eYSIP2017

Automatic Evaluation Of Black Line Following Robot

Nikhilesh M
Mugdha Pandya
Mentor: Khalid Waseem
Duration of Internship: 22/05/2017 – 07/07/2017

2017, e-Yantra Publication

Automatic Evaluation of Black Line Following Robot

Abstract

This project develops a video analytic software system for automatic evaluation of videos of robots following black line in the e-yantra competitions. The videos are taken using unknown camera while the robot moves over predefined arena of fixed dimensions. The arena as well as the robot are labelled with ArUco markers (see Figure 1.1). Robot motion trajectory was extracted from code for Indoor Localization System [3]. Next, the trajectory is evaluated by computing the average deviation from an ideal path. Experiments were carried out to measure the accuracy which showed that the robot can be localized to about 1 cm accuracy on a 6×6 feet arena from a video captured on mobile phones. An automatic evaluation of videos of black line following robots was carried out and found to match the human evaluation.

Completion status

All assigned tasks were completed. Prototype version of the required software has been produced. The completed tasks are as follows:

1. Understanding camera calibration problem. Familiarizing with Indoor Localization System
2. Installation and set up of required software
3. Camera calibration with chessboard patterns and localization with red and green markers
4. Evaluation of robot trajectory over straight line paths using square window

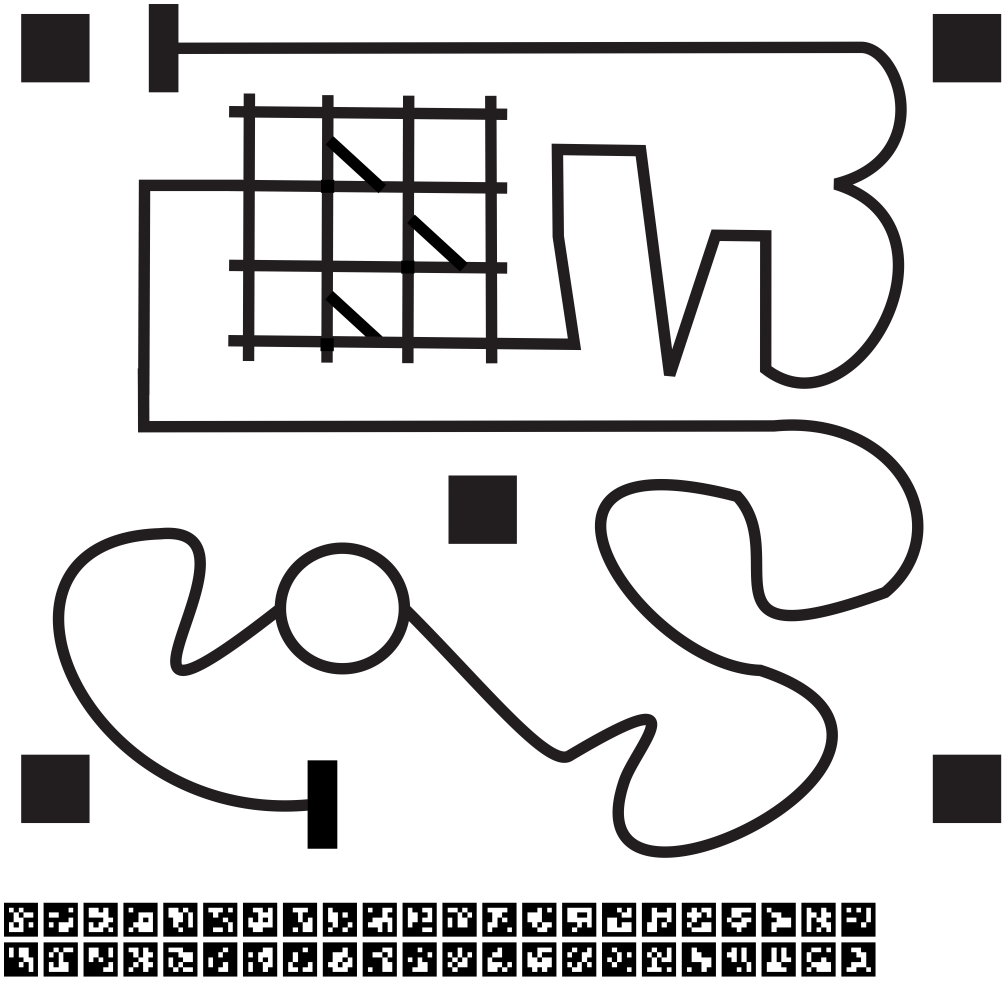

Figure 1.1: Arena with ArUco Markers

5. Data collection for precision analysis of localization with red-green markers
6. Ideal path generation
7. Evaluation of robot trajectory over curved paths using circular window
8. Data collection for precision analysis of localization with ArUco markers
9. Evaluation of robot trajectory with multi-parts
10. System integration
11. Documentation

1.1 Hardware used

- Firebird V, Specification, Vendor link
- Arena used for e-yantra 2012 black line following competition
- Laptop/PC running recent version of Linux OS

1.2 Software used

- openCV version: 3.1, download link, installation steps
- MATLAB version: R2014a, download link, installation steps
- CorelDRAW graphics suite version: X7, download link
- Adobe Illustrator version: 2017, download link
- camera MX version: varies with device, download link
- C++
- ArUco library download link

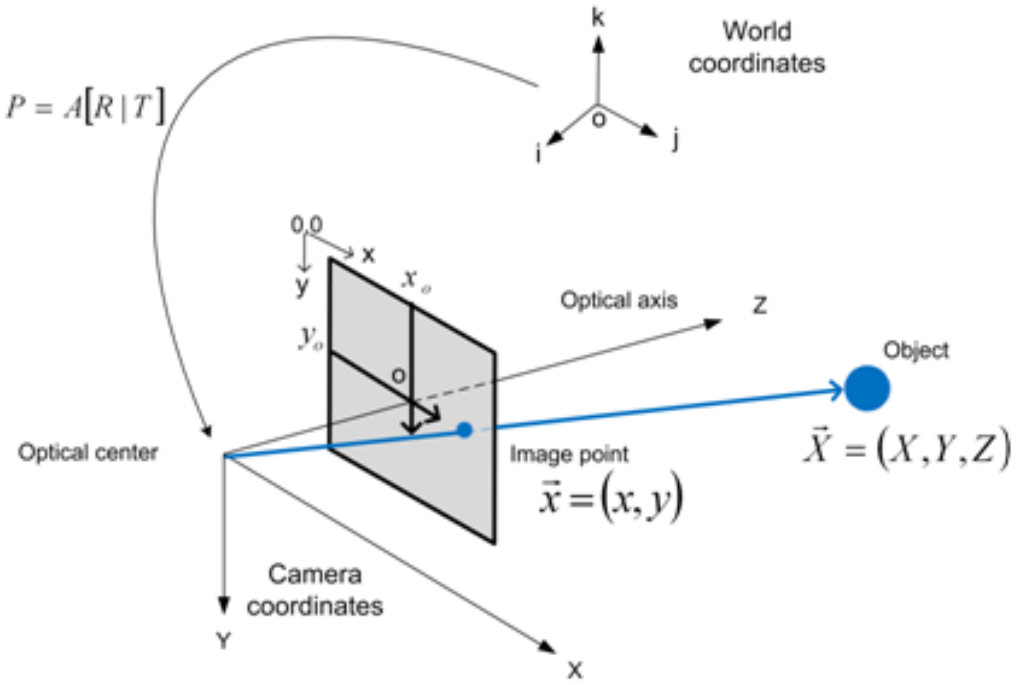

Figure 1.2: Coordinate systems in camera courtesy [5]

1.3 Camera Calibration and Extraction of Robot Trajectory Using Localization

The main challenge in robot trajectory extraction is that camera characteristics and its location are not known. To assist in the evaluation, some markers[7] of known dimensions are placed at known positions on the arena as well as on the robot. It is assumed that the robot moves in a horizontal plain and height of marker on robot is known.

The task of localization involves conversion of 2D image points (pixels) to 3D world points (centimeters) and using these points to generate the trace of the robot's movement in the video. As first step, camera calibration is carried out using the markers at known positions. Next, in each frame, camera pose (location and orientation) is determined with respect to the arena. Using this, the robot position in the world coordinates is localized.

We first summarize the technique of camera calibration and localization as formulated by [2] and [1]. Mathematically, a simple camera model can be formulated as follows. Figure 1.2 depicts the model.

$$s \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = M_{int} \left(R \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} + T \right) \tag{1.1}$$

Moving from 3D world coordinates (x, y, z) to two image coordinates (u, v) is a two step process [2]. See Equation 1.1. We must first move from world coordinates to camera coordinates. This involves rotation and translation and is captured by rotation matrix R and translation vector T. (A composite matrix $M_{ext} = [R\ T]$ consisting of columns of R followed by a column of T is often called extrinsic matrix). This transformation is independent of the camera features such as focal length but it changes for different orientations of the camera. In the next step we move from point in camera coordinates to image point (u, v) in pixels. This transformation depends on the camera and is given by M_{int} below. Each camera has some intrinsic parameters such as focal length, optical center etc. M_{int} has the form

$$\begin{bmatrix} fx & 0 & ox \\ 0 & fy & oy \\ 0 & 0 & 1 \end{bmatrix}$$

(See reference [3], [4].) These parameters remain constant for a particular camera independently of its position and orientation.

Given sufficient number of points in world coordinates and corresponding image coordinates, we can solve Equation (1.1) to obtain intrinsic and extrinsic camera matrices. This called **camera calibration problem** [2].

Now we consider localization. From the above camera model where M_{int} and M_{ext} are known, we can convert from 2D image coordinates to 3D world coordinates only in some special cases. As we can see, the extrinsic matrix is non invertible, thus making it non-trivial to solve the equation. If we assume that the height of the robot (z) is fixed, then the equation can be solved using method described in section 3.4 of [3]. Since we know the height of the robot we set z as the height, we solve these as simultaneous linear equations and get the 3D x,y coordinates as follows[3].

$$\begin{bmatrix} (x_{im} - o_x)r_{31} - f_x r_{11} & (x_{im} - o_x)r_{32} - f_x r_{12} \\ (y_{im} - o_y)r_{31} - f_y r_{21} & (y_{im} - o_y)r_{32} - f_y r_{22} \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \end{bmatrix} = \begin{bmatrix} Z_w[f_x r_{13} - (x_{im} - o_x)r_{33}] + f_x T_x - (x_{im} - o_x)T_z \\ Z_w[f_y r_{23} - (y_{im} - o_y)r_{33}] + f_y T_y - (y_{im} - o_y)T_z \end{bmatrix} \tag{1.2}$$

Equation 1.2 above can be solved to obtain X_w and Y_w. This procedure is repeated for each frame, generating a trace file. Figure 1.3 gives the ArUco markers detected by the software. Figure 1.4 gives part of the trace file generated.

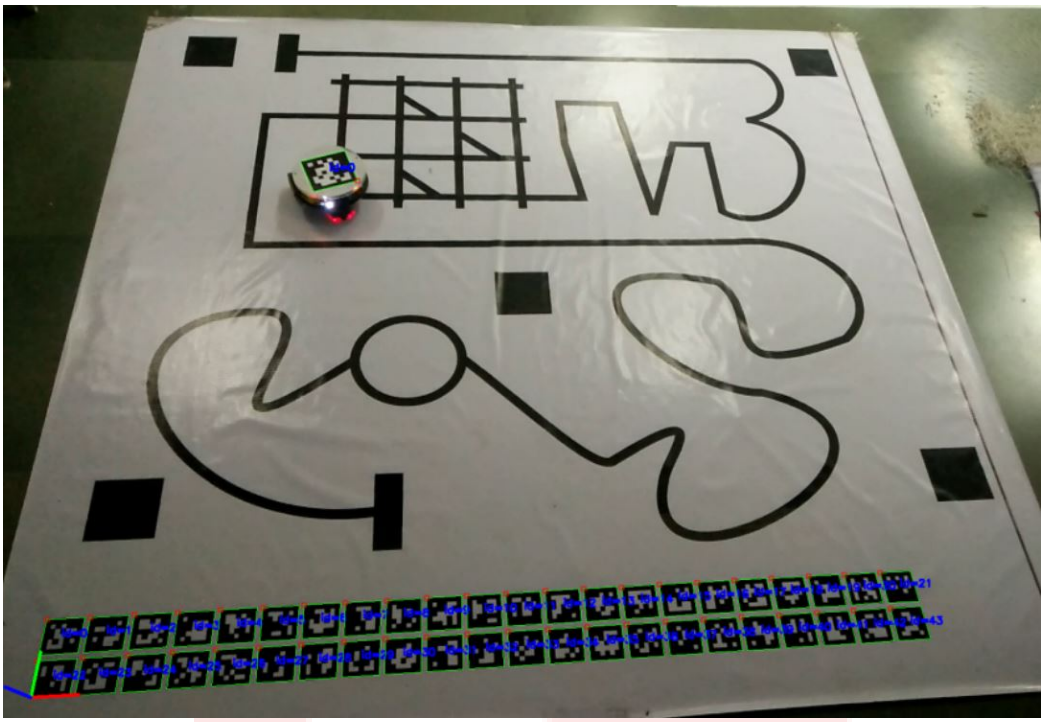

Figure 1.3: Detection of ArUco markers. Detected markers labelled in blue. 3D coordinate axis are shown.

0	178.012	51.944	30.5741	0.04006
1	178.128	51.9634	30.5872	0.080119
2	178.12	51.9669	30.5895	0.120179
3	178.328	51.958	30.6064	0.160238
4	178.136	51.9703	30.5795	0.200298
5	177.88	51.9564	30.5682	0.240357
6	177.564	51.97	30.5316	0.280417
7	177.528	51.9629	30.5244	0.320476

Figure 1.4: Trace File. Each row gives frame number, orientation (degrees), x,y and time (sec)

1.3.1 Precision Analysis

Block with ArUco marker was placed at various known positions and its position was extracted. Output from many such trails using two different mobile phone cameras was statistically analyzed. Detailed analysis can be found in [3]. It was found that error in localization from camera1 had mean error 0.89cm, standard deviation 0.51cm and maximum error 1.2 cm for a 6×6 feet arena. Camera 2 had mean error 0.77 cm, standard deviation 0.43 cm and maximum error 1.25. The maximum error in orientation was under 2 degrees. It was found that error was larger when the block being localized was farther away from the camera.

1.4 Trajectory Evaluation

1.4.1 Main Function

See figure 1.5 on page 8.

1.4.2 Ideal Sort Function

See figure 1.6 on page 9.

1.4.3 Weigh Function

See figure 1.7 on page 10.

1.4.4 compare.m

1. Initialize/declare
2. Read ideal file
3. Find initial point from the ideal file using circular search function
   If the output of the function is -1 then the starting point is not in the ideal file. Hence, wrong position output and returns with -1, 0, 0, 0 as output to main file.
4. Comparison algorithm
   describe the window size by xmax, xmin and ymax, ymin with respect to the initial point.

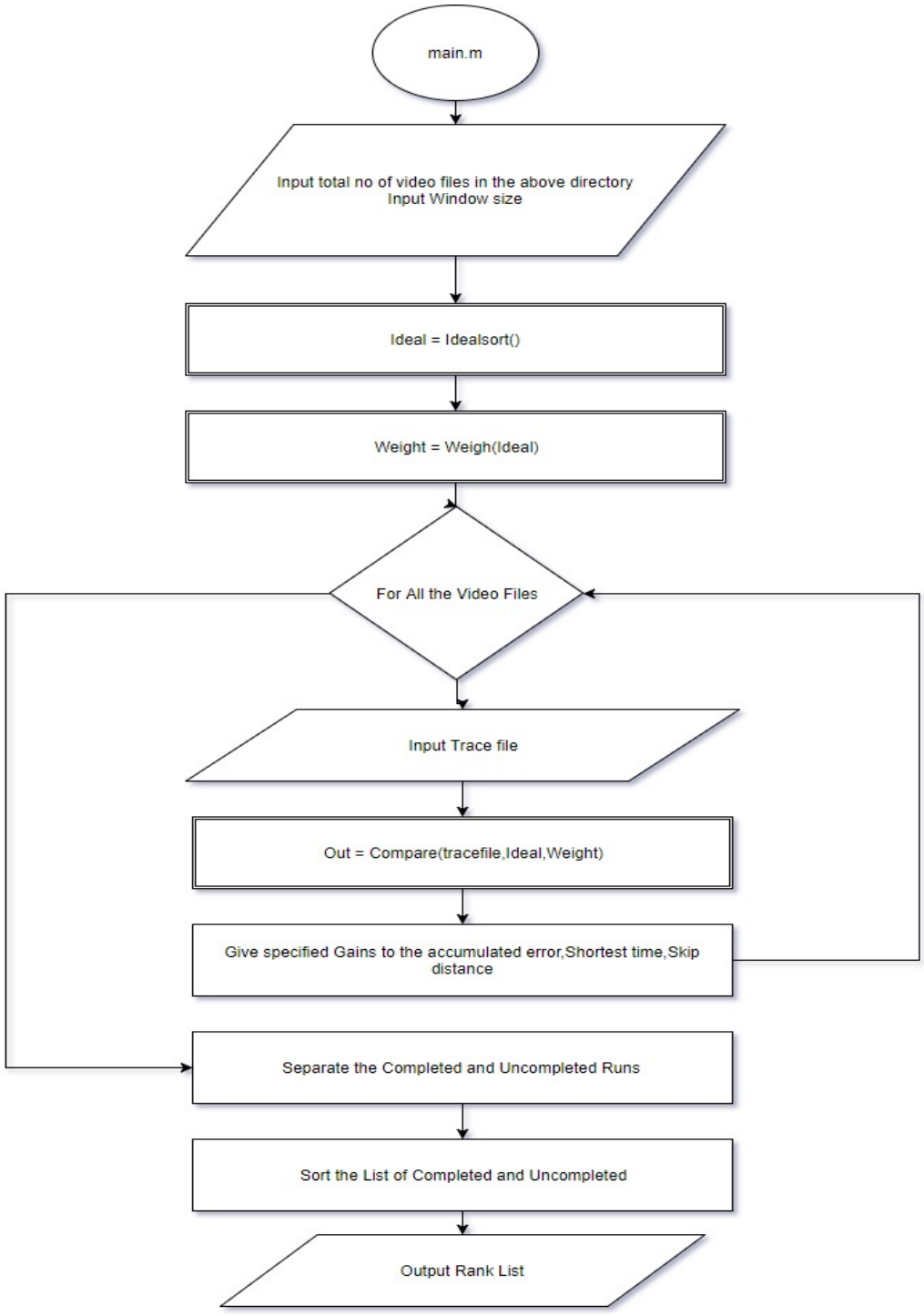

Figure 1.5: Main Function

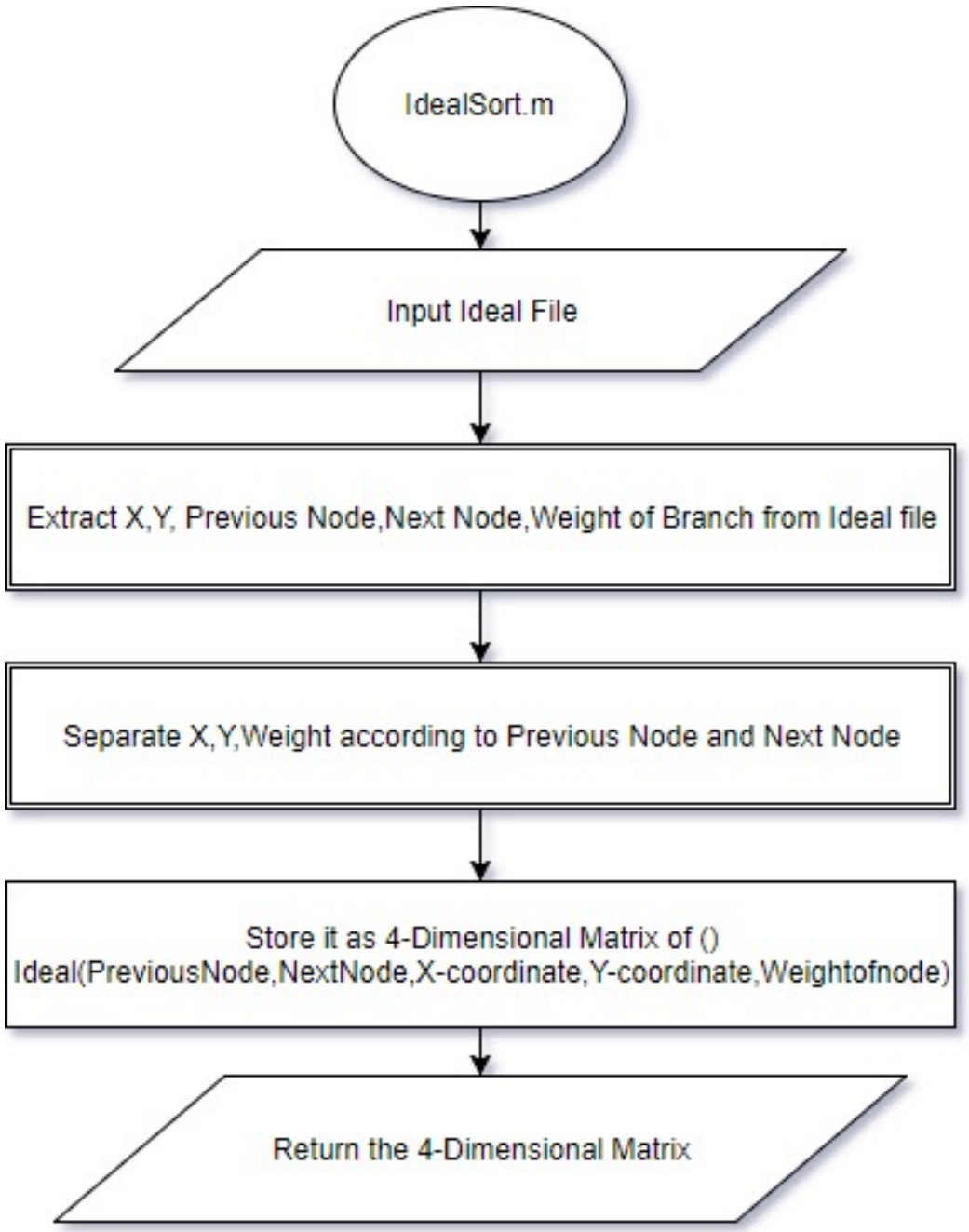

Figure 1.6: Ideal Sort Function

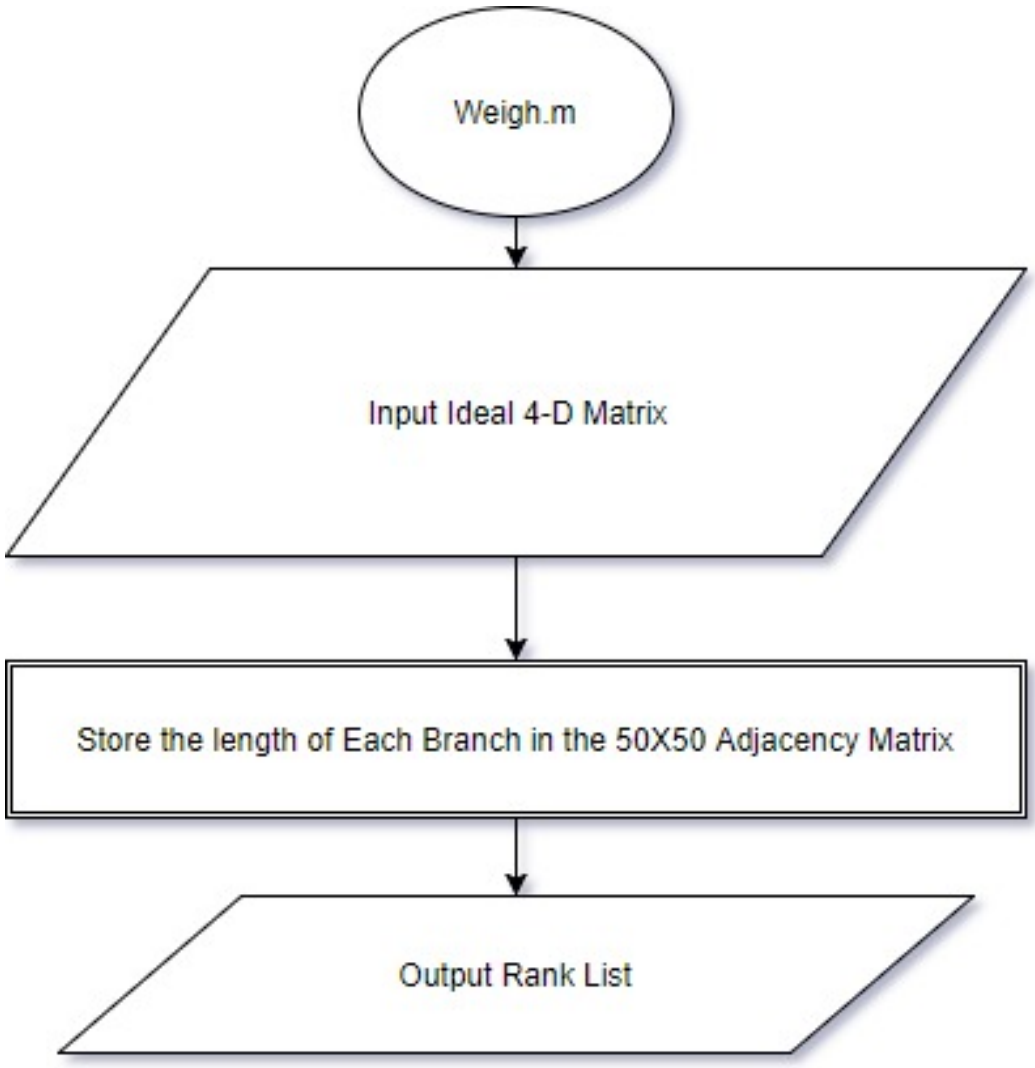

Figure 1.7: Weigh Function

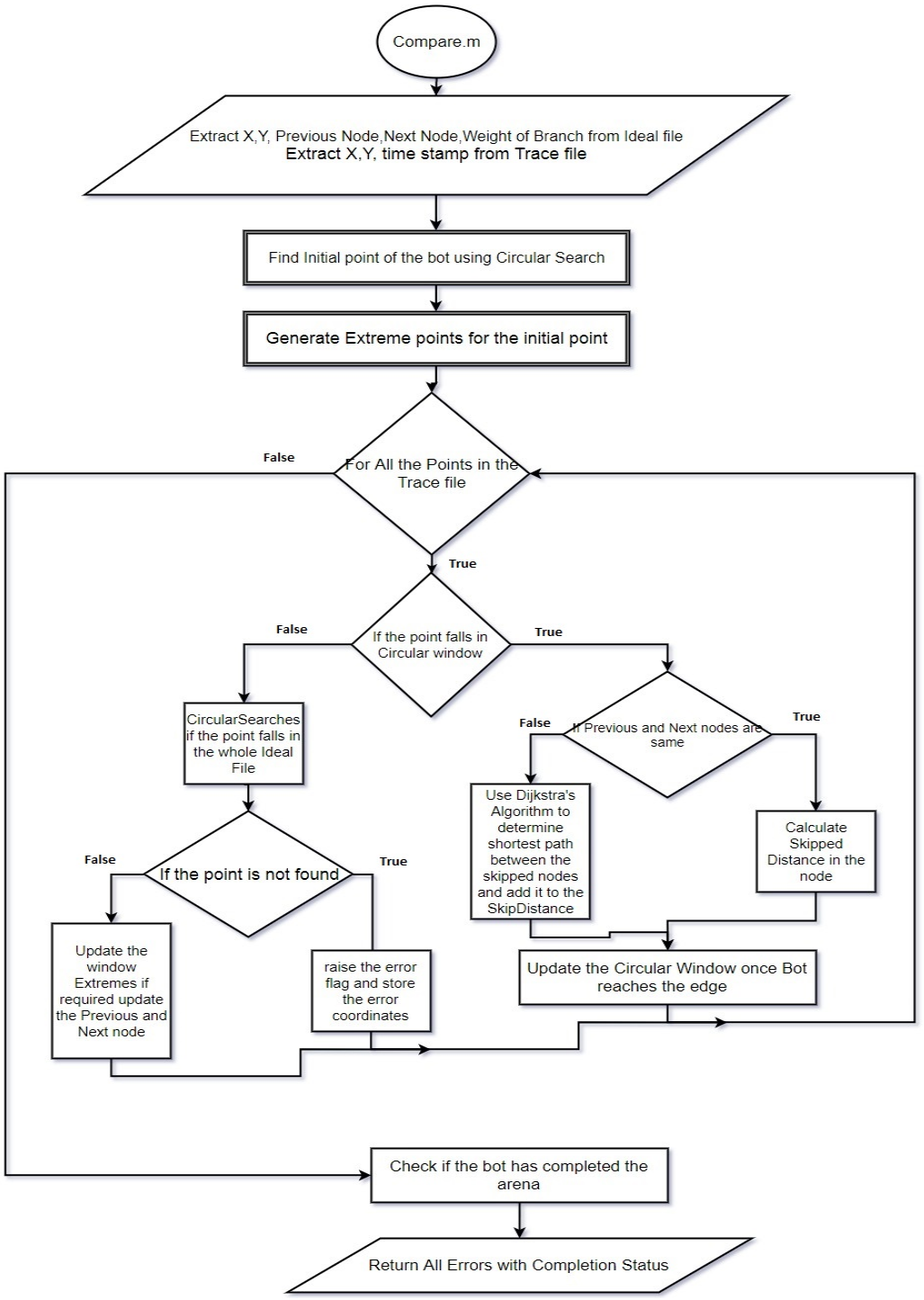

Figure 1.8: Compare Function

(a) Initialize (starting assumption)
Declare Prev node as 1
Declare Next node as 2

(b) Loop for all practical points
(plen = practical file length)

i. Search in the circular window

A. if a point is present
check if it was in error state by error flag=1
if it is in error state then store the present co-ordinates with time stamp that it had come back to the line

B. Calculate skip distance

C. if it comes back in the same node
calculate skip distance within the node by adding all skipped pixels multiplied with pixel:centimeter ratio.

D. else
it goes to another branch

E. calculate from where it took off to the end of the branch and add to skip distance

F. calculate the distance skipped in the receiving node

G. calculate the smallest distance between the next node of the branch it left from to previous node it has arrived using Dijkstra's algorithm which gives shortest weight from adjacency matrix and add to skip distance

H. save in error file

I. Update window location
Along x and y axes it updates when the x or y coordinate reaches beyond 3/4th region of the circular window

ii. else if point is not found in circular window
circular searches that practical point along the ideal file

A. if it is not found in the whole ideal file
save the previous coordinates of the point at which it went off from the path in the error matrix
calculate Euclidean distance and add to accumulated error

B. else
update the index and xmax, xmin and ymax, ymin and prev node and next node

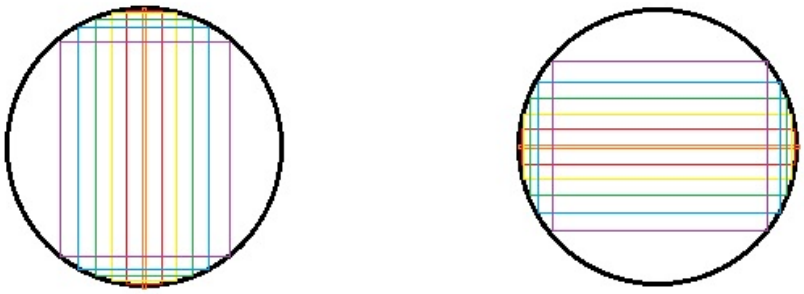

Figure 1.9: Circular Window divided in 9x9 parts

iii. check if it has completed the run

5. calculate total time taken
6. display error
7. return (accumulated error, time, skip distance, completion status)

1.4.5 Circle Window Function

- There are two loops which divide the circle into two.
- The function has used exponential function to have small variation in the beginning and large near the top half. It also checks if it falls in each of rectangular window
- returns the flag 1 if it is present
  returns the flag 0 if it is not within the window

1.4.6 Circular Search Function

- uses circular window function to search along the ideal file
- gets the nearest ideal point to the practical point and sends back the index

1.5 Use and Demo

The source code and example demo files can be downloaded from Github code repository. The Indoor Localization System works in one of three input modes. Mode 0 takes input from set of images listed in a file, whereas mode 1 takes input from a video file. Mode 2 works from webcam. Output is a trace file as shown in Figure 1.4, and an evaluation of multiple videos of black line following robots is shown in Figure 1.10.

1.5.1 Requirements for Motion Evaluation from Video File

- Calibration video: video of the arena shown from different angles with markers visible (more than 10 seconds).
- Competition videos: videos of robot following the black line.
- Ideal file: contains coordinates of the black line to be followed.

Precautions:

- All ArUco markers on the arena must be visible.
- Videos should be taken after turning off auto focus. (Apps such as camera MX may be used)
- Lighting conditions must stay constant.
- No glossiness should be visible on the ArUco markers of the arena.

1.5.2 Camera Calibration

- Input: configuration_calibration.xml
  contains input mode, file name, marker size, number of markers, dictionary id etc.
- Execute: calibrate_camera_v1.0 [3]
  select frames using 'c'
  skip frames using 's'
  use 'esc' to imply end of selection of frames
- Output:calib_video.txt
  text file containing camera parameters

Calibration Using images

- change input mode to 0
- create text file containing names of all the images to be used
- output will be saved in calib.txt

1.5.3 Generate Trace File

- Input: calib_video.txt, configuration_predloc.xml
  contains input mode, file names, marker size, dictionary id, robot marker id etc.
- Execute: pred_loc [3]
  predicts location of bot in each frame
- Output: video_file_output.csv
  contains frame number, angle, x coordinate, y coordinate of bot in each frame along with timing information.

Figure 1.4 shows an example of output file generated.

Using images

- Use calib.txt as input
- change input mode to 0
- create text file containing names of all images to be used
- predicts location of robot in each image

1.5.4 Evaluate Trajectory of Robot

- Input: ideal file, computed trace files, number of videos, circular window size
- Execute: main.m
  compares multiple trace files based on the different criteria for evaluation.
- Output: ranking based increasing order of error
  videos split into two categories - incomplete and complete
  videos in each category ranked based on accumulated error, skipped distance and time taken

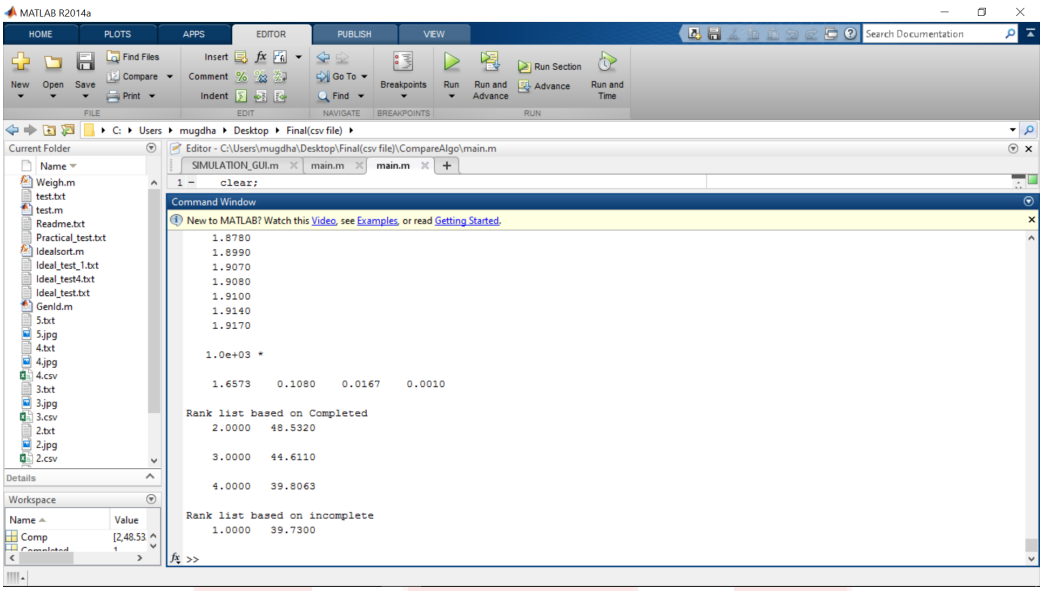

Figure 1.10: Evaluation Results

Figure 1.10 gives the results of one such evaluation.

To compare a set of files with the code

- Name the files in the order of number.txt/number.csv. (Because the result is according to these Numbers)
  ex - 1.txt/1.csv , 2.txt/2.csv ,3.txt/3.csv ,etc.
- Put it in the code folder by deleting the old numbered files.
- The videos must be trimmed before getting the output(because the time stamp is used for determining the fastest bot)

Use different arena

- Generate an Ideal File Using a single pixel Black Image.
- Know the pixel/centimeter ratio (which has to be changed in Ideal-Sample.m and Compare.m where variable name is 'pc').
- Change the offsets of X and Y axis.

- If the file is in pixels run IdealSample.m to convert it to cm and sample it.

  OR

- Provide Ideal file in centimeters

Youtube Link of demonstration video

1.6 Future Work

- Reduction in markers used: Currently, we use 44 markers. A study must be conducted to test the smallest size and the smallest number of markers that can be used without significant loss of precision.
- Different markers: A study can be conducted to see how the accuracy changes when we replace ArUco markers with other markers such as whycon markers.
- Open source release: Currently, the system runs on MATLAB. In the future, the entire system should move to an open source platform like openCV and away from MATLAB.
- Development for other themes: The system currently works only for the black line following theme. It can be extended to other themes in the competition such as pick and move theme.
- Varying conditions: test with different lighting conditions and arenas having different types of paths.

1.7 Challenges

- Video capture techniques: Videos taken from a smart phone camera produced trace files with coordinate values that did not match real coordinate values. It was found that smart phone cameras automatically change focus while capturing videos thus changing the intrinsic parameters of the camera. The solution was to install an application that can turn off auto focus. Videos captured via the application provided significantly better trace files.

- Window shape: While working with straight line paths a square window was enough. All points on the path could be detected. When using a curved path arena a square window excluded some of the points on the path and considered them to be errors. Thus, selecting an ideal shape for the window is necessary.

Bibliography

[1] Emanuele Trucco and Alessandro Verri, *Introductory techniques for 3-D computer vision*, 1998

[2] Z. Zhang, *A exible new technique for camera calibration*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(11):13301334, 2000.

[3] Khalid Waseem, *Video based Automatic Evaluation of Black Line following Robot*, M.Tech thesis, IIT Bombay, 2017

[4] Camera resectioning, Wikipedia article.

[5] Ntawiniga Frederic, e-book, *Head motion tracking in 3D space for drivers*, chapter 5

[6] Computing x,y coordinate (3D) from image point, stack overflow.

[7] ArUco Library Download