1 Problem statement

In today's day and age, as we as human kind advance, with us the transportation system advances rapidly. When it comes to driving, specifically in urban settings, the frequency of encountering traffic lights is high. Repeated stopping and accelerating leads to increased brake wear, fuel consumption and time spent on the road. The so called "greedy" driving approach only contributes to the number of abrupt stops and accelerations. According to a study from 2015 [1], vehicles spend additional 6.9 billion hours on the road, which costs an extra 3.1 billion gallons of fuel. Estimated cost of that comes out to about 160 billion dollars. We can clearly see that we are dealing with a problem that will, unless something is done, only get worse as time goes on.

"Rolling start" strategy presents one of the possible solutions. The idea is that the driver adjusts the approach speed to coincide with the traffic light turning green, which will result in the decrease of the number of abrupt stops. However, the actual implementation of this strategy proves to be rather difficult: if there are multiple vehicles, no way to know the exact time until light turns green or if we are relying on human brain alone to calculate the ideal solution, we can quickly identify the challenges of this possible strategy.

In this paper, we aim to develop an easy-to-follow, practical, real-life-friendly solution that will minimize stop-start events and optimize the speed at which the vehicle approaches the traffic light. We will explore different scenarios, such as when the traffic light timer is present, when it is not, and when there are multiple vehicles on the road. By refining driving rules, we hope to aid in solving this problem by trying to reduce the collective costs associated with urban driving inefficiencies while enhancing the flow and safety for all involved.

2 Task 1

What we are being asked in this part of the problem is to develop an optimal driving strategy for the case when we know the exact time until the traffic light turns green and we are the only vehicle on the road. With these conditions, most of the uncertainty has been removed, leaving us space to reach an exact solution.

Let's now define the notation we will use. First, let x(t) be a function of position in terms of time, v(t) function of velocity in terms of time and a(t) function of acceleration in terms of time. Also, let the starting amount of time left until the traffic light turns green be denoted as T, and the distance from traffic light at the start as d.

The boundary conditions of this system are as follows : $x(t) \in [0,d]$, $v(t) \in [0, v_{max}]$ and $a(t) \in [a_{min}, a_{max}]$. Also, we note that the following holds: x(0)=0, x(T)=d, $v(0) = v_{max}$.

Our goal is to minimize the amount of energy lost, which is the same as minimizing the work done while breaking. It is important to note that, in our system, maintaining current velocity does not contribute to energy loss. So, we have to minimize the following:

$$A = -m \int_0^T a'(t)v(t)\,dt$$

where m is the mass of the vehicle and we define a'(t) like this (Θ is the unit step function):

$$a'(t) = a(t)\Theta(-a(t))$$

In order for the vehicle to be able to stop before it reaches the traffic light, the following has to hold:

$$v_{max}T - \frac{a_{max}T^2}{2} > d$$

If $d > v_{max}T$ there will be no need to break, so we can now focus on the rest of the cases.

In order to solve an optimal control problem, we will write a generalized Hamiltonian of the system, with λ_1 and λ_1 being functions of time (so not always a constant):

$$H = -a'v + \lambda_1 v + \lambda_2 a$$

We know that:

$$\dot{\lambda}_1 = \frac{\partial H}{\partial x} \Rightarrow \lambda_1 = c, \quad \dot{\lambda}_2 = \frac{\partial H}{\partial v} = a\Theta(-a) - c$$

According to Pontryagin's minimum principle, the control a is chosen such that the Hamiltonian remains minimal. So now we have:

$$a(t) = \begin{cases} a_{max} & \lambda_2 < 0 \\ 0 & 0 \le \lambda_2 \le v \\ a_{min} & \lambda_2 > v \end{cases}$$

Transversality condition states that $\lambda_2(t) = 0$.

We can conclude that $a \geq 0$ in the area around T, meaning that

$$\dot{\lambda}_2 = -c$$

Respecting all the stated conditions, c must be greater than 0. Then, the optimal strategy is to break until some time τ (when we are near the traffic light and the driver has enough time to break), and then to continue with a constant velocity.

Taking into account the fact that fairly experienced drivers can estimate that point in time quite well, we can conclude that that is the best strategy.

However, we were interested to fully solve the problem and figure out the exact moment, so here it is using the equation of distance that is covered:

$$d = v_{max}\tau + \frac{a_{min}\tau^2}{2} + (v_{max} + a_{min}\tau)(T - \tau)$$

meaning that the exact value for τ is:

$$\tau = T - \sqrt{T^2 + \frac{2(v_{max}T - d)}{a_{min}}}$$

Graphs below represent the exact showcase of the ideal case scenario:

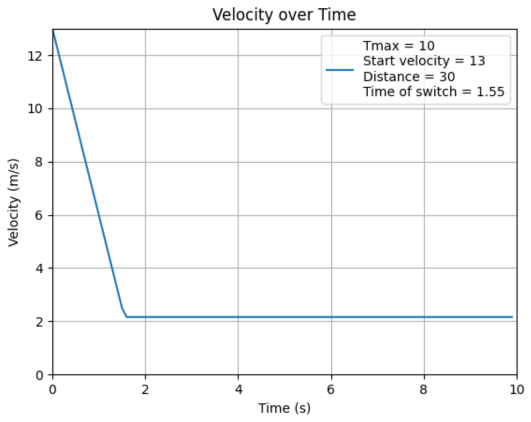


Figure 1: Graph of velocity over time

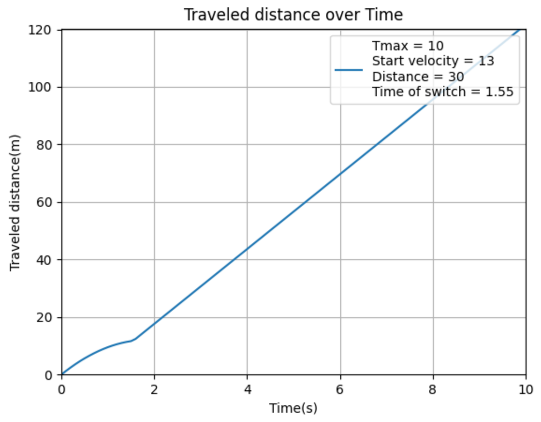


Figure 2: Graph of traveled distance over time

3 Task 2

Our solution to the last problem relied on the fact that we knew the exact time until traffic light turns green. The question arises, what if we don't know the exact time, just the bounds of its value (bounded by the maximal possible time until it turns green, which is the time difference between two green light appearances).

Unlike the first task, where we were able to find an analytical solution, for Task 2 we will find a numerical one.

We opted for using stochastic dynamic programming as our method to solve this problem.

Every point of a trajectory is described by a triplet (t, v, x). Taking into account the delay between actual event and human reaction (it is not instant), we will introduce Δt. This problem can be discretized: $(t, v, x) = (k\Delta t, i\Delta v, j\Delta x)$, where t, v and x are bounded.

What this means is that basically we can fit every possible trajectory in a 3D matrix and utilise Dynamic Programming.

Bellman's equation states the following:

$$J(k,i,j) = \min_a(\phi(a,v(i)) + \beta(k)J(k+1, \frac{V(i)+a\Delta t}{\Delta v}, \frac{x(j) - \frac{1}{2}a\Delta t^2 - v(i)\Delta t}{\Delta x}))$$

This system wants to minimize the lost energy, and the distribution of when the green light will turn red is always the same

$$\phi(a,v) = a\Delta t(v + \frac{1}{2}a\Delta t)$$

Chance that the traffic light will stay red in the moment k, denoted $\beta(k)$, is:

$$\beta(k) = 1 - \frac{\Delta t}{T_{max} - k\Delta t}$$

After taking into account the boundary conditions, the solution for the problem will be a path through our matrix, which has a minimal J at 0, but also satisfies x = 0 at n, and x = d, $v = v_{max}$ at 0.

We opted for coding in C++, and the code can be found in Appendix 1. The following plot shows optimal breaking:

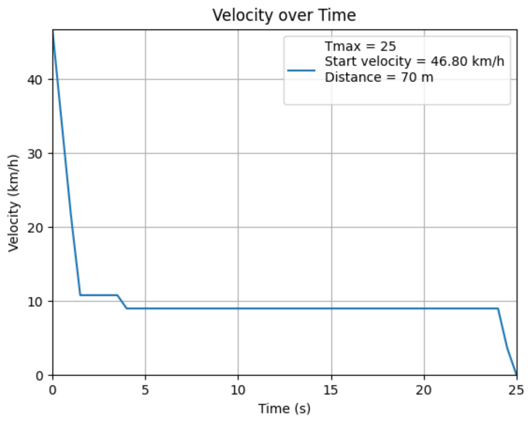


Figure 3: Graph of velocity over time

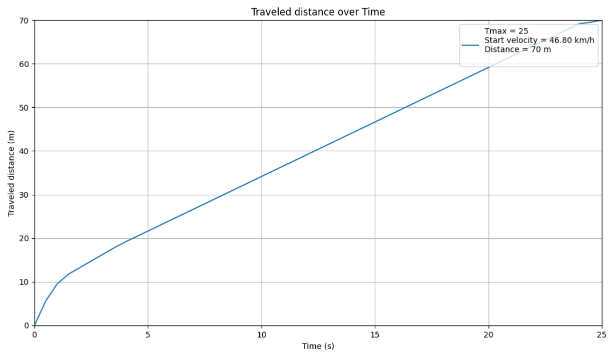


Figure 4: Graph of traveled distance over time

Translating this into language everyone can understand: the optimal strategy is to lower your velocity to about 10 km/h and drive like that until the traffic light. If you notice that by the time you arrive the light will still be red, then just break.

It's important to keep in mind that this optimizes energy loss (which is why it could possible be slightly counter-intuitive). If we wanted to also work on time optimization, then this problem would be much more complex. Still, the solution exists, but it is much more complicated [4].

4 Task 3

The situation for the problem in this task goes as follows: one lane with multiple vehicles, that the driver can classify only roughly, without the possibility of overtaking them.

First important thing to note is that the acceleration of the whole fleet is equal to the acceleration of its weakest vehicle as far as the driver that is last in the fleet is concerned.

Our idea is to approach this problem similarly to how we did for Task 2, using dynamic programming. We want to do that by adding additional condition that vehicles can't crash. This condition can be expressed as: When t > T, where T is starting time until light turns green, and t is the current time

$$x(t) < d + a_{min}\frac{(T-t)^2}{2}$$

We have solved for all different accelerations and the following plots are results:

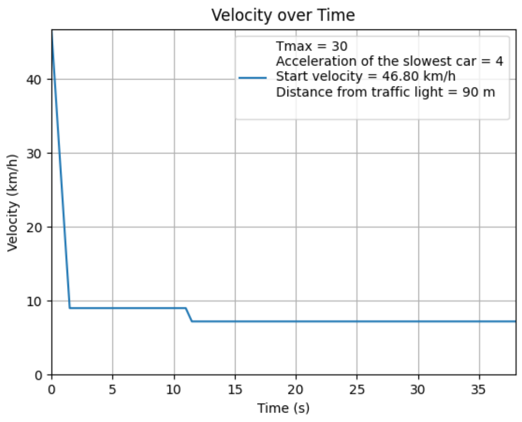


Figure 5: Graph of velocity over time for average vehicles (a=4m/s)

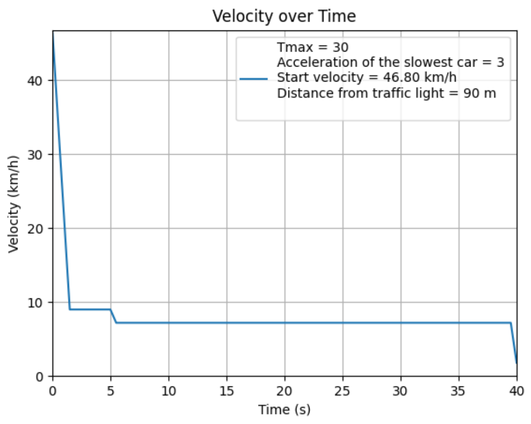


Figure 6: Graph of velocity over time for slower vehicles (a=m/s)

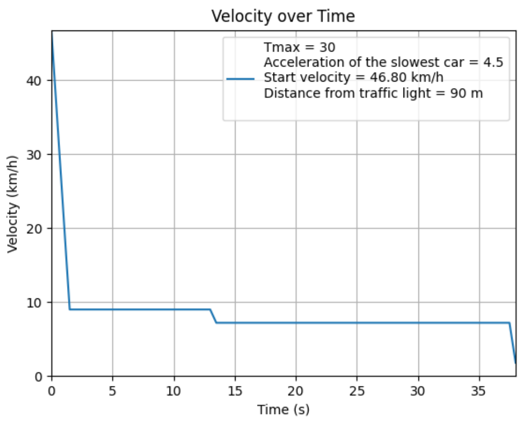


Figure 7: Graph of velocity over time for faster vehicles (a=4.5m/s)

This means that the optimal strategy is to drive at approximately 10km/h, same as in Task 2, unless you are in the situation where you will not crash into vehicles in front, then you can accelerate.

5 Task 4

5.1 Task 4.1

Up until now we were only looking at situations where there was a single lane on the road. There is, however, a possibility that the optimal strategy will change as the number of lanes does. That is exactly what we are being asked to examine in Task 4.

If we are taking into account only our loses, the strategy can be boiled down to picking out the best lane. The optimal one for us to choose is the lane in which we can get back to our maximal velocity in the shortest amount of time.

We will approach this problem by defining a parameter D, which we will use to compare the lanes one to another, thus picking out the optimal one. The "optimality" of a line is dependent on the number of vehicles and their types (including their features such as maximum velocity and acceleration).

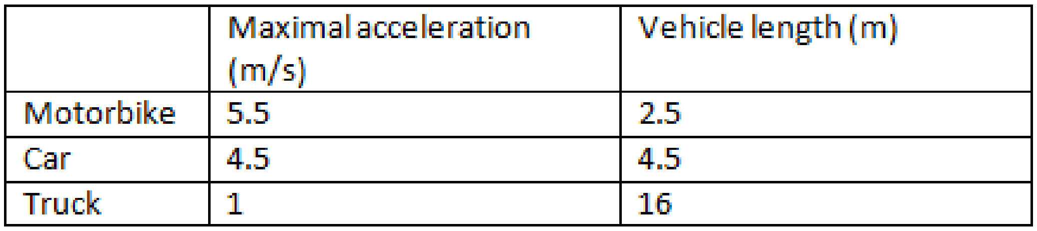
The following notation will be used: let l be the length of the fleet of vehicles, and n the number of vehicles in it. Let's also define array a, where ai represent the maximal acceleration of vehicle i.

D is calculated as follows:

$$D = \begin{cases} n\Delta t + \frac{v_{max}}{a_{min}} & l < \frac{a_{min}}{2}\left(\frac{v_{max}}{a_{min}}\right)^2, \\ n\Delta t + \frac{v_{max}}{a_{min}} + \frac{l - \frac{a_{min}}{2}\left(\frac{v_{max}}{a_{min}}\right)^2}{v_{max}} & \text{otherwise.} \end{cases} \quad (1)$$

Although this gives us an exact value for D, thus giving us an answer as to what lane should be picked, it still isn't perfect due to limitations of human brains.

Plugging in real-world values (see Figure 5), we can clearly see that it is no easy task for a driver to perform given calculations and choose a lane ([5], [6], [7], [8], [9], [10], [11])



	Maximal acceleration (m/s)	Vehicle length (m)
Motorbike	5.5	2.5
Car	4.5	4.5
Truck	1	16

Figure 8: Table with data used

Our approach to adapt this so that it can be easy to decide is to assign a value to each type of the vehicle, so that the driver can choose the lane with the smallest sum of those values. We came up with this easy-to-follow formula that would assign values to each vehicle. Motorbike is worth 1 point, car is worth 2, and trucks and similar vehicles are worth 6 points. Also, because the number of vehicles should be taken into account, to the sum of points for vehicles we should add n/2. Also, it would be beneficial to add another factor, that we'll call Γ, which will represent the number of points you should add based on what is the "worst" vehicle in the fleet (trucks are deemed worse than cars because they are larger and their acceleration ability isn't as good). So, it would look something like this, where M, C, T are, respectively, number of motobikes, cars and trucks:

$$Points = \frac{n}{2} + M + 2C + 6T + \Gamma$$

where

$$\Gamma = \begin{cases} 0 & \text{only motorbikes are in the lane} \\ 1 & \text{only cars and motorbikes are in the lane} \\ 11 & \text{there are trucks or similar vehicles in the lane.} \end{cases}$$

From (1), we arrived to these values for Γ because $\frac{v_{max}}{a_{min}}$ does not depend on the number of vehicles, just on their characteristics. By plugging in values for each of the vehicle types (and saying that the average velocity is 50 km/h), we came to those numbers. When it comes to the part of the formula M + 2C + 6T, it is important to note that the third summand in (1) is a function of length. So, by looking at the proportions of vehicle lengths, we figured out the numbers.

5.2 Task 4.2

The non greedy approach to picking a line, which aims to minimize the average expected losses per drive, can be boiled down to choosing a lane that would minimize

$$\sum_{lanes} D$$

Adding a vehicle to an empty lane as opposed to one that already contains vehicle will always increase the given sum, beacuse:

$$\Delta t + \frac{v_{max}}{a_{min}} > \Delta t + (\frac{v_{max}}{a_{min}} - \frac{v_{max}}{a_{max}}) + \frac{l_{max}}{v_{min}}$$

That is why there should always be an empty lane. It is important to note that there is no need for leaving multiple lanes empty, as one is enough.

It is time to define rules for picking a non-empty lane.

- 1) Always choose the lane whose acceleration is the same as yours, otherwise pick one with a lower one.
- 2) Always choose the lane with a fleet of shorter length.

6 Task 5

Taking into account everything we know about the current state of the traffic system, we can agree that the situation isn't ideal. That calls for change in order to improve something that affects the lives of many. Our strategy can be divided in three sections: detection, calculation, communication.

6.1 1. Detection

Everything that helps gather data on the current situation in traffic can be categorized as detection. Using different sensors and cameras, we can better understand the intricacies of traffic.

- 1. Video cameras (with computer vision) - special cameras could be installed in order to track the vehicles and pedestrians. We can analyze traffic patterns and predict situations that are about to happen.
- 2. Infrared sensors - humans play an integral part on the street, not only as drivers, but as pedestrians and bicycle riders. In situations such as night or fog, it may prove to be rather difficult to detect them. On the other hand, that is very important in order to prevent accidents. That is why we see a potential in implementing infrared sensors, which work by detecting the heat radiation that changes over time and space as humans (and animals) move [3].

- 3. Acoustic sensors - we deeply believe that besides minimizing the overall cost, we should also focus on minimizing accidents and their results. That is why we believe acoustic sensors should be implemented. This way, we could detect sounds of emergency vehicles (police, firefighters) and allow them to pass more quickly.

6.2 Calculations

In order to correctly calculate, we need to have data to put into formulas presented in tasks above. We think that the use of some machine learning models or AI could be beneficial here. To start of, creating a good vehicle identification model (or using an existing one) is key. This way, not only will we be able to calculate the number of vehicles on the road, but by determining the exact model, we would get access to a lot of useful data for each one, such as its velocity, acceleration, mass, dimensions... Another thing we identified as useful is the creation od machine learning models that would analyze traffic patterns for a specific intersection. By understanding those repeating patterns, we can better understand and predict occurrences, which would lead to better instructions for drivers.

6.3 Communication

Communication would present all of the ways of getting any information to drivers, whether its to each driver specifically, or to all at the same time. We identify three ways of doinf that:

- Displays placed 200 m before traffic lights - they would display some of the more general data that could possibly aid drivers in making better decisions, such as whether or not to start slowing down or change the lane.
- Displays for each lane individually - these would present data specific to that lane, such as when to start breaking/accelerating, or give signals to change the lane
- V2I - Vehicle to Infrastructure communication allows vehicles to receive data from traffic lights in real time, that way allowing for driving algorithms to be implemented, thus reducing stop-go situations. This system is good because it overcomes the constraints of human brains because their vehicle would be the ones doing all the calculations.

7 Further research opportunities

As with every model, ours has some limitations. Modelling real-life events is difficult, mostly because they can't always be described using functions and formulas, or they just don't take everything into account. For example, internal loses weren't taken into account, but could still have some effect. Another thing is that we classified vehicles in a rather rough way, meaning that we generalized specifications of all cars and all larger vehicles as trucks etc. Those are all ways in which our solutions could be further improved.

When it comes to further research opportunities, we would like to look at some specific case scenarios, such as borders, or traffic jams during peak times of the day. It would be interesting to find a better solution to those situations because they imply congestion of traffic.

Also, looking further into real-world solutions (Task 5) and how they can really be implemented can lead to some useful discoveries. By developing AI-driven predictive models (that are based on multi-sensor data) we can improve intersection control by anticipating congestion and dynamically adjusting signal timings.

8 Appendix 1

```

#include<bits/stdc++.h>
#define ll long long
const double dt = 0.5;
const double dx = 0.125;
const double du = 1;
const double dv = du*dt;
const double tmax = 40;
const double xmax = 100;
const double umin = -7;
const double umax = 0;
const double vmax = 15;
using namespace std;

double J[90][35][1000];
pair<int,int> back[90][35][1005];

double phi(double v1, double u)
{
    return u*dt*(v1+0.5*u*dt);
}

double beta(int k)
{
    return 1 - dt / (tmax - k*dt);
}

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(0); cout.tie(0);
    vector<double> v, x, u;


    for(double i = 0; i <= vmax; i += dv) v.push_back(i);
    for(double i = umin; i <= umax; i += du) u.push_back(i);
    for(double i = 0; i <= xmax; i += dx) x.push_back(i);

    int n = tmax / dt - 1;


    for(int i = 0; i < v.size(); i++)
    {
        for(int j = 0; j < x.size(); j++)
        {
            if(j == 0) J[n][i][j] = 0;
            else J[n][i][j] = INT_MAX;
        }
    }

    double newj = 0;


    for(int k = n - 1; k >= 0; k--)
    {
        for(int j = 0; j < x.size(); j++)
        {
            for(int i = 0; i < v.size(); i++)
            {
                J[k][i][j] = INT_MAX;
                for(double a = umin; a <= umax; a += du)
                {
                    if(x[j] - 0.5 * dt * dt * a + v[i] * dt < 0 || v[i] + a *
                       dt < dv || v[i] + a * dt > vmax || x[j] > 71)
                        continue;

                    int v1 = max(0, (int)((v[i] + a * dt) / dv));
                    int x1 = max(0, (int)((x[j] - 0.5 * dt * dt * a - v[i] * dt
                       ) / dx));

                    newj = J[k+1][v1][x1] * beta(k) + phi(v[i], a);

                    if(J[k][i][j] > newj)
                    {
                        J[k][i][j] = newj;
                        back[k][i][j] = make_pair(v1, x1);
                    }
                }
            }
        }
    }

    pair<int,int> p = make_pair(9 / dv, 50 / dx);


    for(int i = 0; i < n; i++)
    {
        cout << v[p.first] << " " << x[p.second] << endl;
        p = back[i][p.first][p.second];
    }

    cout << v[p.first] << "\n" << x[p.second] << "\n---\n";
}
```

9 Appendix 2

```
    #include<bits/stdc++.h>
#define ll long long
const double dt = 0.5;
const double dx = 0.125;
const double du = 1;
const double dv = du*dt;
const double tmax = 35;
const double xmax = 100;
const double umin = -7;
const double umax = 0;
const double vmax = 15;
const double A = 3;
const double t = vmax/A;
using namespace std;
double J[90][31][2000];
pair<int,int> back[90][35][1005];
double phi(double v1, double u)
{
    return u*dt*(v1+0.5*u*dt);
}
double beta(int k)
{
    return k/k;
}
int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(0);cout.tie(0);
    vector<double>v,x,u;
    for(double i=0;i<=vmax;i+=dv) v.push_back(i);
    for(double i=umin;i<=umax;i+=du) u.push_back(i);
    for(double i=0;i<=xmax;i+=dx) x.push_back(i);
    int n=(tmax+t)/dt;
    int n1=tmax/dt;
    cout<<tmax-t<<endl;
    for(int i=0;i<v.size();i++)
    {
        for(int j=0;j<x.size();j++)
        {
            if(j==0) J[n][i][j]=0;
            else J[n][i][j]=INT_MAX;
        }
    }
    double newj=0;
    for(int k=n-1;k>=0;k--)
    {
        for(int j=0;j<x.size();j++)
        {
            for(int i=0;i<v.size();i++)
            {
                J[k][i][j]=INT_MAX;
                if(k<n1){
                for(double a=umin;a<=umax;a+=du)
                {
                    if(x[j]-0.5*dt*dt*a+v[i]*dt < 0 || v[i]+a*dt < dv || v[i]+a
                       *dt>vmax)continue;
                    int v1=max(0,(int)((v[i]+a*dt)/dv));
                    int x1=max(0,(int)((x[j]-0.5*dt*dt*a-v[i]*dt)/dx));
                    newj=J[k+1][v1][x1]*beta(k)+phi(v[i],a);
                    if(J[k][i][j]>newj)
                    {
                        J[k][i][j]=newj;
                        back[k][i][j]=make_pair(v1,x1);
                    }
                }
                }
                else{
                    for(double a=umin;a<=umax;a+=du)
                    {
                    if(x[j]-0.5*dt*dt*a+v[i]*dt < 0 || v[i]+a*dt < dv || v[i]+a
                       *dt>vmax || x[j]<vmax*dt*k-0.5*A*dt*dt*k*k)continue;
                    int v1=max(0,(int)((v[i]+a*dt)/dv));
                    int x1=max(0,(int)((x[j]-0.5*dt*dt*a-v[i]*dt)/dx));
                    newj=J[k+1][v1][x1]*beta(k)+phi(v[i],a);
                    if(J[k][i][j]>newj)
                    {
                        J[k][i][j]=newj;
                        back[k][i][j]=make_pair(v1,x1);
                    }
                    }
                }
            }
        }
    }
    pair<int,int>p=make_pair(13/dv,90/dx);
    for(int i=0;i<n;i++)
    {
        cout<<(x[p.second])<<" "<<(v[p.first])<<endl;
        p=back[i][p.first][p.second];
    }
    cout<<70-x[p.second]<<" "<<n*dt<<"\n---\n";
}
```

Literature

[1] D. Schrank, B. Eisele, T. Lomax, and J. Bak, “2015 urban mobility scorecard,” Texas AM Transportation Institute and INRIX, Tech. Rep., 2015.

[2] A. Lawitzky, D. Wollherr and M. Buss, "Energy optimal control to approach traffic lights," 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Tokyo, Japan, 2013, pp. 4382-4387, doi: 10.1109/IROS.2013.6696985. keywords: Trajectory;Switches;Vehicles;Optimal control;Fuels;Acceleration;Optimization,

[3] www.infratec.eu. (n.d.). Infrared Sensor. [online] Available at: https://www.infratec.eu/sensor-division/service-support/glossary/infrared-sensor/.

[4] Meng, Xiangyu and Cassandras, C.G.. (2018). Optimal Control of Autonomous Vehicles Approaching A Traffic Light. 10.48550/arXiv.1802.09600.

[5] Acceleration of a Car - The Physics Factbook. [online] Available at: https://hypertextbook.com/facts/2001/MeredithBarricella.shtml.

[6] Vehicle Acceleration and Braking Parameters. [online] Available at: https://copradar.com/chapts/references/acceleration.html.

[7] Yang, G., Xu, H., Wang, Z. and Tian, Z. (2016). Truck acceleration behavior study and acceleration lane length recommendations for metered on-ramps. International Journal of Transportation Science and Technology

[8] How Fast Do Motorcycles Accelerate? [Motorcycle 0-60 Times!] l. [online] Available at: https://powersportsguide.com/motorcycle-acceleration/.

[9] CarRoar. (2020). Average Car Length Guide (An Exhaustive List Based on Car Types). [online] Available at: https://carroar.com/car-length/.

[10] What are the Sizes of Motorcycles? [Motorcycle Dimension Chart] l. [online] Available at: https://powersportsguide.com/average-motorcycle-dimensions/.

[11] ResearchGate. (2024). Table 1 . Maximum legal dimensions and weights for common grain trailer... [online] Available at: https://www.researchgate.net/figure/Maximum-legal-dimensions-and-weights-for-common-grain-trailer-configurations-in-Brazil$_{t}bl1_{2}$88547880