BTP Report: Learning feedback law for time optimal control of the Game of Two Cars

Shrinivas Prasad Kulkarni, 180070057
Guide: *Prof. Debraj Chakraborty*

June 8, 2024

1 Introduction

A Dubins car is the name given to a point which has the ability to move in a plane with a constraint on its maximum speed, as well as a minimum turning radius. We study the problem of finding optimal trajectories of a pursuer and an evader, both being Dubins cars, given the conditions that the pursuer tries to catch the evader (get into close proximity to the evader) as soon as possible, and the evader tries to avoid this for as much time as possible. This problem can be solved using numerical methods, after a huge amount of computation. Since the problem can be viewed as a control problem, we train Random Forest classifiers for learning the optimal feedback laws for both the pursuer and the evader, and use them as agents in the simulation of the game from various initial states. Before tackling the aforementioned problem, we address a simpler problem, which is finding the optimal path minimizing the time taken by a Dubins Car to reach any destination using a Decision Tree classifier. The smaller problem was used to test the suitability of the ML models for our main problem.

2 Preliminaries and Notations

The kinematic equations of Dubins vehicle are given by:

$$\dot{x}(t) = v(t)\cos\left(\theta(t)\right) \tag{1}$$
$$\dot{y}(t) = v(t)\sin\left(\theta(t)\right) \tag{2}$$
$$\dot{\theta}(t) = v(t)w(t) \tag{3}$$

$x(t)$ denotes the x position and $y(t)$ denotes the y position of the vehicle at time instant t. Furthermore, the Dubins vehicle has an orientation (denoted by $\theta(t)$) and can move only in the direction of its orientation. The orientation $\theta(t)$ is measured in anti-clockwise direction with respect to the $x-$axis as shown in Figure 1.

The combined state vector is $\mathbf{x} := [x(t),\ y(t),\ \theta(t)]$.
Define $\mathbf{f}(t) := [v(t)\cos\left(\theta(t)\right),\ v(t)\sin\left(\theta(t)\right), v(t)w(t)]$. The vehicle cannot turn instantaneously, and the rate at which its orientation $\theta(t)$ changes depends on the applied angular speed $w(t)$ according to (3). The angular speed is bounded, $w(t) \in [-w_m, w_m]$. Also, the speed of the

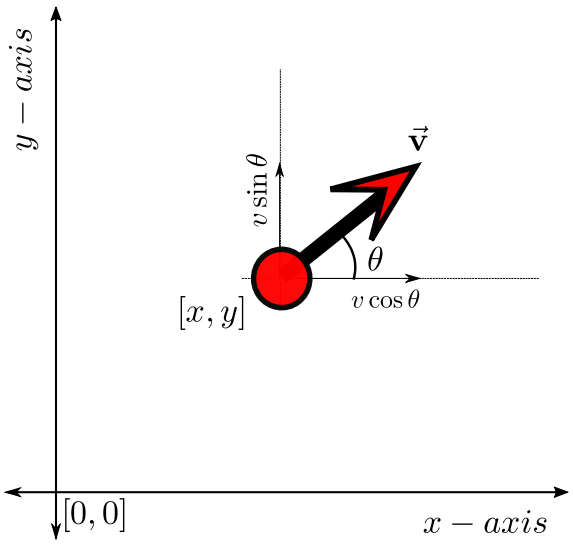


Figure 1: Dubins vehicle

vehicle is denoted by $v(t)$ and is bounded as well, i.e $v(t) \in [0, v_m]$. The angular speed and the forward velocity are considered as the inputs to the Dubins vehicle.

If the pursuer (evader) sets $w(t) = w_m$, then it moves along a circle of radius $r_m = 1/w_m$ in anti-clockwise direction (called anti-clockwise circle hereafter). If it applies an input of $w(t) = -w_m$ then it moves in clockwise direction along the circle of radius r_i (called the clockwise circle hereafter). A path obtained by concatenating a part of a circle of radius r_m with a straight line tangent to the circle will be called a CS (Circle and straight line) type of path. Thus there can be two types of CS paths : RS (Right Circle - Straigh Line) and LS (Left Circle - Straight Line). Similarly, concatenating an arc of anticlockwise circle and clockwise circle gives CC (Circle - Circle) type of paths. There can exist two types of CC paths: RL (Right Circle - Left Circle) and LR (Left Circle - Right Circle).

3 Modelling the Game of Two Cars

3.1 The Problem

Pursuer P and an evader E follow the equations:

$$\dot{x}_i(t) = v_i(t)\cos\theta_i(t) \tag{4}$$
$$\dot{y}_i(t) = v_i(t)\sin\theta_i(t) \tag{5}$$
$$\dot{\theta}_i(t) = v_i(t)w_i(t) \tag{6}$$

where $i \in \{p, e\}$. The state vector of the pursuer is denoted by $\mathbf{p}(t) = [x_p(t),\ y_p(t),\ \theta_p(t)]$ and that of the evader is denoted by $\mathbf{e}(t) = [x_e(t),\ y_e(t),\ \theta_e(t)]$. The combined state vector of the pursuer and evader is denoted by $\mathbf{x_{pe}}(t) = [\mathbf{x_p}(t),\ \mathbf{x_e}(t)] \in R^6$.
Furthermore, let $\mathbf{f_p}(t) := [v_p(t)\cos(\theta_p(t)),\ v_p(t)\sin(\theta_p(t)),\ v_p(t)w_p(t)]$
and $\mathbf{f_e}(t) := [v_e(t)\cos(\theta_e(t)),\ v_e(t)\sin(\theta_e(t)),\ v_e(t)w_e(t)]$. Let the maximum angular speeds of the pursuer (evader) be denoted by $w_{p_m}(w_{e_m})$ and $v_{p_m}(v_{e_m})$ respectively. We say the capture

occurs at time T_c (the capture time) if the separation between pursuer and evader is less than a predefined bound. The pursuer aims to capture the evader in minimum time and wants to design a feedback control policy $\mathbf{u_p} : \mathbf{R}^+ \to \mathbf{R}^2$ ($\mathbf{u_p}(t) = [v_p(t),\ w_p(t)]$) to achieve the same. The evader aims to avoid capture for as long as possible and designs a feedback control policy $\mathbf{u_e} : \mathbf{R}^+ \to \mathbf{R}^2$ ($\mathbf{u_e}(t) = [v_e(t),\ w_e(t)]$) to capture the evader. We wish to find the paths that they will take ideally. Since the pursuer and evader have the opposite objectives, the game becomes zero sum.

3.2 Numerical Solution

Numerous numerical methods exist for solving differential games, but here we briefly describe the algorithm used later to generate the data for training our models. The numerical procedure to solve the pursuit evasion problems involves solving two sub-problems.

Pursuer's problem ($\mathbf{P}$) :

$$\begin{aligned} &\min_{\mathbf{u_p}, T_c} q(\mathbf{p}, \mathbf{e^0}, T_c) \\ &\dot{\mathbf{p}}(t) = f_p(\mathbf{p}(t), \mathbf{u_p}) \qquad \mathbf{p}(0) = \mathbf{p_0} \\ &l(\mathbf{p}(T_c), \mathbf{e}(T_c), T_c) \leq l_c \end{aligned}$$

where $\mathbf{e^0}$ is a fixed trajectory of the evader. The solution of the pursuer's problem is the pursuer trajectory $\tilde{\mathbf{p}}$ and the capture time T_c^0. Let α_0 be the Lagrange multiplier associated with constraint $l(\mathbf{p}(T_c), \mathbf{e}(T_c), T_c) \leq l_c$.

Evader's problem ($\mathbf{E}$):

$$\begin{aligned} &\max_{\mathbf{u_e}} c^\top (\mathbf{e}(T_c^0) - \mathbf{e}^0(T_c^0)) \\ &\dot{\mathbf{e}}(t) = f_e(\mathbf{e}(t), \mathbf{u_e}) \qquad \mathbf{e}(0) = \mathbf{e_0} \end{aligned}$$

where

$$c := \frac{\partial}{\partial e} q(\tilde{\mathbf{p}}(T_c^0), \mathbf{e}^0(T_c^0), T_c^0) + \alpha_0 \frac{\partial}{\partial e} l(\tilde{\mathbf{p}}(T_c^0), \mathbf{e}^0(T_c^0), T_c^0)$$

Let $\mathbf{e^1}$ be the solution of evader's problem and $\mathbf{e}^1$ is defined in time interval $[0, T_c^0]$. The solution is extended in the time-interval $[T_c^0, T_c^0 + \Delta T_c^0]$ by using linear approximation

$$\mathbf{e^1}(T_c^0 + h) = \mathbf{e^1}(T_c^0) + \dot{\mathbf{e}}^{\mathbf{1}}(T_c^0)h, \qquad h \in [T_c^0, \Delta T_c^0]$$

The iterative procedure followed is described:

1. Solve $\mathbf{P}$ by taking a arbitrary trajectory of evader $\mathbf{e^0}$ and obtain T_c^0,$\mathbf{e^0}(T_c^0)$ and α^0. Let $i = 0$.
2. Solve the evader's problem $\mathbf{E}$ by using T_c^i,$\mathbf{e^i}(T_c^i)$ and α^i.
3. Extend the solution obtained $\mathbf{e^{i+1}}$ by using linear approximation.
4. Use the extended solution to solve $\mathbf{P}$ to obtain T_c^{i+1},$\mathbf{e}^{i+1}(T_c^{i+1})$,$\alpha^{i+1}$.
5. If $||\mathbf{e}^{i+1}(T_c^{i+1}) - \mathbf{e}^i(T_c^i)|| < \epsilon$, where ϵ is fixed threshold then stop the iteration. Else, go to Step 2.

4 Overview of the ML models used

We have used Decision Tree Classifier as well as Random Forest classifiers for our purpose, which are briefly explained here.

4.1 Decision Tree Classifier

A decision tree classifier is a supervised learning algorithm used for performing classification based on decision rules inferred from the training data. A trained DTC is essentially a complex structure of nested if-else statements, beginning from the root node (start), following a path according to the conditions at every node till a leaf node is reached. Visualization of a trained classifier takes the form of a tree. At each node (except the leaves) there is a decision rule which checks whether a feature follows come condition or not, based on which, for a probe (test) data point, the next node is decided, till a leaf is reached. Training a DTC essentially means choosing the right decision rules at the nodes.

4.2 Random Forest Classifier

The guiding principle of a Random Forest is almost the same as that of a Decision Tree, the difference being, a Random Forest has multiple decision trees trained over randomly chosen samples of the data. The prediction (output) of the forest is the label which is predicted by a majority of its trees. Furthermore, randomness is introduced by choosing the optimal split feature from a subset of the features and not all the features, and this decision rule could be sub-optimal. But since there are a large number of trees, it is empirically observed that this leads to more accurate results. Since a lot of trees are consulted for a prediction, and a majority of them point to one, it is intuitive that it is the correct one.

5 Data generation for the time optimal control of the Dubins car

In this section, the details about data generation for the case of a single car trying to minimise travel time to a destination (the single car case) would be provided.

5.1 Computation of time optimal control for single Dubins Vehicle

The data for training of the decision tree classifier is obtained by generating optimal policies for the Dubins vehicle problem using direct numerical optimal control. The optimal control problem is converted to a nonlinear optimization problem using an appropriate discretization scheme, e.g. trapezoidal integration. The sampling time interval (denoted by Δt) is kept uniform. Let N be the number of sampling intervals. Thus the optimization variables are $X = [x[1], \dots x[N], y[1], \dots, y[N], \theta[1], \dots \theta[N]]$, $U = [u[1], \dots u[N]]$, and sampling time Δt, where $x[i]$ denotes the value of state variable x at time instant $i\Delta t$. The goal is that the Dubins vehicle should reach the final point $\mathbf{z_f}$ as soon as possible (since the speed would be maximum, the shortest path would also be the least time consuming). Thus the constrained nonlinear

optimization problem is:

$$\min_{X,U,\Delta t} N\Delta t$$
$$\mathbf{x}[k+1] = \mathbf{x}[k] + \left(f(\mathbf{x}[\mathbf{k}+\mathbf{1}]) + f(\mathbf{x}[\mathbf{k}])\right)\Delta t/2$$
$$-w_m \leq w[k] \leq w_m$$
$$||[x[N],\ y[N]] - \mathbf{z_f}||_2 = 0$$

This nonlinear optimization problem was solved using Ipopt through the *JuMP* interface in *Julia Programming Language*. Since the interior point algorithm does not guarantee convergence to global optimality the following heuristics were used:

1. Let $t_{min} = 0$ and $t_{max} = 2||z_f||/v_m$ be the lower and upper bounds on the objective function $N\Delta t$.

2. If the solution returned by Ipopt is infeasible then increase t_{min}.

3. If feasible solution is returned by Ipopt then decrease t_{max}.

The parameters for the Dubins vehicle were selected to be: $v_m = 1$, $w_m = 1$. The number of sampling intervals used for transcription of the optimal control problem to the nonlinear optimization problem was $N = 200$.

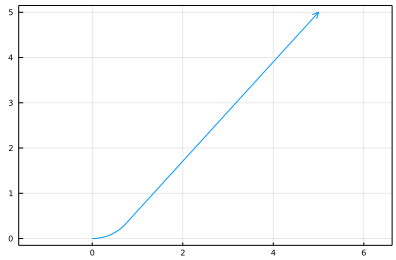
The initial position of the Dubins car was fixed at $[x(0), y(0), \theta(0)] = [0,\ 0,\ 0]$. The final position $\mathbf{z_f}$ was selected so as to generate data with a conscious effort to cover most of the points in the state-space, or at least spread out the data evenly checking any bias. The simulation result provides the optimal state trajectories $\mathbf{x}$, optimal inputs $\mathbf{u}$. Since, we have used 200 discretization intervals, and since the tail path of an optimal path is optimal, for each final position $\mathbf{z_f} = [x_f, y_f]$ we obtain 200 data points comprising of $\mathbf{X_i} := [x_f, y_f, x[i], y[i], \theta[i]]$ and $\mathbf{Y_i} := w(i)$ for $i \in \{1, 2, \ldots N\}$. Note that the inputs obtained from the simulations belonged to the set $\{-w_m, 0, +w_m\}$. The outputs of some sample simulations are plotted in Figure 2. 11.

5.2 Transformation of Data

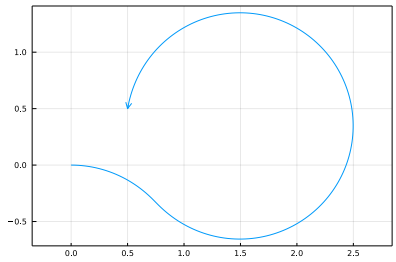
With an aim to reduce the dimensionality of the data, we transform the data to a frame of reference having origin fixed on the Dubins car and the $x - axis$ of the frame is aligned with the orientation of the car. The frame will be called vehicle frame hereafter. Since the frame is attached to the Dubins vehicle, the state of the Dubins vehicle in the new frame is always $[0, 0, 0]$. However, the final position $\mathbf{z_f}$ has to be transformed into new coordinates. Let the new coordinates be denoted by $\mathbf{z'_f}(t) = [x'_f,\ y'_f]$. This can be achieved by following transformation (shifting and rotating).

$$z'_f(t) = \begin{bmatrix} \cos(\theta(t)) & \sin(\theta(t)) \\ -\sin(\theta(t)) & \cos(\theta(t)) \end{bmatrix} \begin{bmatrix} x_f - x(t) \\ z_f - z(t) \end{bmatrix} \tag{7}$$

Thus a data point $\mathbf{X_i} = [x_f[i], y_f[i], x[i], y[i], \theta[i]]$ is transformed to $\mathbf{X'_i} = [x'_f[i], y'_f[i], 0, 0, 0]$. Since the zeros do not add any information the new data point is $\tilde{\mathbf{X}}_\mathbf{i}$ $=[x_f[i], y_f[i], w[i]]$. Thus by a change of coordinates the size of the feature vector is reduced.



(a) Numerical Simulation: $\mathbf{z_f} = [5, 5]$



(b) Numerical Simulation: $\mathbf{z_f} = [0.5, 0.5]$

Figure 2: Optimal paths for Single Dubin's Car

6 Model Training and Simulation for time optimal control of Dubins car

In this section, we train decision trees and random forests for learning the feedback law for the single car problem. It is known that the value function for the case is a discontinuous function with respect to the final position $\mathbf{z_f}$. The discontinuity occurs on the boundary of the minimum radius turning circles. Learning this discontinuity will require significantly higher number of data points. Hence, initially we train classifiers for the two cases separately.
We also define the following two terms as we move forward, for the two cases.

- **Near Case**: the final point is inside the minimum radius turning circles
- **Far Case**: the final point is outside the minimum radius turning circles

6.1 Decision Tree Classifier for the *Far case*

The destination in the vehicle frame, $\mathbf{z'_f}(t) = [x'_f(t), y'_f(t)]$ is the determining factor for the feedback law. The decision tree (denoted by DT) will essentially predict $w(t) \in \{1, -1\}$ for a given $\mathbf{z'_f}(t)$ i.e. $w(t) = DT(\mathbf{z'_f}(t))$. The data used for training the decision tree classifier (after transformation to the vehicle reference frame) is illustrated in Figure 3. In the figure there are many points which lie on the direction of the orientation of the vehicle. The numerical simulation naturally selects $w(t) = 0$ for these points, for once the final point is in the direct line of sight of the vehicle, it does not tend to turn. However, this is equivalent to $w(t)$ switching rapidly between $+1$ and -1 and having the average value of zero. Thus in order to simplify the training, and reduce the problem to binary classification, we remove the points which are on

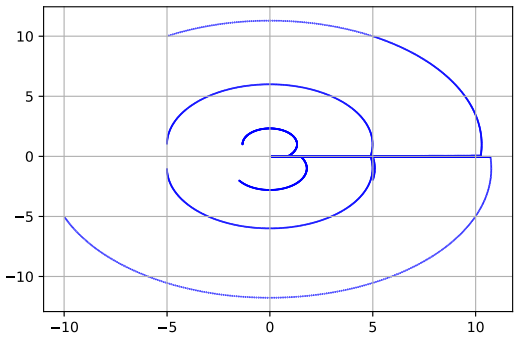


Figure 3: Data points for the *far* case

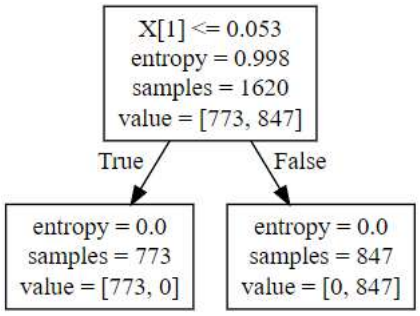


Figure 4: Visualization of the tree

the $x - axis$ in Figure 3. Hence out of the 6800 data points generated initially, what are left are merely 2025 points.

A decision tree classifier was trained using *Scikit-learn* with *Entropy* being used as the criterion for training. The maximum depth of the tree was set to 1 (just a root with leaves). The model was trained with 80% of data reserving 20% for testing the trained model. The accuracy of the resulting model turned out to be more than 99% in all the training iterations of reshuffling the data and re-training, many a times reaching 100%. The tree in Figure 4 is the visualization of the trained model, where $X[1]$ is the y coordinate, and $X[0]$ is the x coordinate of the destination.

6.2 Random forests for the *Near* case and *Far* cases

It was observed that a single decision tree classifier does not work well for the near case, so we trained a random forest classifier for the Near case. The scatter plot of the data (after transformation to the vehicles frame) for the near case is shown in Figure 5. The number of data points used for the near case were 6400, and the accuracy of the trained RFC was around 98%. The random forest classifier in the near case is denoted by F_n and $w(t) = F_n(\mathbf{z}'(t))$ such that $w(t) \in \{+1, -1\}$.

We also trained a random forest classifier for the far case. The data used to train the random forest classifier is shown in Figure 3. The accuracy was 100%. The random forest classifier in the Far case is denoted by F_f and $w(t) = F_f(\mathbf{z}'(t))$ such that $w(t) \in \{+1, -1\}$.

The model details for the feedback law of Dubins vehicle have been summarized in Table 1

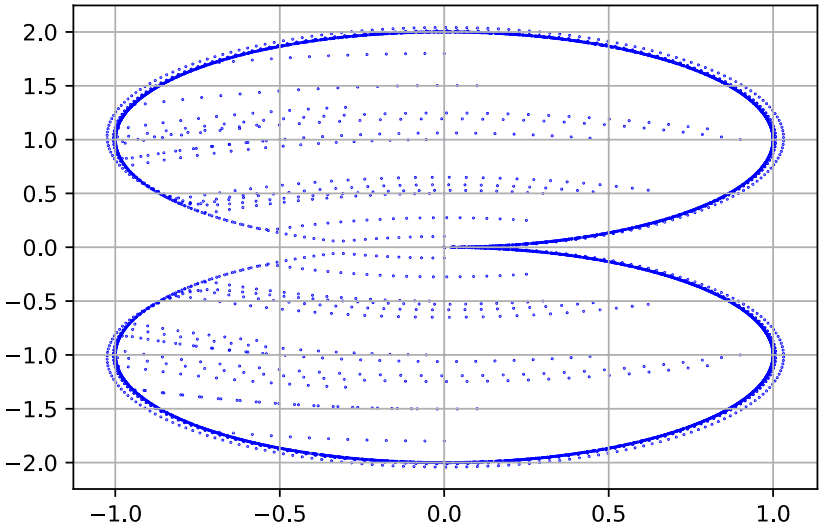


Figure 5: *Near* data

Table 1: Model parameters

(a) *Far* case

Classifier used	DTC	RFC
Training data points	1620	1620
Testing data points	405	405
Criterion	Entropy	Gini
Restriction	Depth = 1	Trees = 100
Accuracy	99.51%	100%

(b) *Near* case

Classifier used	RFC
Training data points	5120
Testing data points	1280
Criterion	Gini
Restriction	Trees = 100
Accuracy	98.67%

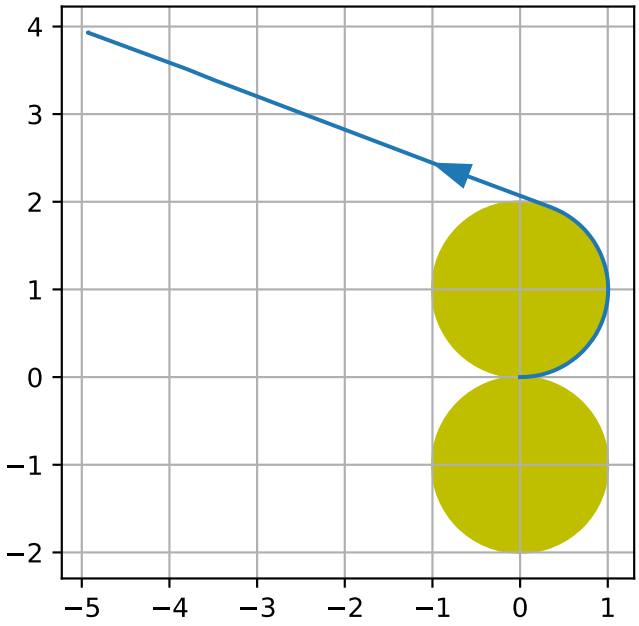


Figure 6: Far case: $\mathbf{z_f} = [-5,\ 4]$

6.3 Simulations for the time-optimal control of Dubins vehicle

Since separate classifiers were trained for the *Near* and the *Far* case, a correct classifier needs to be selected based on the final position $\mathbf{z_f}$ to be reached. The feedback law using two classifiers can be made precise as follows.

1. Take $x(t), z_f$ as inputs
2. Transform the final state $\mathbf{z_f}$ into the vehicle reference frame $\mathbf{z'_f}$ by using the aforementioned transformation
3. If $\mathbf{z'_f}$ is inside the clockwise circle or the anti-clockwise circle:
$$w(t) = F_n(\mathbf{z}'(t))$$
4. If $\mathbf{z'_f}$ is inside the clockwise circle or the anti-clockwise circle:
$$w(t) = F_f(\mathbf{z}'(t))$$

The computed optimal trajectories are shown in Figures 6 and 7. In the figures the solid circles represent the clockwise and anticlockwise circles of Dubins vehicle.

7 Data generation for the game of two cars

7.1 Computation of optimal trajectories for the pursuer and the evader

In this section we solve the game of two cars using Random Forests, i.e. create models which predict the feedback laws. The data required to train the decision tree classifier is obtained

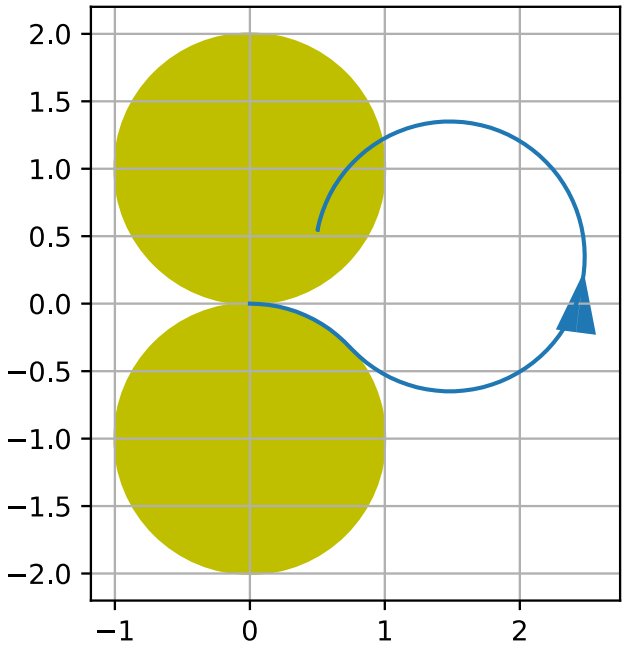


Figure 7: Near case: $\mathbf{z_f} = [0.5, 0.5]$

by numerically obtaining the solution to the pursuit-evasion game using the methodology described previously. The objective function for the game of two cars is $J = T_c$ and the final time constraint is $l_c(T_c) = (x_p(T_c) - x_e(T_c))^2 + (y_p(T_c) - y_e(T_c))^2$. The algorithm solves the Pursuer's problem and the Evader's problem iteratively, while The individual optimal control problems are solved using direct optimal control. The number of sampling time intervals for the pursuer's problem are N_p =200 while that for the evader's problem are N_e =200 as well. The simulations give us the state vector of the pursue and evader $\mathbf{x_{pe}}(t) \in R^6$ and the corresponding input $w_p(t)$ and $w_e(t)$. Since $N_p = N_e = 200$ each simulation gives us 200 sample points and the corresponding inputs. The following parameters were selected in order to perform the simulation: $v_{m_p} = 2$, $v_{m_e} = 1$, $w_m = w_{m_p} = w_{m_e} = 1$. The initial position of the pursuer was fixed at $\mathbf{p_0} = [0, 0, 0]$ while that of the evader was varied to generate a representative sample data. A representative simulation is shown in Figure 8. About 9000 data points were generated using numerical simulations, which were then used for training.

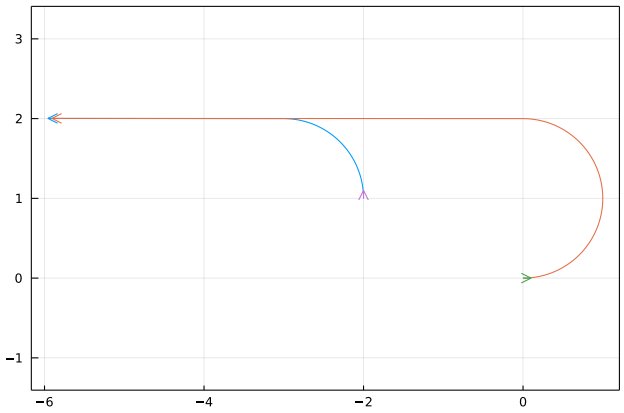


Figure 8: Pursuit-evasion numerical simulation: $\mathbf{p_0} = [0, 0, 0], \mathbf{e_0} = [-2, 1, \pi/2]$

7.2 Transformation of Data for pursuer

For training a classifier for the pursuer, the input feature vector can be reduced by using a suitable transformation, similar to the previous problem. We transform the data to the frame of reference whose origin is fixed on the pursuer and the $x - axis$ of the frame is aligned with the orientation of the pursuer. An important consideration is the orientation of the evader, which is now is measured with respect to the pursuers orientation in the new frame. This can be achieved by following transformation (shifting and rotating)

$$\mathbf{e}'(t) = \begin{bmatrix} \cos(\theta_p(t)) & \sin(\theta_p(t)) & 0 \\ -\sin(\theta_p(t)) & \cos(\theta_p(t)) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_e(t) - x_p(t) \\ y_e(t) - y_p(t) \\ \theta_e(t) - \theta_p(t) \end{bmatrix} \tag{8}$$

The state of the pursuer in the new frame is always $\mathbf{p'_0}$ =$[0, 0, 0]$, and the evader's position has to be transformed into new coordinates. Let the new coordinates be denoted by $\mathbf{e}'(t) = [x'_e,\ y'_e,\ \theta'_e]$. Thus the features of a data point $\mathbf{X_i} = [x_p[i], y_p[i], \theta_p[i], x_e[i], y_e[i], \theta_e[i]]$ are transformed to $\mathbf{X'_i} = [0, 0, 0, x'_e[i], y'_e[i], \theta'_e[i]]$. Removing the zeros, the new data point becomes $\tilde{\mathbf{X}}_\mathbf{i}$ =$[x'_e[i], y'_e[i], \theta'_e[i]]$. Thus by a change of coordinates the size of the feature vector is reduced. The training data for the pursuer after the transformation is shown in Figure 9.

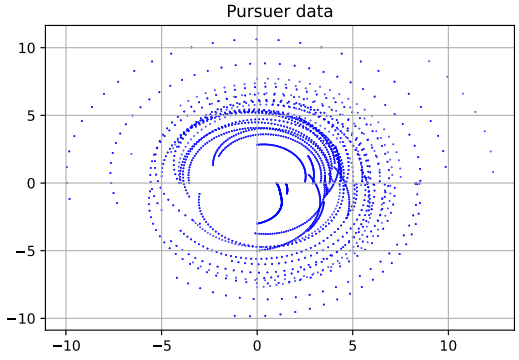


Figure 9: Training data in pursuer's frame of reference

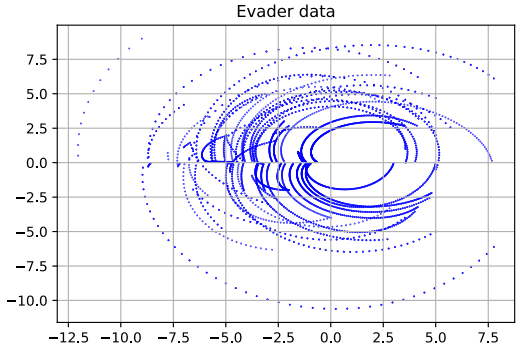


Figure 10: Training data in evader's frame of reference

7.3 Transformation of the data for the evader

Similar transformation is used to generate the data for evader. The pursuer's coordinates in the evader's frame of reference are denoted by $\mathbf{p}'(t)$ and the evader's coordinates are fixed at $[0, 0, 0]$. The data in evader's frame of reference is shown in Figure 10.

8 Training Random Forest Classifier for the game of two cars

Two random forest classifiers were trained one for the pursuer and one for the evader. The random forest classifier for the pursuer is denoted by F_p and $w_p(t) = F_p(\mathbf{e}'(t))$. Similarly, the random forest classifier for the evader is denoted by F_e and $w_e(t) = F_e(\mathbf{p}'(t))$. The models, when trained, gave a test accuracy over 99%.

8.1 Simulations

The feedback law calculation is made precise in the following steps.

1. Take $p(t), e(t)$ as inputs

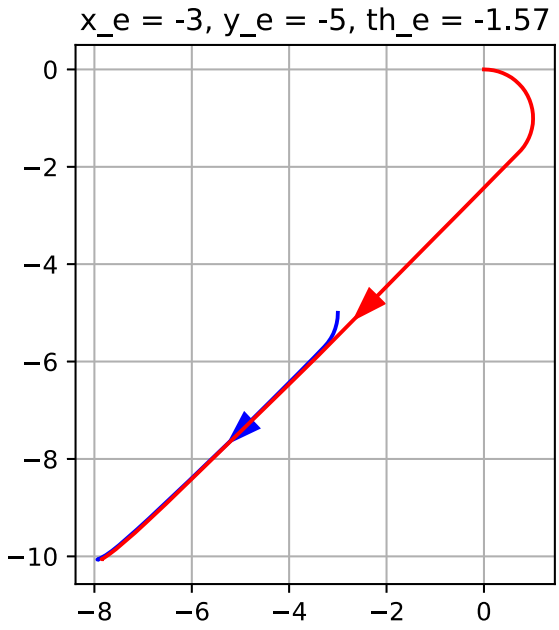


Figure 11: Simulation $\mathbf{p_0} = [0, 0, 0], \mathbf{e_0} = [-3, -5, -\pi/2]$

2. Transform the states to the pursuer's frame of reference and obtain $\mathbf{p'(t)}$ the transformation given in (7).

3. Transform the states to the evader's frame of reference and obtain $\mathbf{e'(t)}$ the transformation given in (7).

4. Compute the pursuer and evader inputs using random forest classifiers.

$$\begin{aligned} w_p(t) &= F_n(\mathbf{e'}(t)) \\ w_e(t) &= F_f(\mathbf{p'}(t)) \end{aligned}$$

The positions of both pursuer and evader were stored and plotted for various initial state variables, and are illustrated in Figure 11.

9 Conclusion

In this paper a **decision tree classifier** was trained to obtain feedback laws for time-optimal control of Dubins vehicle for the **Far case**. However, decision tree classifier could not be trained accurately for the Near case. Hence, a random forest classifier was trained for the Near case. The data for training was generated using numerical optimal control. Similarly, **random forest classifiers** were trained to design feedback laws for the agents in the game of two cars. We can extend the solution to the Dubins problem by adding an additional constraint of the **final orientation** being **fixed**. The trained models for the game of two cars do not solve the problem optimally in quite a few circumstances. Hence, we could, in future, try to understand the **pathological cases**, why are they pathological, and also try to train some model, or improve on the existing one to tackle them.

10 My Contributions

The report described above outlines the work done by myself as well as a fellow student, so I point out my contributions in the whole process in this section.
Overall, I contributed with the ML models, from selection of them, cleaning and transforming the data points, visualizing the data, training and visualization on the far case of the time optimality problem, training models for both the problems and generating feedback laws, simulating the results for both the problems using these trained models (The Game of Two cars and the time optimality problem). This involved a lot of feedback and collaboration with my fellow student, who performed numerical simulations and generated data, as the quality of the data generated plays a huge role in the working of the models. I was also involved in the numerical simulations, in collaborating and ensuring the simulations work efficiently enough for the task of data generation. Also, a lot of heuristics were involved, hyperparameters to be tuned, the ones for the single car problem are summarized in Table 1. I was also involved in writing the paper under the guidance of Prof. Debraj, and I included my work as well as results there.
Specifically, my contributions include sections 4, 5.2, 6, 7.2, 7.3, 8.