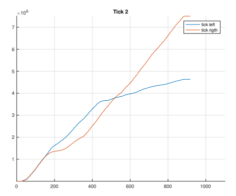


(d) *dataset 4.*

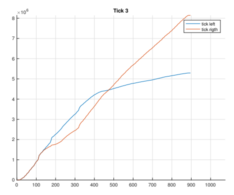
Figure 2: Path from camera



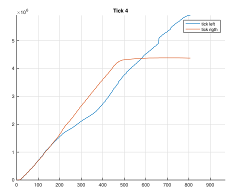
(a) *dataset 1.*



(b) *dataset 2.*



(c) *dataset 3.*



(d) *dataset 4.*

Figure 4: Encoder trend

iii. Calibration Techinque

To estimate the parameters of the robot expressed in the equation (2), namely, the wheel radius values indicated with "r_R" and "r_L", and the axle track as indicated "b", using the method described in the report [1]. Experiments of odometry calibration require measurement of the absolute position and orientation of the mobile robot at suitable locations along the motion trajectories. For instance this calibration technique requires measurement of the starting and final robot configuration for each motion execution. The datasets supplied information related to: the camera position x, y, θ and increments of the encoder. Each information is saved in columns:

- column 1: acquisistions time [ms];
- columns 2–3–4: camera pose x, y, θ [cm, cm, rad];
- column from 5 to 13: camera covariance ordered by rows [cm^2] for pose, [rad^2] for angle and [cm $*$ rad] for mixed terms;
- column 14–15: odometric encoder [ticks left] and [ticks right];

It has been chosen to perform the odometry calibration whereas in the same calculation all four datasets provided as suggested by the technique used. Then the equations (4) and (5) are rewritten limited to the four paths to obtain:

$$\begin{bmatrix} \hat{c}_{2,1} \\ \hat{c}_{2,2} \end{bmatrix} = (\bar{\Phi}_{\theta}^{\mathrm{T}}\,\bar{\Phi}_{\theta})^{-1}\,\bar{\Phi}_{\theta}^{\mathrm{T}} \begin{bmatrix} \theta_{\mathrm{N},1}-\theta_{0,1} \\ \theta_{\mathrm{N},2}-\theta_{0,2} \\ \theta_{\mathrm{N},3}-\theta_{0,3} \\ \theta_{\mathrm{N},4}-\theta_{0,4} \end{bmatrix} \tag{4}$$

$$\begin{bmatrix} \hat{c}_{1,1} \\ \hat{c}_{1,2} \end{bmatrix} = (\bar{\Phi}_{xy}^{\mathrm{T}}\,\bar{\Phi}_{xy})^{-1}\,\bar{\Phi}_{xy}^{\mathrm{T}} \begin{bmatrix} x_{\mathrm{N},1}-x_{0,1} \\ y_{\mathrm{N},1}-y_{0,1} \\ \vdots \\ x_{\mathrm{N},4}-x_{0,4} \\ y_{\mathrm{N},4}-y_{0,4} \end{bmatrix} \tag{5}$$

As a result of simulations shows the values of the matrix C:

$$\begin{bmatrix} 8.1873 & 6.5229 \\ 0.2823 & -0.2807 \end{bmatrix} \tag{6}$$

It is noted that the parameters $c_{1,1}$ and $c_{1,2}$ relative to the spokes of the wheels are different from each other, this is due to the simplifications introduced by the kinematic model. On the other hands, the axle values, $c_{2,1}$ and $c_{2,2}$, without the negative sign, are much more similar because formerly estimated at $c_{1,j}$ therefore do not contain the error propagation. Subsequently, it calculates the average between the values of the obtained rays and the standard deviation, these are shown in table (1).

	Radius	Mean	standard
	[cm]	[cm]	deviation [cm]
r_{R}	13.046	14.710	±2.353
r_{L}	16.375	14.710	±2.353
b	52.243		

Table 1: estimated value

Finally, figures 5, it shows the calculation for each path odometric with parameters previously estimated in comparison with the trajectory recorded by the camera.

iv. Optimization

To achieve the goal of determining the offset of the camera mounted on the robot, it is resorted to multivariable. Using the parameters obtained previously, summarized in the table 1, they were used as the boundary conditions limits. The search space is made up of six variables and their contour conditions are reported in the table 2 where the initials "LB" and "UB" are the *low boundaries* and *upper boundary* conditions.

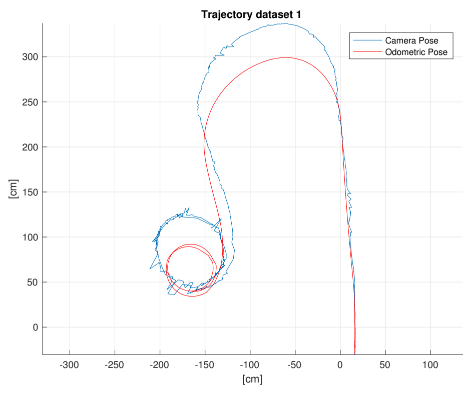
	r_{R}	r_{L}	b	ξ	d	α
	[cm]	[cm]	[cm]	[rad]	[cm]	[rad]
LB	12	12	51	$-\pi$	5	$-\pi$
UB	17.6	17.6	60	π	30	π

Table 2: Boundary condition in GA

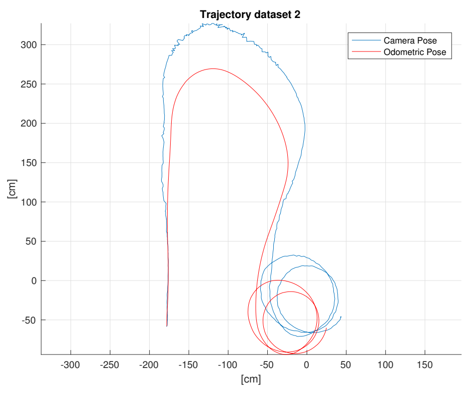
The algorithm used is part of the family of "*genetic algorithms* [2]" used for finding optimal solution. In a genetic algorithm, a population of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties which can be mutated and altered; traditionally, solutions are represented in binary as strings of 0*s* and 1*s*, but other encodings are also possible [3]. The evolution usually starts from a population of randomly generated individuals, and is an iterative process, with the population in each iteration called a generation. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are stochastically selected from the current population, and each individual's genome is modified (recombined and possibly randomly mutated) to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

A typical genetic algorithm requires:

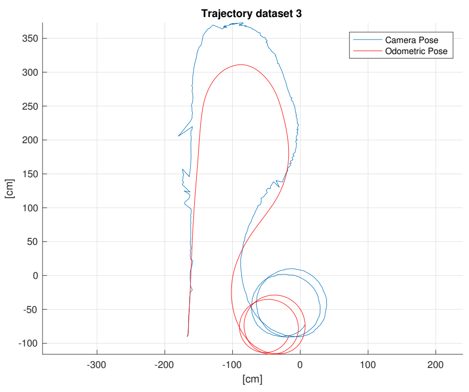
- a genetic representation of the solution domain,



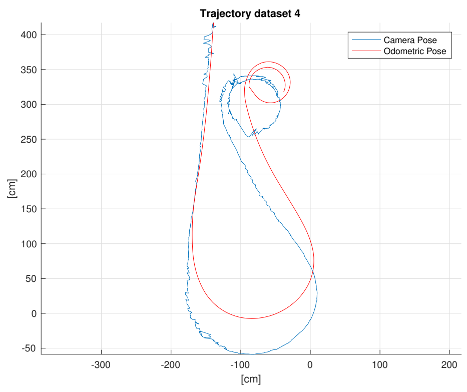
(a) *dataset 1.*



(b) *dataset 2.*



(c) *dataset 3.*



(d) *dataset 4.*

Figure 5: Odometry reconstruction

- a fitness function to evaluate the solution domain.

A standard representation of each candidate solution is as an array of bits[3]. Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, which facilitates simple crossover operations. Variable length representations may also be used, but crossover implementation is more complex in this case. Tree-like representations are explored in genetic programming and graph-form representations are explored in evolutionary programming; a mix of both linear chromosomes and trees is explored in gene expression programming. Once the genetic representation and the fitness function are defined, a GA proceeds to initialize a population of solutions and then to improve it through repetitive application of the mutation, crossover, inversion and selection operators. The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions. Often, the initial population is generated randomly, allowing the entire range of possible solutions (the search space). Occasionally, the solutions may be "seeded" in areas where optimal solutions are likely to be found. During each successive generation, a portion of the existing population is selected to breed a new generation. Individual solutions are selected through a fitness-based process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected. Certain selection methods rate the fitness of each solution and preferentially select the best solutions. Other methods rate only a random sample of

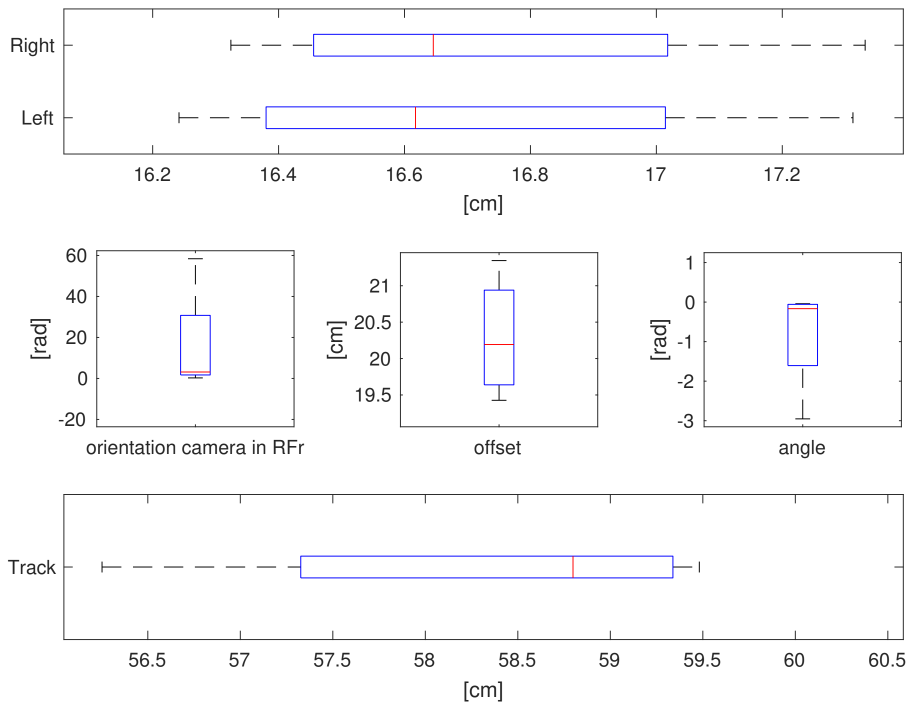


Figure 6: optimization among all dataset

the population, as the former process may be very time-consuming. The fitness function is defined over the genetic representation and measures the quality of the represented solution. The fitness function is always problem dependent. In some problems, it is hard or even impossible to define the fitness expression; in these cases, a simulation may be used to determine the fitness function value of a phenotype, or even interactive genetic algorithms are used. The next step is to generate a second generation population of solutions from those selected through a combination of genetic operators: crossover, and mutation. For each new solution to be produced, a pair of "parent" solutions is selected for breeding from the pool selected previously. By producing a "child" solution using the above methods of crossover and mutation, a new solution is created which typically shares many of the characteristics of its "parents". New parents are selected for each new child, and the process continues until a new population of solutions of appropriate size is generated. Although reproduction methods that are based on the use of two parents are more "biology inspired", some research suggests that more than two "parents" generate higher quality chromosomes. These processes ultimately result in the next generation population of chromosomes that is different from the initial generation. Generally the average fitness will have increased by this procedure for the population, since only the best organisms from the first generation are selected for breeding, along with a small proportion of less fit solutions. These less fit solutions ensure genetic diversity within the genetic pool of the parents and therefore ensure the genetic diversity of the subsequent generation of children. Although crossover and mutation are known as the main genetic operators, it is possible to use other operators such as regrouping, colonization-extinction, or migration in genetic algorithms. It is worth tuning parameters such as the mutation probability, crossover probability and population size to find reasonable settings for the problem class being worked on. A very small mutation rate may lead to genetic drift (which is non-ergodic in nature). A recombination rate that is too high may lead to premature convergence of the genetic algorithm. A mutation rate that is too high may lead to loss of good solutions, unless elitist selection is employed. This generational process is repeated until a termination condition has been reached. Common terminating conditions are:

- A solution is found that satisfies minimum criteria
- Fixed number of generations reached
- Allocated budget (computation time/-money) reached
- The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results
- Manual inspection
- Combinations of the above

It is made of a objective function that evaluates the difference of position and orientation between the path registered by the robot's camera with the odometry reconstruction carried out starting from the data as showed in the first part of listing code 3. As mentioned earlier after reconstructing the odometric path, position and orientation information is subtracted from those in the input files and computed the value using the equation (7), known as *root mean square* (RMS).

$$x_{rms} = \sqrt{\frac{1}{n}(x_1^2 + \cdots + x_n^2)} \tag{7}$$

In this case the function is already available in the matlab environment in which we pass the previously calculated offset variables as arguments, the sequence of operations is visible at the bottom of the list 3. The optimization is performed on all the datasets, thus returning four best combinations for the path observable in boxplot, fig. 6. It is possible to observe the behavior of the four best samples:

- extreme values, represented black dashed lines;
- median values, represented by the red segment;

In the first box at the top it shows the distribution of values for the radii's wheels left and right evaluated for the four data series. Mid boxes show the orientation values, camera distance from robot's center and realtive angle orienatation. Finally, the last box shows the track value. After performing the optimization it is impossible to determine with greater accuracy the searched data and the results for all four datasets are visible in figures 7–8–9–10. It is observed that the odometry reconstruction,in red, deviates from the original with odometric center in robot's middle axle, in black; on the other side, the optimal position of the calculated position camera, in blue, is much closer to the original though it takes into account the displacement from the midpoint previously considered.

III. Results

Using the method proposed in scientific article [1] allows to obtain the values in the table (1); it is observed that the distance between the robot's actual parameters and the estimates are introduced by the assumed model simplifications and errors. In the first analysis, the odometric center is placed in mid's robot axle as well as in the model used, but this method does not take into account that the camera's position may be not aligned with the point previously analyzed. Errors also depend on assumptions regarding the non-deformability of the wheels, the tire above keyed and misalignments. The motion is achieved by considering a regular and uniform surface, ignoring bumps, obstacles, unevenness, etc. In this scientific article, as in other scientific works [4]–[5]–[6], is recommanded to use predetermined paths, possibly in a straight line, with counterclockwise and clockwise rotations in order to minimize errors and to allow independent calibration of the parameters. On the other hand, for the results obtained from optimization we must take into account that there are limitations of the use of a genetic algorithm compared to alternative optimization algorithms. From the optimization operation we get the solution for each path visible in the table 3, you can see that the values are slightly different for each dataset. Repeated fitness function

r_R	r_L	b	ξ	d	α
16.716	16.704	56.253	3.142	21.345	−0.041
16.518	16.587	59.196	3.142	19.427	−0.259
16.242	16.324	58.401	3.142	19.852	−0.073
17.312	17.332	59.481	0.261	20.534	−2.954

Table 3: GA result

evaluation for complex problems is often the most prohibitive and limiting segment of artificial evolutionary algorithms. Finding the optimal solution to complex high-dimensional, multimodal problems often requires very expensive fitness function evaluations. The "better" solution is only in comparison to other solutions. As a result, the stop criterion is not clear in every problem. In many problems, GAs may have a tendency to converge towards local optima or even arbitrary points rather than the global optimum of the problem. This means that it does not "know how" to sacrifice short-term fitness to gain longer-term fitness. The likelihood of this occurring depends on the shape of the fitness landscape: certain problems may provide an easy ascent towards a global optimum, others may make it easier for the function to find the local optima. This problem may be alleviated by using a different fitness function, increasing the rate of mutation, or by using selection techniques that maintain a diverse population of solutions, although the "*No Free Lunch theorem*" proves that there is no general solution to this problem. A common technique to maintain diversity is to impose a "niche penalty", wherein, any group of individuals of sufficient similarity (niche radius) have a penalty added, which will reduce the representation of that group in subsequent generations, permitting other (less similar) individuals to be maintained in the population. This trick, however, may not be effective, depending on the landscape of the problem. Another possible technique would be to simply replace part of the population with randomly generated individuals, when most of the population is too similar to each other. Diversity is important in genetic algorithms (and genetic programming) because crossing over a homogeneous population does not yield new solutions. In evolution strategies and evolutionary programming, diversity is not essential because of a greater reliance on mutation.

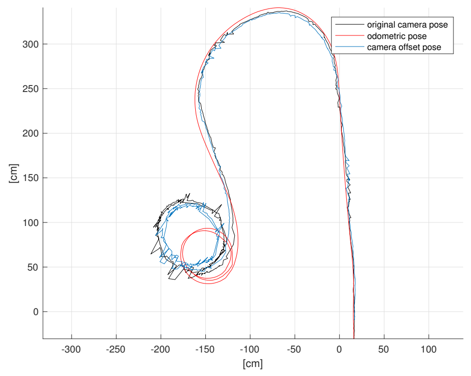


Figure 7: Optimal odometric and camera position, dataset: 1

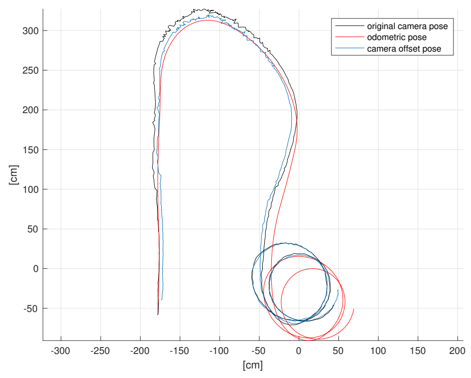


Figure 8: Optimal odometric and camera position, dataset: 2

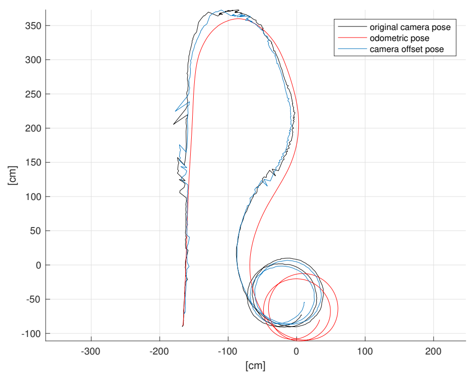


Figure 9: Optimal odometric and camera position, dataset: 3

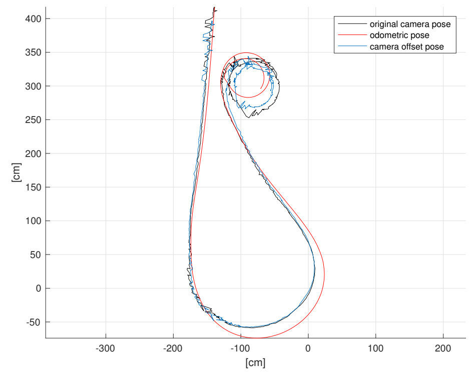


Figure 10: Optimal odometric and camera position, dataset: 4

References

[1] G. Antonelli, S. Chiaverini, and G. Fusco, "A calibration method for odometry of mobile robots based on the least-squares technique: theory and experimental validation," *IEEE Transactions on Robotics*, vol. 21, no. 5, pp. 994–1004, Oct 2005.

[2] Wikipedia, "Genetic algorithm — wikipedia, the free encyclopedia," 2017. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Genetic_algorithm&oldid=774791868

[3] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, no. 2, pp. 65–85, 1994.

[4] A. Censi, A. Franchi, L. Marchionni, and G. Oriolo, "Simultaneous calibration of odometry and sensor parameters for mobile robots," *IEEE Transactions on Robotics*, vol. 29, no. 2, pp. 475–492, April 2013. [Online]. Available: http://purl.org/censi/2012/joint_calibration

[5] D. Jung, J. Seong, C.-b. Moon, J. Jin, and W. Chung, "Accurate calibration of systematic errors for car-like mobile robots using experimental orientation errors," *International Journal of Precision Engineering and Manufacturing*, vol. 17, no. 9, pp. 1113–1119, 2016.

[6] K. S. Chong and L. Kleeman, "Feature-based mapping in real, large scale environments using an ultrasonic array," *I. J. Robotics Res.*, vol. 18, no. 1, pp. 3–19, 1999.

Listing 1: main.py

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import re
import sys
import tkinter as tk
from tkinter import filedialog
from dataframe import df, originalset


def multifilelaoader():
    root = tk.Tk()
    root.withdraw()
    file_path = filedialog.askopenfilenames()
    for x in file_path:
        print x
    return file_path


# count number of line in file input
def numberline(p_input):
    f = open(p_input, "r")
    n_line = len(f.readlines())
    f.close()
    return n_line


# regex filename
def newnamefile(p_name, p_inputfile):
    tmp = re.split(r"\/", p_inputfile)
    m = re.search(r"(\w+).(\w+)", tmp[5])
    # extract the entire namefile.ext
    complete_name = m.group(0)  # The entire match
    # extract namefile
    name = m.group(1)  # The first parenthesized subgroup.
    # extract extension
    ext = m.group(2)  # The second parenthesized subgroup.
    namefile = str(p_name) + "_" + name + "." + ext
    print ("New file created: "), namefile
    return namefile


# print line readed
def printerline(plines, pinput_file):
    for i in range(0, pinput_file, 1):
        print ("l " + str(i + 1) + ': ' + str(plines[i]))


# perform regex on line of file, return a dictionary
def generatedataset(plines, pinput_file):
    # pre allocate variable and split information
    n_time = []
    pose_x = []
    pose_y = []
    pose_theta = []
    cov_11 = []
    cov_12 = []
    cov_13 = []
    cov_21 = []
    cov_22 = []
    cov_23 = []
    cov_31 = []
    cov_32 = []
    cov_33 = []
    tick_L = []
    tick_R = []

    # define pattern for regex and cycle for all lines
    pattern = " \| "
    for n in range(2, pinput_file, 1):
        # regex to divide information in local variable
        new_data = re.split(pattern, str(plines[n]).replace('\n', ''))
        # fill the variables with splitted information
        n_time.append(int(new_data[0]))
        pose_x.append(float(new_data[1]))
        pose_y.append(float(new_data[2]))
        pose_theta.append(float(new_data[3]))
        cov_11.append(float(new_data[4]))
        cov_12.append(float(new_data[5]))
        cov_13.append(float(new_data[6]))
        cov_21.append(float(new_data[7]))
        cov_22.append(float(new_data[8]))
        cov_23.append(float(new_data[9]))
        cov_31.append(float(new_data[10]))
        cov_32.append(float(new_data[11]))
        cov_33.append(float(new_data[12]))
        tick_L.append(int(new_data[13]))
        tick_R.append(int(new_data[14]))

    # invert value of left encoder
    for j in range(0, len(tick_L), 1):
        tick_L[j] = - (tick_L[j])

    for j in range(0, len(tick_L), 1):
        tick_R[j] = abs(tick_R[j])

    # give "-" sign to pose.theta
    for z in range(0, len(pose_theta), 1):
        pose_theta[z] = - pose_theta[z]

    # set dictionary for dataset
    dataset = {'n_time': n_time,
               'pose_x': pose_x,
               'pose_y': pose_y,
               'pose_theta': pose_theta,
               'cov_11': cov_11,
               'cov_12': cov_12,
               'cov_13': cov_13,
               'cov_21': cov_21,
               'cov_22': cov_22,
               'cov_23': cov_23,
               'cov_31': cov_31,
               'cov_32': cov_32,
               'cov_33': cov_33,
               'tick_L': tick_L,
               'tick_R': tick_R}

    return dataset


# MAIN FUNCTION
if __name__ == "__main__":

    for input_file in multifilelaoader():
        # read the content of file
        file_open = open(input_file, "r")
        text = file_open.read()

        # # divide text in lines
        lines = text.splitlines(True)
        # print type(lines)  # for debugging, comment to avoid print
        # printerline(lines, numberline(input_file))

        rewrite = originalset(generatedataset(lines, numberline(input_file)))
        # rewrite dataset file
        print "Rewriting of the file..."
        out_file = open(newnamefile("rew", input_file), "w")
        out_file.write(rewrite)
        out_file.close()
        print "Done!"

        # create new dataset correct format
        newdataset = df(generatedataset(lines, numberline(input_file)))
        print "Writing new file..."
        # create new dataset file
        out_file = open(newnamefile("new", input_file), "w")
        out_file.write(newdataset)
        out_file.close()
        print "Done!"

    # auto exit script
    print "ALL DONE..."
    sys.exit()
```

Listing 2: dataframe.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import pandas as pd


def df(dataset):
    # export order
    order = ['n_time', 'pose_x', 'pose_y', 'pose_theta', 'cov_11',
             'cov_12', 'cov_13', 'cov_21', 'cov_22', 'cov_23',
             'cov_31', 'cov_32', 'cov_33', 'tick_L', 'tick_R']

    # generate data frame
    data = pd.DataFrame.from_records(dataset, columns=order)

    # list parametrs to eliminate duplicates
    param = ['pose_x', 'pose_y', 'pose_theta', 'cov_11',
             'cov_12', 'cov_13', 'cov_21', 'cov_22', 'cov_23',
             'cov_31', 'cov_32', 'cov_33']
    # print preview whitout duplicate, uncomment to debugging
    data = data.drop_duplicates(subset=param, keep='last')
    # possibiy dublicate in time gnerate 'inf' value in matlab
    param2 = ['n_time']
    no_duplicates = data.drop_duplicates(subset=param2, keep='last')

    # return correct dataframe without duplicates
    return no_duplicates.to_string(index=False)


def originalset(dataset):
    # export order
    order = ['n_time', 'pose_x', 'pose_y', 'pose_theta', 'cov_11',
             'cov_12', 'cov_13', 'cov_21', 'cov_22', 'cov_23',
             'cov_31', 'cov_32', 'cov_33', 'tick_L', 'tick_R']

    # generate data frame
    data = pd.DataFrame.from_records(dataset, columns=order)

    # return correct dataframe without duplicates
    return data.to_string(index=False)
```

Listing 3: gaerror.m

```matlab
function [ error ] = gaerror(inputparams, data)
% OBJFUN calculates error for every configuration of parameters return
% error along the path

% Encoder resolution (multiplied by gear train ratio)
res_encoder = 16384 * 25;

% Initialize local variable
[ diffleft, diffright ] = tick2differenceTick( data );

% Initialize array and calc new position
newpose.x(1)   = data.pose.x(1);
newpose.y(1)   = data.pose.y(1);
newpose.psi(1) = data.pose.psi(1);

% Calculate ododometric recostruction
for i=1:length(data.pose.psi)
    newpose.x(i+1)   = newpose.x(i)   +   pi * (diffright(i) * inputparams(2) + diffleft(i) * inputparams(1)) * (cos(newpose.psi(i))/res_encoder);
    newpose.y(i+1)   = newpose.y(i)   +   pi * (diffright(i) * inputparams(2) + diffleft(i) * inputparams(1)) * (sin(newpose.psi(i))/res_encoder);
    newpose.psi(i+1) = newpose.psi(i) + 2*pi * (diffright(i) * inputparams(2) - diffleft(i) * inputparams(1)) / (res_encoder * inputparams(3));
end

% clear duplicate rows
newpose.x(1)   = [];
newpose.y(1)   = [];
newpose.psi(1) = [];

% calc offset
x_offset = data.pose.x - inputparams(5) * cos(inputparams(6) + data.pose.psi - inputparams(4));
y_offset = data.pose.y - inputparams(5) * sin(inputparams(6) + data.pose.psi - inputparams(4));
xhi = data.pose.psi - inputparams(4);

% Objective function, adjust newpose to perform difference
X_err     = x_offset - newpose.x.';
Y_err     = y_offset - newpose.y.';
XHI_err   = xhi - newpose.psi.';

% total error
error = rms(X_err + Y_err + XHI_err);

end
```