

Adaptive kinematic particle filter classifier for autonomous robots

MSc Research Project
Data Analytics

Ayoola Idris-Animashaun
Student ID: x20103689

School of Computing
National College of Ireland

Supervisor: William Clifford

National College of Ireland
Project Submission Sheet
School of Computing



Student Name:	Ayoola Idris-Animashaun
Student ID:	x20103689
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	William Clifford
Submission Due Date:	20/12/2018
Project Title:	Adaptive kinematic particle filter classifier for autonomous robots
Word Count:	1780
Page Count:	15

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	16th December 2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission , to each project (including multiple copies).	<input type="checkbox"/>
You must ensure that you retain a HARD COPY of the project , both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Adaptive kinematic particle filter classifier for autonomous robots

Ayoola Idris-Animashaun
x20103689

1 Introduction

This configuration manual contains hardware and software requirements. The Readme.md file contains the full details of this manual and can be consulted for easy running of this config.

2 Hardware and Software Requirements

2.0.1 Hardware Description

This project was researched on a Lenovo T450s with the following specification Processor: Intel(R) Core(TM) i5-5300U CPU @ 2.30GHz 2.29 GHz RAM: 16GB Operating System: 64-bit, x64based processor, Windows 10 Home/20H2, Windows Feature Experience Pack 120.2212.3920.0

2.0.2 Software Description

The following softwares will need to be installed prior to running this project. Standard distributions should work across operating systems. Please refer to these dependency versions if needed. Python 3.9.9 Jupyter Notebook 6.4.0 Opencv-python 4.5.3.56 Pandas 1.2.4 Scikit 1.0.1 Scipy 1.7.0 Seaborn 0.11.1 Notepad++ Web browsers

Phase 1 - Initializing the robot

Step 1: Run 'robot driver auto before motion model'.py from the command line by typing "python3 'robot driver auto before motion model'.py -steps 100" into the command line. You can enter 5, 10, 20, 50 or 100 only steps only. Please enter 100 to replicate this project.

Step 2: A file titled, "Before motion model (Number of steps you selected from Step 1) steps particle tracking auto.csv" will be available after Step 1 is done running.

Step 3: Run the "particle tracker before motion model".ipynb file in a Jupyter Environment

Step 4: A plot of the average distance between the robot and each particles will be plotted. The aim of this research is to get an average number that is lesser than this average.

Phase 2 - Generating the robot poses Step 1: From the Jupyter home of this directory. Run the 'robot wheel generator'.ipynb file in Jupyter to synthesize robot poses. This file will generate robot poses for 5, 10, 20, 50 and 100 steps each for a single fixed wheel, an omni wheel and a two fixed wheels. This will be used later by the classifiers we will build. The last cell in this notebook will generate test data that none of the classifiers will have seen. This will be used to ensure there is no over-fitting during hyper-parameter tuning

Step 2: Still in the jupyter home, run the 'Classifier Analysis'.ipynb file to see how the selected classifiers perform on the data generated in step 1 of Phase 2. The initial execution will run for 100 robot steps. A table of comparison among the classifiers will be presented along with the confusion matrix for each classifier. The following execution will then run for 5, 10, 20, 50, 100 to improve the performance of each classifier based on more input to the classifier. We see that the performance of the classifiers will improve with more steps. We see that KNN has a high log loss and so we will exclude it from the next stage where we tune the hyper-parameters on the models.

Step 3-5 is the same but for different classifiers

Step 3a: Run the 'Hyperparameter Tuning Decision Tree'.ipynb file. It will tune hyper-parameters for 100 robot steps initially. The output will contain the best parameters and plot of the test vs train error for accuracy and log loss. The second run will use the tuned hyper-parameters values to see how the model performance improves with more robot steps. A plot of accuracy and log loss will be displayed at the end of the run.

Step 3b. Run the 'Test Overfitting Decision Tree'.ipynb file to be sure the model in 3a has not been overfitted and can be generalised. PLEASE NOTE THAT STEP 3A MUST BE CARRIED OUT BEFORE THIS STEP CAN BE CARRIED OUT DUE TO THIS STEP USING THE WEIGHTS FROM STEP 3A FOR PREDICTION

Step 4a: Run the 'Hyperparameter Tuning Random Forest'.ipynb file. It will tune hyper-parameters for 100 robot steps initially. The output will contain the best parameters and plot of the test vs train error for accuracy and log loss. The second run will use the tuned hyper-parameters values to see how the model performance improves with more robot steps. A plot of accuracy and log loss will be displayed at the end of the run.

Step 4b. Run the 'Test Overfitting Random Forest'.ipynb file to be sure the model in 4a has not been overfitted and can be generalised. PLEASE NOTE THAT STEP 4A MUST BE CARRIED OUT BEFORE THIS STEP CAN BE CARRIED OUT DUE TO THIS STEP USING THE WEIGHTS FROM STEP 4A FOR PREDICTION

Step 3a: Run the 'Hyperparameter Tuning Logistic Regression'.ipynb file. It will tune hyper-parameters for 100 robot steps initially. The output will contain the best parameters and plot of the test vs train error for accuracy and log loss. The second run will use the tuned hyper-parameters values to see how the model performance improves with more robot steps. A plot of accuracy and log loss will be displayed at the end of the run.

Step 3b. Run the 'Test Overfitting Logistic Regression'.ipynb file to be sure the model in 3a has not been overfitted and can be generalised. PLEASE NOTE THAT STEP 3A MUST BE CARRIED OUT BEFORE THIS STEP CAN BE CARRIED OUT DUE TO THIS STEP USING THE WEIGHTS FROM STEP 3A FOR PREDICTION

Phase 3 Step 1: Run the 'Tuned Random Forest'.ipynb file to generate model weights for 5, 10, 20, 50, 100 robot steps.

Step 2: Run 'robot_driver_auto'.py from the command line by typing "python3 'robot_driver_auto'.py -steps 100" into the command line. You can enter 5, 10, 20, 50 or 100 only for steps. This will determine the number of steps before a robot processes

a prediction of the wheel type it has and apply the necessary motion constraint to its particles. Please enter 100 to replicate this project.

Step 3: A file titled, "After motion model (Number of steps you selected from Step 1) steps particle tracking auto.csv" will be available after Step 2 is done running.

Step 4: Run the "particle tracker after motion model".ipynb file in a Jupyter Environment

Step 5: Two plots of the average distance between the robot and each particle for before the motion model was added and for after the motion model was added will be plotted. The average distance will be shown after each plot. We will see that the distance for the after the motion model was added is lesser than the distance before the motion model was added. This shows that the particles were constrained and moved in close proximity with the robot within the space, especially for moments when a landmark was not visible.

Step 6: Run the "robot_driver.py" file from the command line to interact with the live robot environment and see the particles converge around the robot as you navigate it through the environment. Use keys 'a', 'd' to rotate the robot clockwise and anti-clockwise, and the keys 'w','s' to move the robot forward and backwards. It does not require a number of steps to run. You can remove the set seed on line 60 to generate a new environment on every run.

Please see screenshots below

2.1 Datasets

2.1.1 Phase 1 - Initializing the robot

- Step 1: Run '*robot driver auto before motion model*'.py from the command line by typing "**python3 'robot driver auto before motion model'.py -steps 100**" into the command line. You can enter 5, 10, 20, 50 or 100 only as input steps only. Please enter 100 to replicate this project.
- Step 2: A file titled, "*Before motion model (NUMBER OF STEPS YOU SELECTED FROM STEP 1) steps particle tracking auto.csv*" will be available after Step 1 is done running.
- Step 3: Run the "*particle tracker before motion model*".ipynb file in a Jupyter Environment
- Step 4: A plot of the average distance between the robot and each particles will be plotted. The aim of this research is to get an average number that is lesser than this average.

3 Data Pre-processing.

The data generated by the robot wheel generator program is read in from comma separated values into pandas dataframes. We check the balance of the classes and see if we have missing values.

```

In [1]: 1 import numpy as np
        2 import csv
        3 import cv2 as cv

In [2]: 1 np.random.seed(16)
        2
        3 steps = [5, 10, 20, 50, 100]
        4 #steps = [500, 1000]
        5 #no_of_robot_steps = 5
        6
        7 for no_of_robot_steps in steps:
        8     all_fixed_wheel_sequences = { 'xs' : np.ones(no_of_robot_steps),
        9                                 'ys' : np.ones(no_of_robot_steps),
        10                                'theta' : np.ones(no_of_robot_steps)
        11                                }
        12
        13     for i in range(0,1000):
        14         x_initial = np.random.rand() * 750
        15         random_slope = np.random.rand() # m
        16         random_intersection = np.random.rand() * 350 # c value
        17         #print("x_initial: ", x_initial, "random_slope: ", random_slope, "random_intersections: ")
        18
        19
        20         x_vals_updates = x_initial + np.cumsum((np.random.rand(no_of_robot_steps) * 5) - (np.r
        21         y_vals_updates = (random_slope * x_vals_updates) + random_intersection
        22         theta_updates = np.ones(no_of_robot_steps) * np.arctan(random_slope)
        23         #print("x_vals: ", x_vals_updates, "y_vals: ", y_vals_updates, "theta_updates ", theta
        24
        25         all_fixed_wheel_sequences['xs'] = np.vstack((all_fixed_wheel_sequences['xs'], x_vals_u
        26         all_fixed_wheel_sequences['ys'] = np.vstack((all_fixed_wheel_sequences['ys'], y_vals_u
        27         all_fixed_wheel_sequences['theta'] = np.vstack((all_fixed_wheel_sequences['theta'], th
        28
        29         #print("fw xs", all_fixed_wheel_sequences['xs'])
        30         #print("fw ys", all_fixed_wheel_sequences['ys'])
        31         #print("fw theta", all_fixed_wheel_sequences['theta'])
        32
        33
        34         np.savetxt(str(no_of_robot_steps) + "_steps_fixed_thetas.csv", all_fixed_wheel_sequences['
        35         np.savetxt(str(no_of_robot_steps) + "_steps_fixed_xs.csv", all_fixed_wheel_sequences['xs'])
        36         np.savetxt(str(no_of_robot_steps) + "_steps_fixed_ys.csv", all_fixed_wheel_sequences['ys'])
        37
        38         #####Omni Wheels#####
        39

```

Figure 1: Data generation 1

4 Data Mining

4.1 Classifier Analysis

We compare four classifiers. We consider Logistics Regression, KNN, Decision Trees and Random Forest.

4.2 Hyper-parameter tuning

The following models had their hyper-parameters tuned to find the best prediction scenario. Log loss was compared with accuracy and visual observation of confusion matrix to select the winning model.

4.3 Run robot auto sequence

References

```

1001 1001 1001
1001 1001 1001

In [4]: 1 np.random.seed(64)
2
3
4 # fixed wheel
5
6 steps = [5, 10, 20, 50, 100]
7 #steps = [500, 1000]
8 #no_of_robot_steps = 5
9
10 for no_of_robot_steps in steps:
11     all_fixed_wheel_sequences = { 'xs' : np.ones(no_of_robot_steps),
12                                 'ys' : np.ones(no_of_robot_steps),
13                                 'theta' : np.ones(no_of_robot_steps)
14                                 }
15
16     for i in range(0,1000):
17         x_initial = np.random.rand() * 750
18         random_slope = np.random.rand() # m
19         random_intersection = np.random.rand() * 350 # c value
20         #print("x_initial: ", x_initial, "random_slope: ", random_slope, "random_intersection:
21
22
23         x_vals_updates = x_initial + np.cumsum((np.random.rand(no_of_robot_steps) * 5) - (np.r
24         y_vals_updates = (random_slope * x_vals_updates) + random_intersection
25         theta_updates = np.ones(no_of_robot_steps) * np.arctan(random_slope)
26         #print("x_vals: ", x_vals_updates, "y_vals: ", y_vals_updates, "theta_updates ", theta
27
28         all_fixed_wheel_sequences['xs'] = np.vstack((all_fixed_wheel_sequences['xs'], x_vals_u
29         all_fixed_wheel_sequences['ys'] = np.vstack((all_fixed_wheel_sequences['ys'], y_vals_u
30         all_fixed_wheel_sequences['theta'] = np.vstack((all_fixed_wheel_sequences['theta'], tr
31
32         #print("fw xs", all_fixed_wheel_sequences['xs'])
33         #print("fw ys", all_fixed_wheel_sequences['ys'])
34         #print("fw theta", all_fixed_wheel_sequences['theta'])
35
36
37     np.savetxt(str(no_of_robot_steps) + "_steps_fixed_thetas_test.csv", all_fixed_wheel_sequer
38     np.savetxt(str(no_of_robot_steps) + "_steps_fixed_xs_test.csv", all_fixed_wheel_sequences[
39     np.savetxt(str(no_of_robot_steps) + "_steps_fixed_ys_test.csv", all_fixed_wheel_sequences[
40
41     #####Omni Wheels#####
42

```

Figure 2: Data generation Test for over-fitting dataset

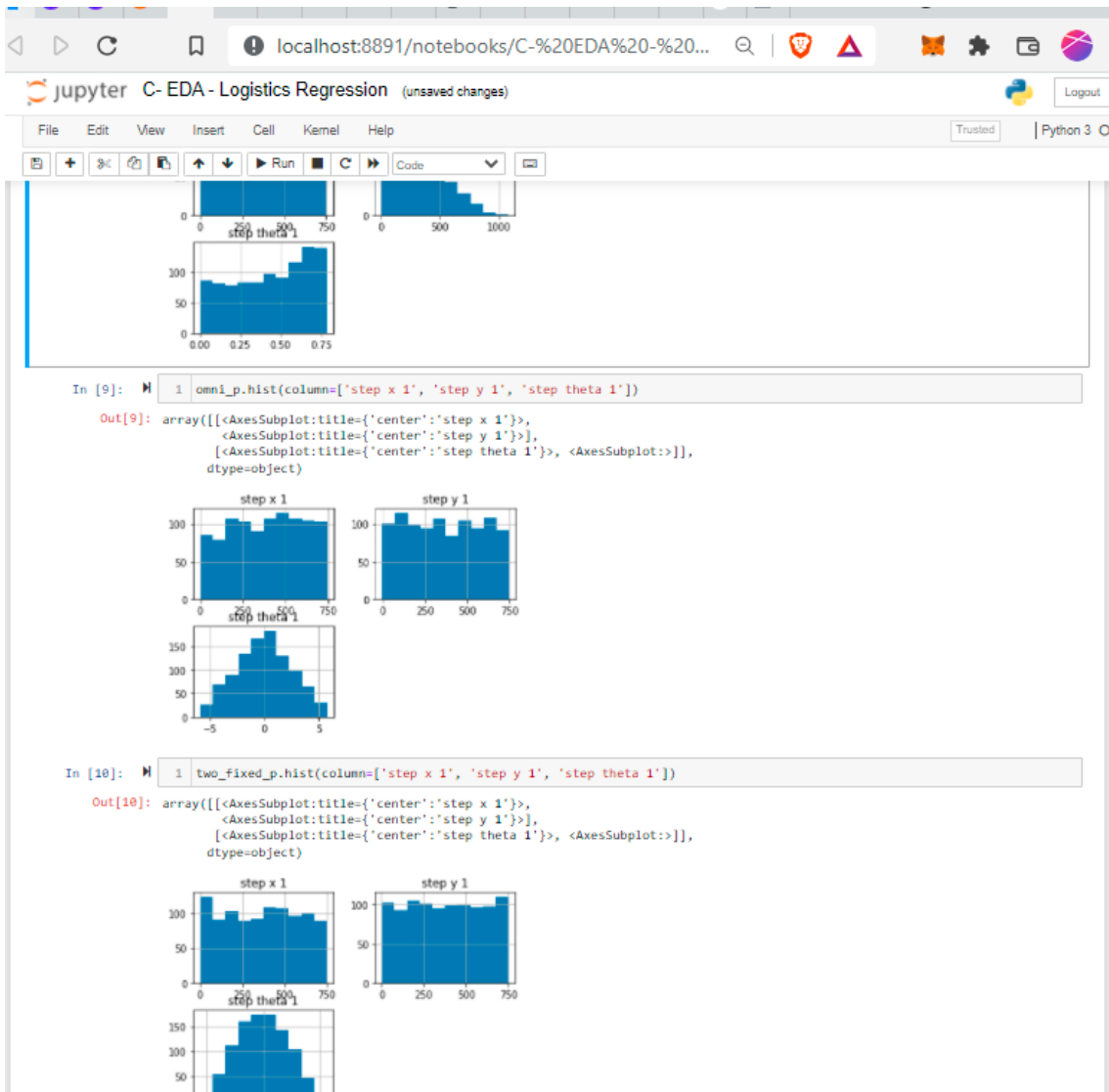


Figure 3: Exploratory Data Analysis


```
1001 1001 1001
1001 1001 1001

In [4]: 1 np.random.seed(64)
2
3
4 # fixed wheel
5
6 steps = [5, 10, 20, 50, 100]
7 #steps = [500, 1000]
8 #no_of_robot_steps = 5
9
10 for no_of_robot_steps in steps:
11     all_fixed_wheel_sequences = { 'xs' : np.ones(no_of_robot_steps),
12                                 'ys' : np.ones(no_of_robot_steps),
13                                 'theta' : np.ones(no_of_robot_steps)
14                                 }
15
16     for i in range(0,1000):
17         x_initial = np.random.rand() * 750
18         random_slope = np.random.rand() # m
19         random_intersection = np.random.rand() * 350 # c value
20         #print("x_initial: ", x_initial, "random_slope: ", random_slope, "random_intersection:
21
22
23         x_vals_updates = x_initial + np.cumsum((np.random.rand(no_of_robot_steps) * 5) - (np.r
24         y_vals_updates = (random_slope * x_vals_updates) + random_intersection
25         theta_updates = np.ones(no_of_robot_steps) * np.arctan(random_slope)
26         #print("x_vals: ", x_vals_updates, "y_vals: ", y_vals_updates, "theta_updates ", theta
27
28         all_fixed_wheel_sequences['xs'] = np.vstack((all_fixed_wheel_sequences['xs'], x_vals_u
29         all_fixed_wheel_sequences['ys'] = np.vstack((all_fixed_wheel_sequences['ys'], y_vals_u
30         all_fixed_wheel_sequences['theta'] = np.vstack((all_fixed_wheel_sequences['theta'], tr
31
32         #print("fw xs", all_fixed_wheel_sequences['xs'])
33         #print("fw ys", all_fixed_wheel_sequences['ys'])
34         #print("fw theta", all_fixed_wheel_sequences['theta'])
35
36
37     np.savetxt(str(no_of_robot_steps) + "_steps_fixed_thetas_test.csv", all_fixed_wheel_sequer
38     np.savetxt(str(no_of_robot_steps) + "_steps_fixed_xs_test.csv", all_fixed_wheel_sequences[
39     np.savetxt(str(no_of_robot_steps) + "_steps_fixed_ys_test.csv", all_fixed_wheel_sequences[
40
41     #####Omni Wheels#####
42
```

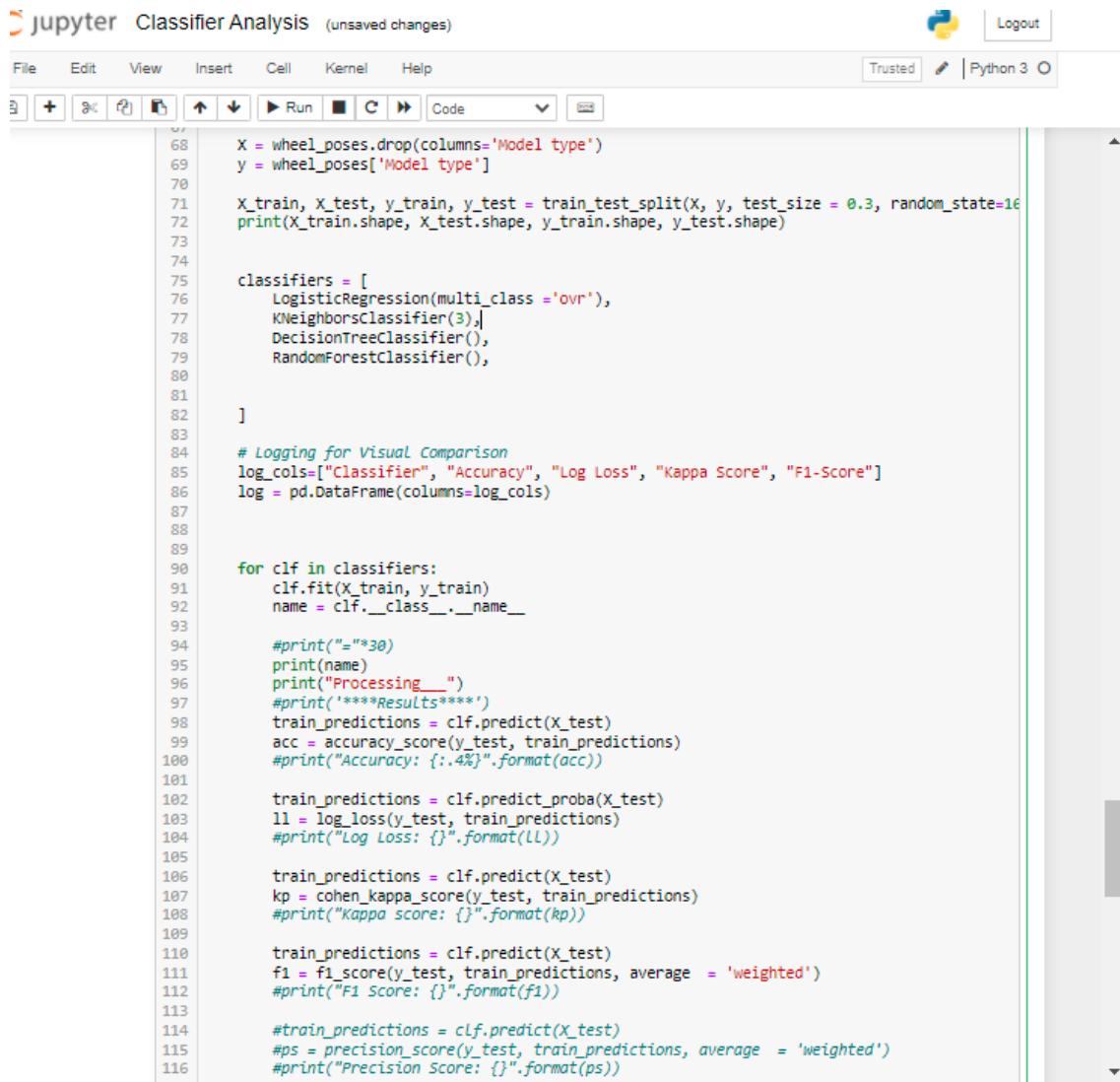
Figure 4: Check for data imbalance

```

2 def get_step_results(no_of_robot_steps):
3     fixed_wheel_x = np.loadtxt(str(no_of_robot_steps) + "_steps_fixed_xs.csv", delimiter=',')
4     fixed_wheel_y = np.loadtxt(str(no_of_robot_steps) + "_steps_fixed_ys.csv", delimiter=',')
5     fixed_wheel_theta = np.loadtxt(str(no_of_robot_steps) + "_steps_fixed_thetas.csv", delimit
6
7     omni_wheel_x = np.loadtxt(str(no_of_robot_steps) + "_steps_omni_xs.csv", delimiter=',')
8     omni_wheel_y = np.loadtxt(str(no_of_robot_steps) + "_steps_omni_ys.csv", delimiter=',')
9     omni_wheel_theta = np.loadtxt(str(no_of_robot_steps) + "_steps_omni_thetas.csv", delimiter
10
11     two_wheel_x = np.loadtxt(str(no_of_robot_steps) + "_steps_two_fixed_xs.csv", delimiter=',')
12     two_wheel_y = np.loadtxt(str(no_of_robot_steps) + "_steps_two_fixed_ys.csv", delimiter=',')
13     two_wheel_theta = np.loadtxt(str(no_of_robot_steps) + "_steps_two_fixed_thetas.csv", delin
14
15
16     len_of_steps = len(fixed_wheel_x[0])
17     total_len_of_columns = len_of_steps * 3
18     total_pose_size = len(fixed_wheel_x.flatten()) + len(fixed_wheel_y.flatten()) + len(fixed_
19     total_pose_size = int(total_pose_size/total_len_of_columns)
20
21     print("Number of robot steps: ", len_of_steps, "\nLenght of columns for steps (x,y,theta):
22
23     fixed_wheel_poses = np.column_stack((fixed_wheel_x.flatten(),fixed_wheel_y.flatten(),fixed
24     fixed_wheel_poses = np.reshape(fixed_wheel_poses,(total_pose_size,total_len_of_columns))
25
26
27
28     omni_wheel_poses = np.column_stack((omni_wheel_x.flatten(),omni_wheel_y.flatten(),omni_whe
29     omni_wheel_poses = np.reshape(omni_wheel_poses,(total_pose_size,total_len_of_columns))
30
31
32     two_fixed_wheel_poses = np.column_stack((two_wheel_x.flatten(),two_wheel_y.flatten(),two_w
33     two_fixed_wheel_poses = np.reshape(two_fixed_wheel_poses,(total_pose_size,total_len_of_col
34
35     x_headings = ["step x " + str(i+1) for i in range(int(total_len_of_columns/3))]
36     y_headings = ["step y " + str(i+1) for i in range(int(total_len_of_columns/3))]
37     theta_headings = ["step theta " + str(i+1) for i in range(int(total_len_of_columns/3))]
38
39     headings = []
40     for i in range(0,len(x_headings)):
41         headings.append(x_headings[i])
42         headings.append(y_headings[i])
43         headings.append(theta_headings[i])
44     #headings = x_headings + y_headings + theta_headings
45     #headings
46
47     fixed_p = pd.DataFrame(fixed_wheel_poses, columns = headings)
48     omni_p = pd.DataFrame(omni_wheel_poses, columns = headings)
49     two_fixed_p = pd.DataFrame(two_fixed_wheel_poses, columns = headings)
50
51     #Add Class Labels

```

Figure 5: Importing data for classifier analysis



```
67
68 X = wheel_poses.drop(columns='Model type')
69 y = wheel_poses['Model type']
70
71 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state=16)
72 print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
73
74
75 classifiers = [
76     LogisticRegression(multi_class='ovr'),
77     KNeighborsClassifier(3),
78     DecisionTreeClassifier(),
79     RandomForestClassifier(),
80
81 ]
82
83
84 # Logging for Visual Comparison
85 log_cols=["Classifier", "Accuracy", "Log Loss", "Kappa Score", "F1-Score"]
86 log = pd.DataFrame(columns=log_cols)
87
88
89
90 for clf in classifiers:
91     clf.fit(X_train, y_train)
92     name = clf.__class__.__name__
93
94     #print("-"*30)
95     print(name)
96     print("Processing...")
97     #print("****Results****")
98     train_predictions = clf.predict(X_test)
99     acc = accuracy_score(y_test, train_predictions)
100    #print("Accuracy: {:.4%}".format(acc))
101
102    train_predictions = clf.predict_proba(X_test)
103    ll = log_loss(y_test, train_predictions)
104    #print("Log Loss: {}".format(ll))
105
106    train_predictions = clf.predict(X_test)
107    kp = cohen_kappa_score(y_test, train_predictions)
108    #print("Kappa score: {}".format(kp))
109
110    train_predictions = clf.predict(X_test)
111    f1 = f1_score(y_test, train_predictions, average = 'weighted')
112    #print("F1 Score: {}".format(f1))
113
114    #train_predictions = clf.predict(X_test)
115    #ps = precision_score(y_test, train_predictions, average = 'weighted')
116    #print("Precision Score: {}".format(ps))
117
```

Figure 6: Classifier analysis

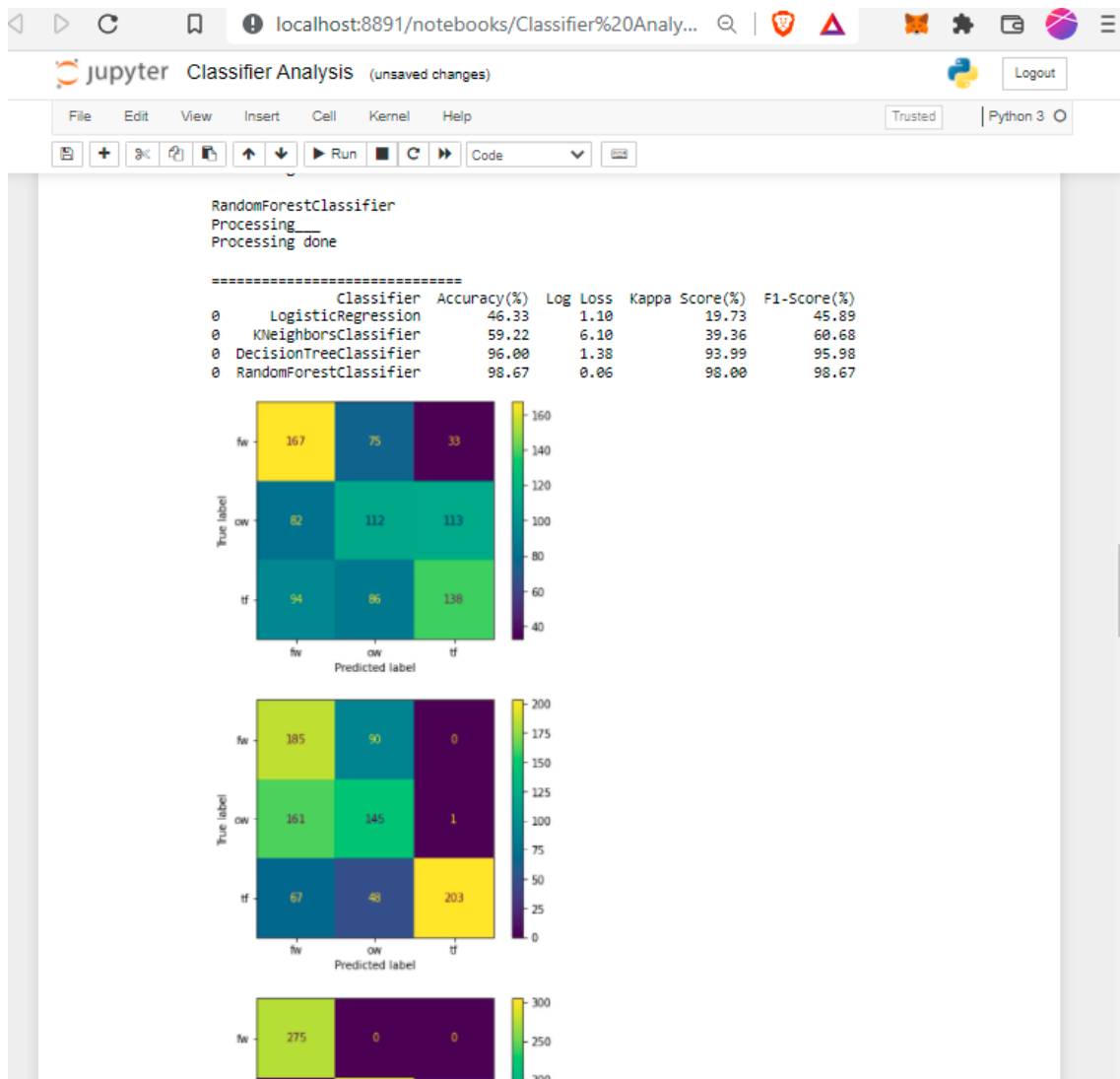


Figure 7: Result of Classifier analysis

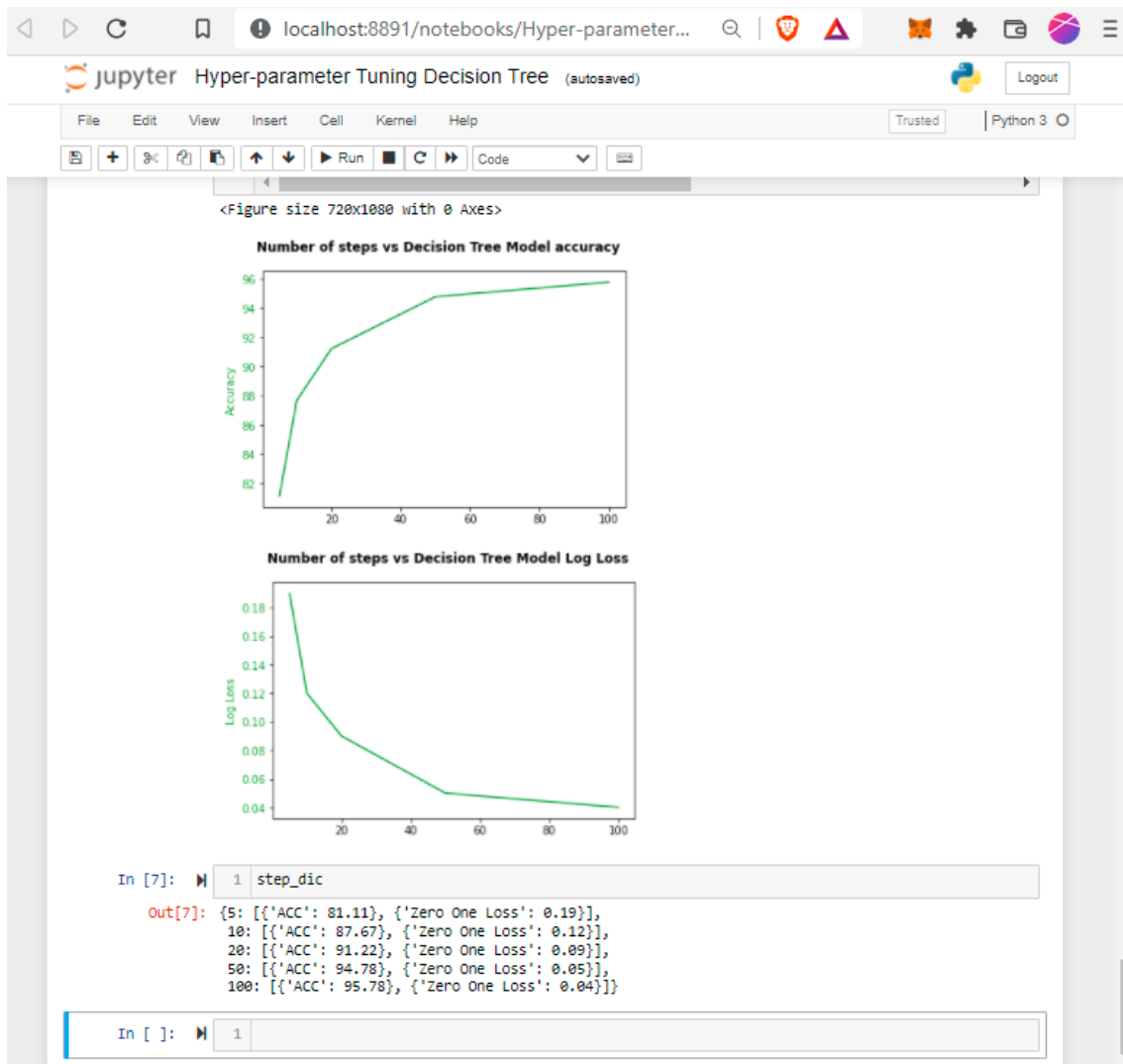


Figure 8: DecisionTree tuned result

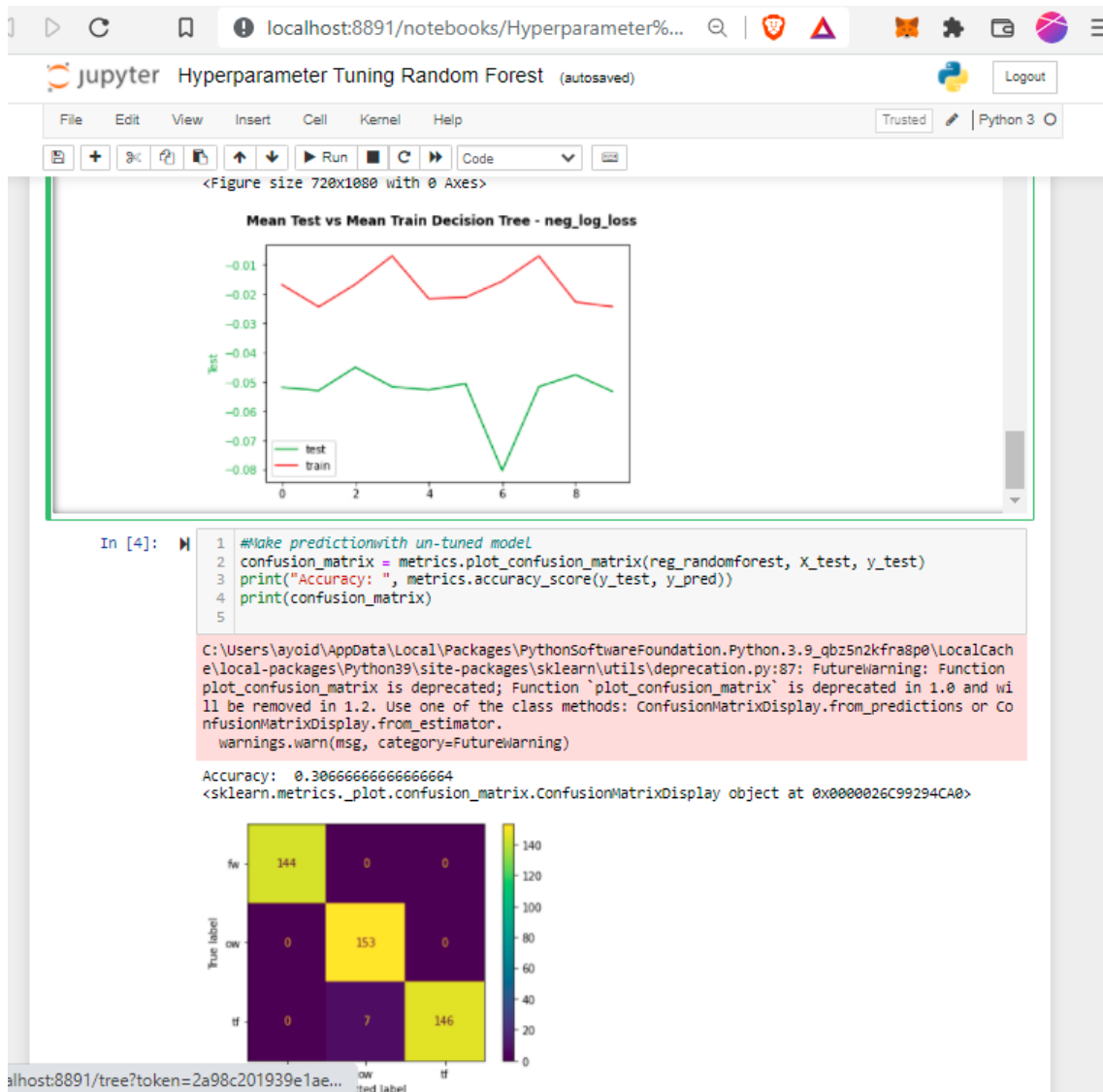


Figure 9: RandomForest tuned result

```
chosen pf model bb4 prediction: ow
key fw is 115
len of 2nd visible_landmarks is 2
received pf model in mm is: ow
model ow in use
chosen rb model b4 prediction: fw
chosen pf model bb4 prediction: ow
key fw is 115
len of 2nd visible_landmarks is 2
received pf model in mm is: ow
model ow in use
chosen rb model b4 prediction: fw
chosen pf model bb4 prediction: ow
key fw is 119
len of 2nd visible_landmarks is 2
received pf model in mm is: ow
model ow in use
chosen rb model b4 prediction: fw
chosen pf model bb4 prediction: ow
key fw is 115
len of 2nd visible_landmarks is 2
received pf model in mm is: ow
model ow in use
chosen rb model b4 prediction: fw
chosen pf model bb4 prediction: ow
key fw is 115
len of 2nd visible_landmarks is 2
received pf model in mm is: ow
model ow in use
```

Figure 10: Generating 100 robot steps

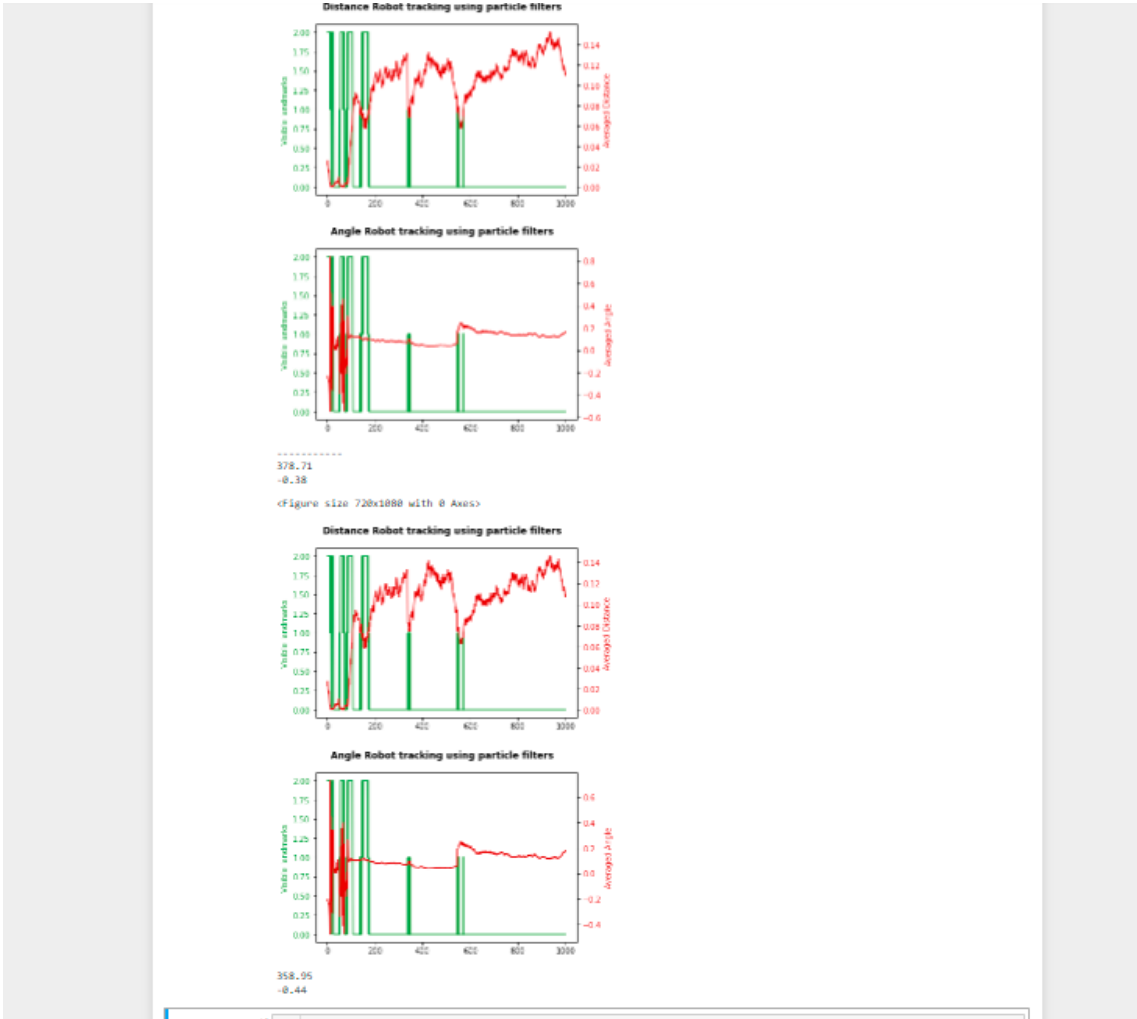


Figure 11: Comparing average length from before and after

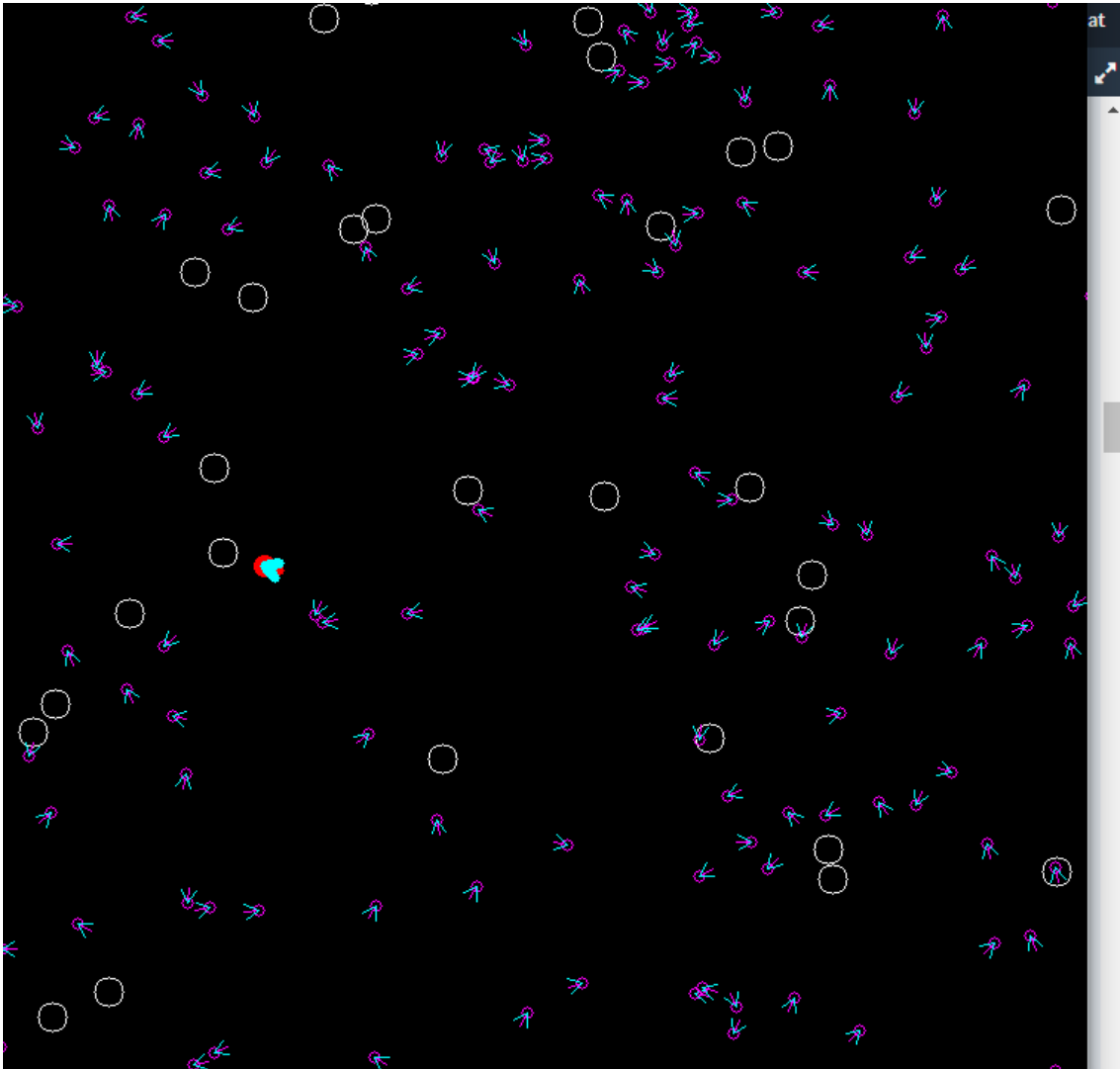


Figure 12: Robot world